THE CORE AND THE PERIPHERY IN DISTRIBUTED AND SELF-ORGANIZING INNOVATION SYSTEMS

by

KARIM R. LAKHANI

B.ENG.MGT. Electrical Engineering and Management
McMaster University, 1993

S.M. Technology and Policy
Massachusetts Institute of Technology, 1999

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JAN 25 2006
LIBRARIES

Submitted to the Alfred P. Sloan School of Management
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

at the

Massachusetts Institute of Technology

February 2006

© 2006 Karim R. Lakhani. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Signature of Author: _____
MIT Sloan School of Management
January 13, 2006

Certified by: _____
Eric von Hippel
Professor of Management
Chair, Technology, Innovation and Entrepreneurship Group
Thesis Supervisor

Accepted by: _____
Birger Wernerfelt
Professor of Management Science
Chair, PhD Program, Sloan School of Management

ARCHIVES

[This page intentionally left blank]

THE CORE AND THE PERIPHERY IN DISTRIBUTED AND SELF-ORGANIZING INNOVATION SYSTEMS

by

KARIM R. LAKHANI

Submitted to the Alfred P. Sloan School of Management
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Management

Abstract

The internet has enabled the large-scale mobilization of individuals to self-organize and innovate outside of formal organizations. My dissertation consists of three studies examining the functioning of such self-organizing and distributed innovation systems. I focus on the differing roles of core and peripheral participants in the distributed innovation process and explore the potential generality of this new form of innovating.

The first study explores a systematic method of broadcast search used by corporations to search for solutions to internal R & D problems by involving peripheral problem solvers – those who are outside of their organizations. I find that innovative solutions to difficult scientific problems can be effectively identified by broadcasting problems to a large group of diverse solvers in different fields. Broadcast search yields innovative solutions by peripheral solvers who are crossing scientific disciplines. The central characteristic of problems that were successfully solved is the ability to attract specialized peripheral solvers with heterogeneous scientific interests.

The second study examines how participants jointly innovate in a Free and Open Source Software community. I find that members at the periphery – those outside of the core project team – are responsible for developing a majority of functionally novel software features. In contrast, core members develop performance-related features. Peripheral members also initiate the majority of the development activity and provide critical input into the technical problem solving processes. Ongoing interactions between core and peripheral members are the primary enablers of collective problem solving. I discuss how core and peripheral members enact six work practices in jointly producing software in a distributed and virtual setting.

The third study examines the motivation of core participants in 287 Free and Open Source Software communities. Theorizing on individual motivations for participating in communities has posited that external motivational factors in the form of extrinsic benefits as the main drivers of effort. I find that enjoyment-based intrinsic motivation, namely how creative a person feels when working on the project was the strongest and most pervasive driver of effort. I also find that user need, intellectual stimulation derived from writing code, and improving programming skills as top motivators for project participation.

Thesis Supervisor: Eric von Hippel
Title: Professor of Innovation Management

Thesis Committee:

Eric von Hippel (Chair)
Professor of Innovation Management
Chair, Technology, Innovation and Entrepreneurship Group
MIT Sloan School of Management

Wanda J. Orlikowski
Eaton-Peabody Chair of Communication Sciences
Professor of Information Technology and Organization Studies
MIT Sloan School of Management

Thomas J. Allen
MacVicar Faculty Fellow
Howard W. Johnson Professor of Management
MIT Sloan School of Management

"THE DISUNITY OF THE SCHOLARS IS A MERCY FOR THE NATION"
- attributed to Prophet Mohammed (PBUH)

ROOMS ARE NEVER FINISHED

- Agha Shahid Ali

FOR PETAL & BEE

[This page intentionally left blank]

Table of Contents

List of Tables 12
List of Figures 14
Acknowledgements 15
Chapter 1: Introduction and Overview 18
References 22
Chapter 2: Broadcast Search in Scientific and Technical Problem Solving: Finding Solutions from the Periphery 23
2.1: Introduction 23
2.2: Problem Solving and Local Search 25
2.2.1: Problem solving with experience for Individuals and Teams 26
2.2.2: Local Search in Firms 28
2.3: Innovation and External Knowledge 29
2.3.1: People and Capability for Integrating External Knowledge 30
2.3.2: Alternatives to Local Search: Alliances and Mobility as Means of Accessing External Knowledge 33
2.4: Broadcast Search 35
2.4.1: Distributed and decentralized problem solving and the importance of the periphery 35
2.4.2: Defining Broadcast Search 39
2.4.3: Overcoming Newton's Folly By Broadcast Search 40
2.4.4: Contemporary Research on Broadcast Search 41
2.5: Setting and Data Sources 42
2.6: Findings 45
2.6.1: Broadcast Search Performance and Solver Base Profile 45
2.6.2: Problem Solving Process Used By Solvers 49
2.6.3: Determinants of a Solver Creating a Winning Solution 56
2.6.4: Determinants of a Problem Being Successfully Solved 61
2.7: Discussion 66
2.7.1: Cross-fertilization of Scientific Fields 66
2.7.2: Attracting Many Heterogeneous Solvers 69
2.7.3: Solver Motivations to Participate 70
2.7.4: No One Genius Solver 70
2.7.5: Seeker Learning and Structuring Ill-Structured Problems 71
2.7.6: Value to Seekers 72
2.8: Implications 74
2.8.1: Implications for Distributed Problem Solving 74
2.8.2: Implications for Organizations 77
2.8.3: Implications for Science 79
2.9: Future Research 81
References 84
Appendix 91
Chapter 3: The Primacy of the Periphery in a Distributed Problem Solving Community 105
3.1: Introduction 105
3.2: Defining Community 107

3.2.1: Traditional Communities 107
3.2.2: Instrumentally Oriented Communities 109
3.2.3: Communities of Practice 111
3.2.4: Distributed Problem Solving Communities 114
3.3: The Periphery and its Value to the Community 118
3.3.1: Defining the Periphery 120
3.3.2 The Periphery as the Locus of Innovation and Source of Novelty 122
3.3.3 Defining core and periphery in F/OSS communities 126
3.4: A Practice View of Distributed Problem Solving Communities 128
3.5: Research Setting, Data and Methods 132
3.5.1: Research Setting 132
3.5.2: Data and Methods 136
3.6: The Primacy of the Periphery 142
3.6.1: Community participation structure 142
3.6.2: Contribution to software development and problem solving process 144
3.7: Vignettes of Distributed Core-Periphery Problem Solving 148
3.7.1: Vignette 1 – Periphery Members Develop Novel Feature 148
3.7.2: Vignette 2 – Core Member Improves Existing Feature 161
3.7.3: Vignette 3 – Joint Problem Solving and New Feature Creation by Core and Peripheral members 173
3.8: Distributed Problem Solving Practices in the PostgreSQL Community 188
3.8.1: Collective Practice 1 - Work Broadcasting 191
3.8.2: Collective Practice 2 – Building and Using Community Memory 197
3.8.3: Collective Practice3 – Distributed Decision Making 201
3.8.4: Individual Practice 1 – Choosing type and level of participation 207
3.8.5: Individual Practice 2 – Using The Community's Output 212
3.8.6: Individual Practice 3 - Coordinating Action and Building Trust Through Evidence 215
3.9: Discussion 218
3.10: Implications and Conclusion 226
3.10.1: Implications for Innovation and Product Development 227
3.10.2: Implications for Organization Theory 228
References 233
Appendix 240
Chapter 4: Motivation and Effort of Core Developers in Open Source Communities ... 250
4.1: Introduction 250
4.2: Literature Review - Motivations to Participate 251
4.2.1: Intrinsic Motivation 251
4.2.2: Extrinsic Motivation 255
4.2.3: Current findings on motivations in F/OSS projects 257
4.3: Research Methods and Sample Characteristics 259
4.3.1: Sample Selection 259
4.3.2: Data Collection 260
4.3.3: Sample Characteristics 261
4.4: Findings - Payment Status and Effort in Projects 262
4.5: Findings - Creativity and Motivation in Projects 265

4.6: Findings: Determinants of Effort....273
4.7: Discussion and Conclusion....280
References....282
Appendix....284
Chapter 5: Summary and Conclusion....292
5.1: Overview of the empirical studies....293
5.1.1: Study 1 – "Broadcast Search and Solution Finding from the Periphery"....293
5.1.2: Study 2 – "The Primacy of the Periphery in Open Source Software Development"....298
5.1.3: Study 3 – "Motivations of Core Developers to Contribute to Open Source Projects"....302
5.2: Distributed Information and Knowledge and the Value of the Periphery....304
5.3: The Advantage of the Periphery....306
5.4: Heterogeneity in Motivations to Participate in Distributed Innovation Systems....309
5.5: Generalizable Conditions for Distributed and Self-Organizing Innovation Systems....311
5.5.1: Entry and Participation....311
5.5.2: Decomposable and "Well-Structured" Problems....314
5.5.3: Solution Generation....316
5.5.4: Substitutes for "Trust" and "Management"....318
References....321

List of Tables

Table 2.1: Overall Performance Of Broadcast Search By Scientific Disciplines 47
Table 2.2: Profile Of Solver Base.....50
Table 2.3: Respondents' Familiarity With The Problem 50
Table 2.4: Respondent Mapping of Problem And Their Own Field Of Expertise.....52
Table 2.5: Respondents' Solution Information Knowledge Upon Seeing The problem ..52
Table 2.6A and 2.6B: Source Of Solution Information Used by Winning Solvers in Their Submission.....54
Table 2.6C: Source And Amount of Modification Of Prior Solutions In Creating Present Solution By Winning Solvers 55
Table 2.7: Motivations To Participate in Broadcast Search Problem Solving, Scores and Factor Loadings 57
Table 2.8: Correlations Between Variables Predicting Solver Being A Winner (N = 295 Respondents) 58
Table 2.9: Probit Regression Predicting Which Solver Submits A Winning Solution 60
Table 2.10: Correlations Between Variables Predicting Problem Being Solved 63
Table 2.11: Probit Regression On Problem Being Solved 64
Table 3.1: Subtypes of Communities 117
Table 3.2: Release History of PostgreSQL Software 135
Table 3.3: Community Structure during the PG 7.4 Release Cycle (November 2002 - November 2003).....144
Table 3.4: Core Members Dominate Email Message Traffic.....144
Table 3.5: Periphery Drives Community Discussion 144
Table 3.6: Feature Type Author by Member Status 145
Table 3.7: Source of Initiation of Feature by Community Member Status 146
Table 3.8: Role of Periphery in Community Problem Solving 147
Table 3.9: Code Developed by Periphery Arrives "Pre-Made" 147
Table 4.1: General Characteristics of Survey Respondents 264
Table 4.2: Location and Work Relationship for F/OSS Contributions 264
Table 4.3: Hours Spent / Week on F/OSS Projects 264
Table 4.4: Creativity in F/OSS Projects 266
Table 4.5: Responses to "Flow" questions 266
Table 4.6: Motivation to contribute to F/OSS projects 268
Table 4.7: Project Topics By User Need 268
Table 4.8: Cluster results based on motivations and paid status 272
Table 4.9: OLS Regression on Log Project Hours/Week.....275
Table 4.10: Impact of Significant Variable on Project Hours/Week (Standardized Coefficients) 278
Table 4.11: Fixed Effects OLS Regression on Log Project Hours/Week 279
Table 5.1: Probit Regression on Problem Being Solved 296
Table 5.2: Probit Analyses Predicting Which Solver Submits A Winning Solution 297
Table 5.3: Feature Type Author by Member Status 300
Table 5.4 : Role of Periphery in Community Problem Solving 300
Table 5.5: Practices for Community-Based Distributed Problem Solving.....301
Table 5.6: Motivation to contribute to F/OSS projects 303

Table 5.7: Impact of Significant Variable on Project Hours/Week (Standardized Coefficients) 303

List of Figures

Figure 3.1: Skewed Participation in Linux Kernel Email Discussion 119
Figure 3.2: Graphic of Performance Improvement Submitted by Matthew O'Connor... 157

Acknowledgements

I have many people to thank for helping me on my intellectual journey as a doctoral student at MIT. This dissertation would not have been possible without the attention, care, concern, feedback, ideas, and love of my colleagues, friends, family and mentors. I am constantly grateful for the fact that I have been lucky enough to be have been graced by such wonderful people who have willingly shared their best with me. I would also like to acknowledge the generous financial support of Canada's Social Science and Humanities Research Council Doctoral Fellowship.

First I would like to thank my colleagues from the open source community for generously sharing their experiences and lessons. Jeff Bates, Brian Behlendorf, Chris DiBona and Luis Villa were always available when I had a question or wanted an explanation about something. Bruce Momjian and Neil Conway were a tremendous help in my study of the PostgreSQL community. I would not have been able to complete this dissertation without their input and guidance. Stefano Mazzocchi played a critical role in helping me understand the patterns of interactions I had observed in open source communities. The distributed problem solving practices that I outline in Chapter 3 came about through intense discussions with him. Ben Hyde claims that he is the open source "native" that I have captured and now tour around the halls of the Academy. Far from being a naïve native, Ben has been a real chum in helping me think through the intricacies of open source and to be an observer of behavior that I would have mostly missed on my own. It is always a treat to chat with Ben!

Jill Panetta and Peter Lohse from InnoCentive.com have been very gracious with their time and ideas. Chapter 2 in the dissertation is a testament to their hard work in building a new model for innovation. I was delighted when they agreed to my proposal for a study on their company and appreciated all of their work and insights as we collected the data. Lars Bo Jeppesen also played an important role in helping to formulate the research question and data collection for Chapter 2. Our path's crossed when he was a visitor at MIT and we have struck a fun and friendly research collaboration.

A special note of thanks to my wonderful colleagues at The Boston Consulting Group. I never thought it possible that I would be able to be in a doctoral program and keep working at BCG. I deeply appreciate their sponsorship of my work. Mark Blaxill's leadership of BCG's Strategy Practice Initiative enabled me to keep pursuing my interest in open source from both the academic side and the strategy side. He was always a strong supporter of my work and I have to thank him for pushing through BCG such an unconventional relationship. Philip Evans provided me with significant intellectual challenges and food for thought. Many times I would think of an issue one way and Philip would completely invert it and change my perspective. It was a pleasure to work with Emily Case. She was a source of immense calm as both of us were fighting our way through UCINET and social networks. Finally through this process Bob Wolf has become a great friend and intellectual partner. Bob and I connected on intellectual issues from day one and we have had a ton of fun exploring new ideas and generally going

against conventional wisdom at BCG. Bob was also a co-conspirator in getting the data for Chapter 4.

Many Professors at MIT and Harvard have played an instrumental role in shaping my intellectual trajectory. They serve as great role models for patience, intelligence and scholarship. In particular I would like to thank Stephen Ansolabehere, Lotte Bailyn, Carliss Baldwin, Pablo Boczkowski, Michael Cusumano, Roberto Fernandez, Rebecca Henderson, Lee Fleming, Alan MacCormack, Fiona Murray, Siobhan O'Mahony, Ed Roberts, Stefan Thomke, Jim Utterback, Jesper Sørensen, and Ezra Zuckerman. I also had a lot of fun and learned a lot through individual reading classes with Paul Carlile and John van Maanen. They had a tremendous influence in me developing a taste for qualitative and sociological analysis. Leslie Perlow and her founding of the QUIET group at Harvard bolstered my confidence in doing both qualitative and quantitative analyses.

Fellow students at Sloan made this journey fun and exciting. Their support in coping with the minutiae of problem sets and assignments, the stress of the dreaded general exam, picking a dissertation topic, and tearing apart bad ideas was essential in keeping my sanity. I would like to thank the following for their friendship and kindness through this long process: Lourdes Sosa, Joao Cunha, Maria Quijada, Ruthanne Huising, Kate Kellogg, Sarah Kaplan, Tony Briggs, Rodrigo Canales, Nicola Lacetera, Kevin Boudreau, Luis Vives Prada, Steve Kahl, Ethan Mollick, and Sung Joo Bae. I would also like to acknowledge two chums, Rafel Lucea and Ramana Nanda, for always being there and helping to co-found the "T"- Club and the Quorum.

Jen Cohen, Wick Sloane and Bob Dunham convinced me that doing a PhD was the right thing to do and provided me with the courage and the tools to pull it off. I am very appreciative of their support, encouragement and the care they took by making my concerns their own concerns. Roy Bivens, my best friend, put up with a lot of missed phone calls and conversations. He always has been a strong supporter and could understand why I wanted to go the academic route.

One of the distinguishing features of my dissertation committee is that all three members have entries profiling their careers and highlighting their scholarship in Wikipedia.org. It has been a real privilege to work with and learn from Tom Allen, Wanda Orlikowski and Eric von Hippel. Tom encouraged me to think beyond the specifics of open source and to connect my research with the broader literature on technical problem solving. His exhortations to generalize my findings and to speak to a broader audience still resonates with me. Wanda's doctoral seminar on IT and Organizations was my favorite formal class in my doctoral studies. I was thrilled when she accepted my invitation to join the dissertation committee and I sincerely appreciate her patient input and guidance on this dissertation. Tom and Wanda are tremendous role models in how to produce careful, considered and ground breaking scholarship.

Eric von Hippel, once again, has played the most important role in my intellectual maturation. It is because of Eric that I endeavored to do the PhD. As a doctoral student

Eric provided me with tremendous latitude to explore many intellectual domains and to carve out my own particular niche. I aspire to have his skills for identifying interesting research questions and developing studies that provide clear answers to research problems. He has tried to train me to find parsimonious and sophisticated explanations for complex phenomenon. Eric has been a true Doktor-Father to me. He was always available for advise and counsel on both professional and personal matters. Even though we still compete on matters of diets and weight, I truly value his concern for my personal and professional success and well-being. I look forward to our continued collaboration as we set an academic agenda on distributed innovation.

I have to thank my family members for their love and support throughout this journey. I suffered a great loss when my grandmother passed away in the second year of the PhD program. NaniMa was a pillar of faith and encouragement for me. She encouraged my studies and was very proud that her grandson was at MIT. She constantly prayed for my success, easement of difficulties and provided me with tremendous moral support. I miss her terribly. My sister Diloo was full of enthusiasm and encouragement during the PhD process. Even though she lives more than 5000 miles away, I could always count on her to say the right things and to make my day. My Mom has been a role model for academic and personal accomplishment. It was her sacrifices, early on in my life, that provided our family with the ability to move to the West and to get an education and a better life. More recently, she selflessly devoted herself to help Shaheen and me with our new-born daughter. Funny enough, Mom always wanted her kids to be teachers like her, and in our own way, both my sister and I are following in her footsteps. I can't thank her enough.

The completion of this dissertation also marks a significant milestone in my relationship with my beautiful wife and friend, Shaheen. We met as freshmen at McMaster University in 1988 and have been in a committed relationship since 1989. Between us we have collected three undergraduate degrees, two masters degrees, and now two doctorates. I am thrilled that our formal academic training is now finally over! Shaheen has been through all the ups and downs of the PhD with me and always offered her unconditional love. I could not have done this without her. In my third year of doctoral we achieved another milestone in our relationship with the birth of our amazing daughter, Sitarah Noor. "Situ" has been a source of tremendous joy in our lives. Watching her grow and learn brings new meaning to the word "thrilling." ***Petal and Bee make all of this worthwhile.***

Chapter 1 - Introduction and Overview

Distributed and self-organizing innovation systems have emerged as a robust organizational form co-existing with centralized models of innovation and product development. The success of Free/Open Source Software (F/OSS) communities (e.g.: Apache, Linux, Mozilla, and Perl) has brought this model to general attention, but it is also rapidly taking hold in industries as diverse as custom integrated circuits, biotechnology, pharmaceuticals, music and software development (von Hippel 2005). Fundamental to the operation of these systems is the recognition that knowledge is heterogeneously distributed in society (Hayek 1945).

F/OSS communities are a leading example of self-organizing distributed innovation systems. Software produced by F/OSS communities has been called the "impossible public good" (Kollock 1999: 230). It has been a major surprise to researchers to learn that complex software systems, running mission critical applications, can be designed, developed, maintained and improved by a virtual "collective" of mostly volunteer computer programmers for "free." Even more surprising, some of the largest software companies and the biggest holders of intellectual property rights (e.g.: IBM, Sun, Apple, Oracle) have embraced F/OSS communities by encouraging the participation of their personnel in these communities, donating software and patents to these communities, and integrating F/OSS software in their strategic product and service offerings.

The existence and sustained success of such distributed innovation systems has not been widely anticipated by the normative and theoretical literatures on innovation and product development, organizations, and strategy. They present a unique opportunity to expand our theoretical understanding about the dynamics of the emerging distributed innovation process and its impact on organizations and strategy. This dissertation consists of three independent studies examining the functioning of such self-organizing and distributed innovation systems. In particular I focus on the differing roles of core and periphery participants in the distributed innovation process and explore the potential generality and robustness of this new form of innovating.

The dissertation is organized into five self-contained chapters that can be read independently. This chapter provides an overview of the studies and highlights the main empirical findings. Chapters 2, 3, and 4 contain empirical findings from the three studies. Chapter 5 summarizes the findings across all three studies and attempts to generalize the conditions under which distributed and self-organizing innovation systems operate.

Chapter 2[1] explores an alternative mechanism of scientific and technological problem solving that focuses on solution generation from a wide range of dispersed and peripheral problem solvers. I find that innovative solutions to difficult scientific and technical problems can be effectively identified by broadcasting problems to a large

[1] Data for this chapter was collected in cooperation with Jill Panetta and Peter Lohse of Innocentive.Com and Lars Bo Jeppesen of Copenhagen Business School.

group of diverse solvers in different fields and providing incentives for external solvers to solve them. Broadcasting problems to a group of diverse solvers is a radical departure from traditional problem solving search as it inverts the typical problem solving process by focusing the efforts of the problem holders (e.g. R&D labs) into attracting solutions from external solvers instead of creating solutions themselves and that it allows for the mitigation of some of the negative issues (e.g.: competency traps, excessive reliance on existing knowledge) associated with "local search" (March and Simon 1958; Nelson and Winter 1982; Podolny, Stuart and Hannan 1996).

I analyze 166 discrete life sciences and chemistry and applied science problems that originating in the R &D labs of 26 firms from 10 countries. These problems were broadcasted to a network of 80,000 independent solvers, for a financial reward for successful solution, from over 150 countries via InnoCentive.com, an independent subsidiary of Eli Lilly. The analysis shows that the broadcast search method yields a 29.5% solving rate for problems that well-renowned and large R & D intensive firms had not been successful in solving themselves. The central characteristic of problems that were successfully solved is the ability to attract specialized solvers with heterogeneous scientific interests. A web-based survey of 370 solvers indicated that most of the winning solvers based their submissions, partially or fully, on previously developed solutions from their own and/or someone else's work. Solutions to problems that firms seek to solve encounter their resolution in already existing (complete or partial) solutions in the distributed solvers' domains, which are then reused or transformed, showing that broadcast search effectively utilizes existing distributed knowledge.

The probability of being a winning solver is significantly correlated with both a desire to win the award money as well as intrinsic motivations like enjoying problem solving and being intellectually challenged. However, even though there was a substantial prize award for creating the best solution, the effect of intrinsic motivation is stronger and more significant.

I also find that individuals who are successful problem solvers report that the broadcasted problem was at the boundary or outside their field of expertise. This has a positive and significant effect in predicting who becomes a winning solver and may be due to the ability of "outsiders" from relatively distant fields to see problems with fresh eyes and apply solutions that are novel to the problem domain. Importantly, it implies that problem broadcasting to solvers in diverse fields triggers productive cross-fertilization of knowledge bases between scientific disciplines.

Chapter 3 examines the value of peripheral members to the software development effort in a F/OSS community. Studies of F/OSS projects have shown participation of hundreds of individuals in a core-periphery community structure with a relatively small number of core members contributing most of the software code and dominating the technical discussions (Koch and Schneider 2002; Lee and Cole 2003; von Krogh, Spaeth and Lakhani 2003). I analyzed a one year period of software development activity in the PostgreSQL database community by creating a unique analytic tool called an "innovation process history." This consisted of matching 241 concrete software features to 2,402

changes in the software source code repository and 20,129 email messages exchanged between 798 individuals.

The analysis shows that peripheral members are responsible for developing a significant majority of functionally novel software features while core members develop performance-related features. The code developed by periphery members typically solves their own local concerns. Periphery members are also responsible for initiating the majority of development activity in the community and provide critical solution, usage, and test information during the development process.

I show that ongoing interactions between core and periphery members is the primary driver of problem solving and knowledge creation in the OSS community. Specifically, the following six set of work practices enable them to produce software in a distributed and virtual setting: *work broadcasting, building and using community memory, distributed decision making, choosing type and level of participation, using the community's output* and *coordinating action and building trust through evidence.* Using vignettes from the innovation process histories, I demonstrate that these practices are not separate activities from the work of software development itself. Rather, they are embodied in the way these communities produce software, and are at the heart of core-periphery problem solving.

Chapter 4[2] examines the motivation and effort of core participants in OSS communities. "What drives F/OSS developers to contribute their time and effort to the creation of free software products?" is an often posed question by software industry executives, managers, and academics when they are trying to understand the relative success of OSS communities. Many people are puzzled by what appear to be irrational and altruistic behavior by movement participants: giving code away, revealing proprietary information, and helping strangers solve their technical problems. I used a web-based survey, administered to 684 software developers in 287 F/OSS projects, to learn what lies behind the effort put into such projects.

Academic theorizing on individual motivations for participating in OSS projects has posited that external motivational factors in the form of extrinsic benefits (e.g.; better jobs, career advancement) as the main drivers of effort (Lerner and Tirole 2002). I found, in contrast, that enjoyment-based intrinsic motivation (Csikszentmihalyi 1975; Deci and Ryan 1985; Frey 1997; Lindenberg 2001), namely how creative a person feels when working on the project (Amabile 1996), was the strongest and most pervasive driver of effort. I also found that user need (von Hippel 1988), intellectual stimulation (Nakamura and Csikszentmihalyi 2003) derived from writing code, and improving programming skills (Lerner and Tirole 2002) as top motivators for project participation.

Chapter 5 summarizes the result of the three studies and concludes with an attempt to generalize the operating conditions of distributed and self-organizing innovation systems. I discuss the theoretical underpinnings of the knowledge and information advantages for peripheral members and then conclude by considering how

[2] Data for this chapter was collected with Bob Wolf and colleagues from BCG.

entry of participants, heterogeneity in motivations, decomposable problems, diversity in search mechanisms and substitutes for trust and management form the basis for distributed innovation systems.

***

External sources of knowledge have been found to be critical for the innovation process (Cohen and Levinthal 1989; Cohen and Levinthal 1990). Research has shown that organizations typically seek knowledge from the outside and solve problems inside. My dissertation shows that in distributed innovation systems the periphery is an important source of actual solution generation activity and provides critical problem solving information. However, instead of gatekeepers (Allen 1977) seeking relevant information from the outside, peripheral members, on their own accord, deliver complete solution information.

The reliance on the periphery for innovations and solutions makes sense because of the distributed nature of knowledge and heterogeneity in skills and abilities in the general population and the sticky nature of knowledge. Overall, distributed innovation systems exhibit organizational architectures that enable the periphery to play a central role in the organization's problem solving effort. This dissertation provides a lens into understanding their operations and their extension into other spheres of innovation.

References

Allen, Thomas, J. 1977. *Managing the flow of technology*. Cambridge, MA: MIT Press.

Amabile, Teresa M. 1996. *Creativity in context*. Boulder, CO: Westview Press.

Cohen, Wesley M., and Daniel A. Levinthal. 1989. "Innovation and Learning: The Two Faces of R & D." *The Economic Journal* 99:569-596.

—. 1990. "Absorptive Capacity: A New Perspective on Learning and Innovation." *Administrative Science Quarterly* 35:128-152.

Csikszentmihalyi, Mihaly. 1975. *Beyond Boredom and Anxiety: The Experience of Play in Work and Games*. San Francisco: Jossey-Bass, Inc.

Deci, Edward L, and Richard M Ryan. 1985. *Intrinsic motivation and self-determination in human behavior*. New York, NY: Plenum Press.

Frey, Bruno. 1997. *Not just for the money: an economic theory of personal motivation.* Brookfield. VT: Edward Elgar Publishing Company.

Granovetter, M. 1973. "The strength of weak ties." *American Journal of Sociology* 78:1360-1380.

Hayek, F. A. 1945. "The use of knowledge in society." *American Economic Review* 35:519-530.

Koch, S, and G Schneider. 2002. "Effort, Cooperation and Coordination in an Open Source Software Project: GNOME." *Information Systems Journal* 12:27-42.

Kollock, Peter. 1999. "The Economies of Online Cooperation." Pp. 220-239 in *Communities in Cyberspace*, edited by Peter Kollock and Marc A. Smith. New York, NY: Routledge.

Lee, Gwendolyn, and Robert E Cole. 2003. "From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development." *Organization Science* 14:633-649.

Lerner, Josh, and Jean Tirole. 2002. "Some Simple Economics of Open Source." *Journal of Industrial Economics* 50:197-234.

Lindenberg, Siegwart. 2001. "Intrinsic motivation in a new light." *Kyklos* 54:317-342.

March, James G, and Herbert Simon. 1958. *Organizations*: Wiley.

Nakamura, Jeanne, and Mihaly Csikszentmihalyi. 2003. "The construction of meaning through vital engagement." in *Flourishing: positive psychology and the life well-lived*, edited by Corey L Keyes and Jonathan Haidt. Washington, DC: American Psychological Association.

Nelson, Richard R., and Sidney G. Winter. 1982. *An evolutionary theory of economic change*. Cambridge, MA: Belknap Harvard.

Podolny, Joel M., Toby E. Stuart, and Michael T. Hannan. 1996. "Networks, Knowledge, and Niches: Competition in the Worldwide Semiconductor Industry, 1984-1991." *American Journal of Sociology* 102:659-689.

von Hippel, Eric. 1988. *The Sources of Innovation*. New York, NY: Oxford University Press.

—. 2005. *Democratizing Innovation*. Cambridge, MA: MIT Press.

von Krogh, Georg, Sebastian Spaeth, and Karim R Lakhani. 2003. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study." *Research Policy* 32:1217-1241.

Chapter 2 – Broadcast Search in Scientific and Technical Problem Solving: Finding Solutions from the Periphery

2.1: Introduction

How are solutions to scientific and technical problems found? Problem solving usually involves search (Baron 1988; Cyert and March 1963; March and Simon 1958; Simon 1969) combined with a process of learning (Allen 1966b; Levinthal and March 1993; Marples 1961; von Hippel and Tyre 1995). Prior problem solving experience often becomes a natural starting point for a solution search to a new problem (Rosenkopf and Almeida 2003) as solvers tend to re-apply solutions and methods which were found successful in one instance to subsequent rounds of problem solving on different problems (Allen and Marquis 1964), thus resulting in a "local" search of the potential solution space. The presence of local search effects have been found at the individual level (Adamson 1952; Birch and Rabinowitz 1951; Duncker 1945; Luchins 1942) as well as at the organizational level (March and Simon 1958; Nelson and Winter 1982; Stuart and Podolny 1996), where its effect is a more rapid solution to problems similar to those experienced in the past and decreased effectiveness in solving novel problems.

Thus re-applying prior experience can yield significant increases in productivity in repetition-based manufacturing through learning curve effects (Argote and Epple 1990; Yelle 1979) but it can also lead to obsolescence in new technological domains (Sorensen and Stuart 2000). Since progress in innovation and R&D, by definition requires solutions to novel problems (Brown and Eisenhardt 1995; Clark 1985), local search-based approaches may negatively impact creative solution generation.

The purpose of this chapter is to demonstrate an alternative mechanism of technological problem solving that focuses on finding solutions from a wide range of dispersed and peripheral problem solvers. It is based on von Hayek's (1945) central insight that knowledge is unequally and widely distributed amongst individuals and the central challenge in society is to find ways to access this knowledge. I show that for

certain types of discrete scientific and technical problems, broadcasting a problem to many heterogeneous potential solvers is a highly effective means of finding solutions and that it allows for the mitigation some of the negative issues (e.g.: competency traps, excessive reliance on existing knowledge) associated with local search.

The chapter develops the concept of broadcast search as method of problem solving consisting of the following four steps: (1) problem definition, (2) problem broadcast, (3) solution attraction, and (4) solution selection. Broadcast search inverts the typical problem solving process by focusing the efforts of the problem holders into attracting "complete" solutions from external sources instead of creating solutions themselves. Problem holders transform from problem solvers to become solution seekers by defining and broadcasting problems, attracting many potential solvers from different domains and evaluating solution submissions. Broadcast search overcomes the limitation of local search by distributing the problem into different domains and by inducing many different solvers to engage in problem solving based on their own expertise and skill sets.

The application of broadcast search to 166 discrete scientific problems[3], originating from the research and development laboratories of 26 firms, shows a 29.5% solving rate with novel solutions arriving from a dispersed pool of peripheral solvers not known to the seeker firms. The problems were broadcasted as "winner-take-all" challenges with substantial financial reward (average $30,000 US) for the winning submission. Each problem attracted interest from, on average, 240 individuals and solution submissions from 10. Seekers firms decided which solution submission(s) best met their criteria, if any, and picked a winning solution.

The probability of a problem being solved was a function of the number of solvers submitting solutions and the heterogeneity in the scientific interests of the solvers. Problems that attracted more specialized solvers were also more likely to be solved. Most of the winning solvers based their submissions, partially or fully, on previously

[3] Data for this paper was collected in cooperation with Peter Lohse and Jill Panetta of InnoCentive.com and Lars Bo Jeppesen from Copenhagen Business School.

developed solutions from their own and/or someone else's work. Winning solvers also rated the problems to be at the boundary or outside their own field of expertise suggesting that broadcast search was initiating cross-fertilization amongst scientific disciplines. Past firm experience and learning with problem articulation and decomposition had a significant effect on the probability of future problems being solved. Firms using broadcast search achieved significant value creation in the process of attracting solutions form the periphery.

I begin by discussing the salient literature on problem solving and local search (section 2) and the role of external knowledge in innovation (section 3). I then develop a framework for broadcast search (section 4) and lay out the data sources and setting (section 5). I then present the empirical findings on the efficacy and functioning of broadcast search (section 6) and a discussion on their relevance (section 7). Finally, implications of the findings and further research directions (section 8 & 9) conclude the paper. The appendix to the paper consists statistical methods and the solver survey.

2.2: Problem Solving and Local Search

The seminal work of Newell and Simon and colleagues (Newell and Simon 1972; Simon and Newell 1962) has described the problem solving process as an attempt of getting from the present to desired situation by a process of "searching through a large maze." The maze depicts the problem space; the nodes of the problem space represent situations; and the paths joining one node to another are the actions that will transform one situation into another. A problem space has an initial state and a goal state and a set of means that allows a solver to move from one state to another. Problem solving is an act of stepwise search through the problem space (Dunbar, 1998) and decisions by problem solvers are taken under significant uncertainty (Simon, 1969: 68).

Newell and Simon (1972) characterize the differences in search processes in problem solving as "trial and error", "hill-climbing", and "means-end analysis". The three approaches form a succession in which the latter requires information the former

does not. Trial and error problem requires solvers to only recognize that they have reached their goal; hill climbing requires that solvers can assess the relative closeness of their position to the goal; and means-end analysis requires that the solver be able to discern the type of difference between the current state and the goal state (Baron 1988). The problem solving approach used depends on the characteristics of the problem itself, but also to a large extent on the past experience of the problem solver (March and Simon 1958: 177; Ward 1995).

2.2.1: Problem solving with experience for Individuals and Teams

Research within cognitive and social psychology adds to the above view by emphasizing the effects of solvers' past experience with a problem, focusing on the learning that occurs during problem solving and its impact on selection of future problem solving strategies (Lovett and Anderson 1996). Prior experience with a solution assists in problem resolution by allowing the solver to see its applicability to the problem at hand, resulting in a successful conclusion (Saugstad 1955; Staats 1957). However, solvers pay a price for experience in problem-solving when the problems to be solved are different in nature from the problem a solver has worked with on the past. The reason for this is that experience in problem solving has a tendency to produce attitudes and biases that favor the choice of problem solving strategies found successful in one instance to subsequent problems irrespective of whether the problem is similar or dissimilar to the one experienced earlier. This effect is described in the Gestalt tradition of cognitive psychology as set or "Einstellung" (Luchins 1942; Luchins and Luchins 1959) and it indicates that past experience biases the problem representation and that a resolution requires changes in representation and reduction of restraining forces (Lewin 1936).

In studies of the role of experience on problem solving Luchins (1942) found that individuals exposed to a solution to a complex problem overwhelmingly used the same (complex) solution methodology to solve simpler problems. In fact, 81% of the subject who had previously learned a complex solution to a problem utilized it to solve problems for which less complicated solutions would suffice. Experimental subjects could not

"see" the simple method until it was pointed out to them. Subjects were observed saying "how stupid I am" or "how blind I am" when they were later confronted with the more effective, but during the experiment, unnoticed solutions.

Similarly, Duncker (1945) and colleagues (Adamson 1952; Birch and Rabinowitz 1951) have shown the existence of "functional fixedness" where problem solvers have difficulty in using familiar tools in novel ways. In one of Duncker's experiments, he created five problems which could only be solved by applying a new way of using a tool. The first of the two groups of experimental subject saw the tool being used in a usual way while the second group did not. The result of the exercise was that subjects were more likely to solve the problems requiring a novel way of using the tool if they had not observed how that tool was used in the usual way while the problem solving success of subjects that had previously observed it used was hampered. In Duncker's terms the subjects were "fixated" on the tools' normal function and could not re-conceptualize it in a way that permitted them to solve the problem. Related work by Gordon (1961) also pointed to "blindness to solutions" as a main hindrance for effective problem solving and contemporary research (e.g.: Lovett and Anderson 1996) shows that initial success with problems causes solvers to cling their "history-of-success" in subsequent solving rounds with negative effects.

Outside of the laboratory, Allen and Marquis (1964), in their study of teams of government R & D contractors (Allen and Marquis 1964) found that they overwhelmingly used prior experience and knowledge in new technical problem solving. The authors defined a "set" as a bundle of prior experience with tools or approaches that are transferred across problem situations. The sets can have both positive and negative biasing effects on problem resolution. In line with the argument above, their research found, that in a few cases prior experience was helpful in creating a solution to previously experienced similar problem, however, in many cases, prior experience hindered problem resolution because it created an inappropriate bias that obfuscated attempts to reach a more superior solution. Interestingly, Allen and Marquis (1964) did find that team which

considered alternative approaches beyond their prior experience increased the probability of successful outcome from zero to 50 percent.

2.2.2: Local Search in Firms

At the firm level, re-applying experience is likely to increase problem solving effectiveness if the problems at hand look similar to prior problems that were solved successfully by the solver organization. The role of experience in increasing productivity was first observed by aeronautical engineers such as (Wright 1936) noting that the number of labor hours required to produce an airplane body is a decreasing function of the number of airplane bodies previously produced. Subsequent work on the concepts of "learning curve effects" (e.g. Alchain (1963)) and the related concept of "experience curve effects" (Boston Consulting Group, 1972) has shown this regularity of cost reduction over time (Arrow 1962); as a function of output (Rapping (1965) and Sheshinski (1967)); investment (Lieberman 1984); and learning effects variation in different contexts (Adler and Clark 1991; Argote and Epple 1990). Thus, organizations whose primary mission is to (re)produce the same type of products and or experience are likely to benefit from reapplying experience to future problems.

However, a number of researchers studying problem solving at the organizational level have argued that prior experience leads to a number of biases that block the organization from seeing potentially more effective alternative problem solving approach (March and Simon 1958; Nelson and Winter 1982). As demonstrated in the literature on evolutionary economics (Dosi 1982; Nelson and Winter 1982), organization learning (Levitt and March 1988) and technology management (Anderson and Tushman 1990) the search for solutions to novel technological problems often involves a "local search" process.

Several scholars have systematically demonstrated the presence of local search tendencies in R & D and new product development activities through empirical studies. Helfat observed very little variance, over time, in R & D spending on various

technologies in energy firms. Stuart and Podolny (1996) showed that nine out of the ten largest semiconductor firms concentrated their new patenting activity in technological niches where the firm had previously patented. Martin and Mitchell (1998) demonstrated that new product introductions from incumbent firms often incorporated designs already present in their existing products. Similarly, Sørensen and Stuart (2000) have shown that older firms in the biotechnology and semiconductors industries tend to patent in areas that are close to their expertise but are considered "obsolete" by the rest of the industry.

2.3: Innovation and External Knowledge

In this chapter I focus on innovation, a process characterized by its requirement for new knowledge, new combinations of existing knowledge, and non-standard procedures. Problem solving for innovation thus requires that solutions to novel problems be found (Clark 1985). Innovation will thus be affected negatively by the local search-based problem solving approaches and its tendency to re-apply prior successful solutions to novel problems.

External sources of knowledge have been found to be critical to the innovation process (Cohen and Levinthal 1989; Cohen and Levinthal 1990). March and Simon (1958: 188) first posited that, at the organizational level, most innovations result from "borrowing rather than invention." They contended that borrowing was either a function of imitation or by importing new personnel into the organization. Regardless of the mechanism for borrowing, they postulated that the rate and type of innovation was based on the communication structures of the organization with a special emphasis on external connections.

For an R & D laboratory, seeking knowledge from the outside has been viewed as important for getting information about user and markets needs and for the latest in technological developments. Lab personnel need to understand market requirements (Meyers and Marquis 1969) and the use conditions for current and future products (von Hippel 1978; von Hippel 1982; von Hippel 1988) in order to create competitive products. They also need to keep up to date on the most current developments in science,

technology and technical processes in order to stay innovative (Allen 1977; Allen and Cohen 1969; Utterback 1971). Thus seeking knowledge from the periphery of the focal innovating unit has been identified and empirically shown as being important to the innovation process.

2.3.1: People and Capability for Integrating External Knowledge

Studies of R&D activity have shown the emergence of both special individuals and a generalized capability for the task of integrating external knowledge into the firm.

Technological Gatekeepers

In the 1960's a range of studies on technical organizations found that there was a consistent inverse relationship between the performance of industrial and governmental scientists and technical staff and the extent to which thy used people outside of their organizations as sources of information (Allen 1966a; Allen, Gerstenfeld and Gerstberger 1968; Berul et al. 1965). Equally compelling was the data that showed the intra-organizational communication had a strong effect on performance (Allen, Gerstenfeld and Gerstberger 1968; Schilling and Bernard 1964). Allen (1966a) suggested that the explanation for this inverse relationship was due to the mismatch between the coding schemes (Katz and Kahn 1966) of the information seeking organization and the information source. The issue was not the external source of information, rather, the internal information seekers had not articulated the problem in a manner that allowed them to seek an appropriate information source.

Allen and Cohen (1969) hypothesized that the mismatch between the coding schemes could be reduced by individuals who specialized in playing an informal bridging role between the external environment and the R & D laboratory. Inspired by research on mass communications (Katz 1960; Katz and Lazarsfeld 1955; Lazarsfeld, Berelson and Gaudet 1948), they hypothesized a two-step process "through which the average engineer was connected by an intermediary to information sources outside his laboratory" (Allen and Cohen 1969: 13). They analyzed the communication pattern in two R & D laboratories and found that some individuals were more frequently chosen than others for

technical discussion. Scientific and technical staff in these laboratories reported using these sociometric "stars" as sources of information in their critical incident reporting. An analysis of the communication patterns of the stars showed them to be consulted more frequently then other internal sources and that they were well connected to both external individuals and read significantly more professional, scientific and technical journals. These stars were dubbed technological gatekeepers. Allen (1970) also replicated these findings in a large aerospace firm. Tushman (1977) showed the importance of "special boundary roles" in dynamic environment with the observation of a curvilinear relationship between the number of boundary roles and project performance for teams facing work-related uncertainty due to changing task environment or high task interdependence. Tushman also found that work characteristics were not determining the creation of boundary roles, instead, high-performing teams were evolving the appropriate number of boundary roles to fit the external information processing demands of their work.

It is important to note that R&D labs did not necessarily have a formal gatekeeper role. Rather, Allen and colleagues, discovered a statistical relationship and researched its implications in terms of internal and external communication and performance. Thus there was no apriori identification of technological gatekeepers. The crucial function that gatekeepers perform is to translate between internal organizational codes and external sources of information. Some may help insiders to better articulate and describe the research problem and make a match between external information, while others may simply absorb external information and when necessary translate that into a form that fits best into the language and culture of the organization (Tushman and Katz 1980). Also note that the local search problem articulated above still exists. In this case the gatekeeper is the primary individual that is conducting search to find and absorb relevant information for the organization. While we can expect the search by the gatekeepers to be slightly broader, gatekeepers will still be limited by their own prior experience and success and failures within the organization.

Absorptive Capacity

Cohen and Levinthal (1989; 1990) argued that the R&D function in a firm serves to both create new technologically based products and to leverage external information for the firm's future competitive advantage. They defined "absorptive capacity" as a firm-based capability to understand and value external information and to appropriate it for commercial ends. Cohen and Levinthal posited that absorptive capacity was present at both the individual and firm level and that prior experience and diversity of expertise were critical components for effective external knowledge assimilation and exploitation (Zahra and George 2002). They used findings from cognitive and behavioral sciences to argue that absorptive capacity, at the individual level, was a type of learning capability that relied on past experience with the intensity of previous effort being an important ingredient (pg 131). They also claimed an equivalence between problem solving ability and learning capability, arguing that both developed in a similar fashion and relied on prior knowledge for effectiveness, even though problem solving aims to create new knowledge and learning capability seeks to assimilate existing knowledge (pg 130).

Cohen and Levinthal also argue that diversity of knowledge structures at the individual and organization level is critical for effective absorptive capacity. Since absorptive capacity is based on deep prior knowledge having prior access to diverse kinds of knowledge is essential for the ability to both assimilate and to exploit differing external knowledge. For the individual, they cite Simon's (1985) contention that diverse knowledge structures coexisting in the same mind can lead to learning and problem solving that yields innovation. And for the firm, they use Utterback's (1971) findings that diversity in the workplace can stimulate the generation of new ideas. Overall they posit that "interactions across individuals who each possess diverse and different knowledge structures will augment the organization's capacity for making novel linkages and associations" (pg 133). Interestingly they also noted that gatekeepers may not be suitable in conditions of technological uncertainty since centralized interfaces to the environment are not going to be able to access and assimilate vast amounts of rapidly changing technological information. Their solution was to recommend that firms expose a broad and large range of "receptors," i.e. many individuals, to the environment.

The strong link between prior knowledge accumulation and absorptive capacity has been replicated and extend via numerous empirical and theoretical studies (for a review see Zahra and George (2002)). However this implies that the local search effects of absorptive capacity will be even stronger. While diversity of searchers focused on external information and collaboration may have a positive impact on innovation performance (Cockburn and Henderson 1998) it is also important to note that there is strong tendency towards homophily in groups (McPherson 1983; McPherson 1981; McPherson and Smith-Lovin 1987; Popielarz and McPherson 1995) with empirical evidence showing that hiring practices inside of firms usually ends up either reducing diversity (Sosa and Fernandez, 2005) or sustaining current structures (Rubineau and Fernandez, 2005). So a focus on absorptive capacity as a means to bring external information may end up bringing the same type of information into the organization. Empirical evidence of the failure of large firms to innovate in new technological domains seems to support this contention (Christensen, Suarez and Utterback 1998; Utterback and Suarez 1993).

2.3.2: Alternatives to Local Search: Alliances and Mobility as Means of Accessing External Knowledge

Alliances

Spanning the boundaries of the focal R & D organization has been suggested as an alternative to overcoming the negative biases of local search (Rosenkopf and Nerkar 2001). Typically external boundary spanning occurs via formal alliance agreements. Two dimensions of boundary spanning activity have been posited: 1) technological and 2)organizational. Technological boundary spanning can occur in domains that are similar or distant to current capability. Stuart and Podolny (1996) showed that in the period between 1987-1992 changes in semiconductor firms' technological position were highly and positively correlated with the number of technology-exchange and technology-development alliance agreements they had consummated. Only one firm, Mitsubishi, was able to change its technological profile in the semiconductor industry and avoided the pitfalls of local search by having the greatest number of strategic technology alliances with other firms. Similarly, Nagarajan and Mitchell (1998) showed that in the lithotripsy

industry access to radical technologies occurred through equity-based relationships between firms.

Organizational boundary spanning can be intra-organizational, i.e. going to a different unit for access to technology (for example Hansen's (1999) analysis of technology search and transfer in a high-tech multi-national firm) or inter-organizational (e.g.: Nonaka and Takeucki's (1995) example of Matsushita learning the art of kneading bread from a chef in order to build its own electronic home bakery). Rosenkopf and Nerkar's (2001) patent data-based analysis of boundary spanning activity in the optical disc industry showed that inter-organizational boundary spanning combined with distant technological boundary spanning yielded the greatest impact, i.e. future citations from other firms within the same patent domain.

Boundary spanning via alliances can be used to either converge or diverge knowledge bases within alliance partners. Mowery et al (1996) have shown that alliances can be used to transfer capabilities between firms or as a means for complementary specialization. However, when Rosenkopf and Almeida (2003) excluded arms-length contracts, the significance of alliance as a predictor of knowledge flows disappeared (pg 763). Thus the general ability of alliances as a means of overcoming local search may still be limited to search within a rather well-known environment.

Mobility

Mobility of technical and managerial staff has also been proposed as way for overcoming local search constraints (Rosenkopf and Almeida 2003). The basic underlying mechanism is that people are a key source of firm-to-firm knowledge spillovers and as engineers switch jobs and firms they also transfer with them unique, to the hiring firm, experiences and knowledge (Saxenian 1994). Thus, Almeida and Kogut (1999) have shown that in the semiconductor industry hiring of a new engineer resulted in the hiring firm exhibiting higher rates of citing prior patents of the new employee than would be expected given its current technological niche. Similarly Rosenkopf and Almeida (2003), study of the semiconductor industry, demonstrated that mobility of

patent authors resulted in inter-firm knowledge transfer. However, this needs to be tempered with the findings that new technical employees in R & D laboratories have the most difficulty and take a long time in assimilating into pre-existing communication environment of development organizations (Lee and Allen 1982).

One potential critique of the "alternatives to local search" literature is that from a theoretical point of view boundary spanning and mobility do not overcome local search bias at all, rather they simply shift the axis of local search from technological problem solving to either one of finding the right alliance partners or the best technical staff. Firms employing these solutions are still going to face local search constraints in the areas of identifying relevant capabilities and skills in potential firms in the environment or individuals in the labor force. The characteristics of the problem solving process – search and learning are now located in strategic and managerial domains instead of the technological domain.

2.4: Broadcast Search

When innovation requires access to knowledge that is widely distributed then a local search-based problem solving effort may not yield the most optimal or efficient solution. There is an inherent tension between the extensive use of local search in problem solving and the observation that knowledge is widely distributed in society (Hayek 1945). Although the problem of making optimal use of distributed knowledge is one that any complex social system confronts, the problem of distributed knowledge is especially pressing for firms involved in innovative efforts as its relies on its ability to rapidly access varying knowledge bases often collocated with highly specialized individuals.

2.4.1: Distributed and decentralized problem solving and the importance of the periphery

Research on the sources of technological innovation has repeatedly highlighted that novel innovations arise when problem solving activity is decentralized (von Hippel 1988; von Hippel 2005). Users have been shown to innovate in a variety of consumer, industrial and scientific settings (von Hippel 2005), often preceding and initiating firm-

based efforts (von Hippel 1978; von Hippel 1982; von Hippel 1988; von Hippel 1989). Here users are on the periphery and the firms who commercialize and sell products which have innovations are the core. Thus in the field of scientific instruments, Riggs and von Hippel (1994) found that 44% (n=64) of the innovations emerged from users dispersed in industry, universities and government laboratories while the remaining 56% of the innovation emerged from a handful of manufacturers. They further found that the vast majority of functionally novel innovations, i.e. enabling new technical capability in the equipment, were developed by dispersed users and "dimension of merit" improvements, i.e. convenience or reliability, were developed by manufacturers. More recently, DeMonaco, Ali & von Hippel (2005) have shown that in pharmaceuticals industry, 76% (n=29) of the new drugs introduced in 1998 had significant "off-label", i.e. novel uses not in the original drug approval process, applications. They found that 59% (85/144) of the "off-label" drug therapy innovations were discovered by distributed and peripheral practicing clinicians via field discovery as compared to the scientists working inside of the pharmaceutical companies.

One stream of research in the sociology of science has argued that the flow of ideas and innovation in scientific communities is centripetal instead of centrifugal (Chubin 1976), that is, the margins of the scientific community are the drivers of change and progress. Thus Crane (1969: 349) in her study of the "invisible college" in the natural sciences speculated that "outsiders" were a likely source of new ideas and innovation: "Most problem areas are open to influence from other fields. The desire for originality motivates scientists to maintain contacts with scientists and scientific work in areas different from their own in order to enhance their ability to develop new ideas in their own areas."

A review of six scientific disciplines, (radio astronomy, bacteriology, psychology, phage group, physical chemistry, x-ray protein crystallography) by Edge and Mulkay (1974) (cited by Chubin 1976) showed that innovations from the margins and the mobility of scientists across fields were the only consistent factors in scientific innovation and specialty development across these fields. Edge and Mulkay did express concern that

very little was known about the social process underpinning their findings: "If we are correct in suspecting that many major scientific innovations come from the outside, or from the margins of, established research communities (either from applied research contexts, or by migration between research networks), then it is surprising that so little is known about this process" (Edge and Mulkay (1974) cited in Chubin (1976: 457)).

Within sociology, Weiman's (1982) study of the flow of information and influence in the personal network of an Israeli kibbutz community also shows "the importance of marginality" or peripheral participation. Weiman gathered sociometric data from 270 members of the kibbutz regarding conversational ties with other members of the community yielding 2511 conversation ties. Weiman then used matrix algebra to determine cliques in the community and then derived a network position of each individual based on the number of times a person was chosen as a conversational tie by someone else. "Centrals" and "Marginals" were then determined by using the upper and lower quartiles of the choice distribution in a clique. As expected centrals, dominated in all types of communication patterns. In addition centrals, were more efficient in the flow of information. Information originating from centrals flowed more faster, was deemed more accurate and more credible than the information activated by the marginals. However, marginals were key for inter-group or inter-clique communication. Marginals were both receivers and transmitters of information amongst the 16 distinct groups in the kibbutz. Weiman showed that marginals were the importers of new information across groups and that centrals then served as the transmitters of that information within groups. Implying that centrals rely on marginals for imported information while the marginals required the enlistment of centrals for spreading the information in the group.

There are two theoretical perspectives underpinning the findings related to the importance of the periphery. Granovetter's (1973) seminal article on the strength of weak ties posits that weak ties amongst individuals allow for the transfer of non-redundant and novel information amongst colleagues as opposed to strong ties amongst friends. Strong ties imply that the information flow amongst strongly connected individuals will be homogenous and already known, while weak ties may enable the

transfer of new and heterogeneous information. Thus those on the periphery of a community are more likely to be weakly tied to the core, while they may serve as "bridges" between other communities and thus transfer novel information amongst them. This theoretical perspective is the basis for both Weiman's empirical findings about the importance of marginality and Chubin's assertion regarding the centripetal flow of novel information in science communities. Although not mentioned by Granovetter, Hayek's (1945) central insight about the unequal and distributed nature of knowledge in society explains why non-redundant information may exist in the first place. If knowledge is both spatially and intellectually distributed – then gaining access to this knowledge via weak ties may be one mechanism by which peripheral members provide advantages to communities.

The other theoretical perspective on the value of the periphery arises from von Hippel's findings about the critical role of users in the innovation process. Here the theoretical perspective is the relative stickiness of information. Von Hippel argues that the locus of innovation shifts to where the information is the most stickiest (von Hippel 1994a; von Hippel 1994b; von Hippel 1999). Thus users innovate in areas where they have needs not met by manufacturers, typically in using technologies in novel ways, while manufacturers innovate in areas where they have pre-existing expertise, typically manufacturing the technology or improving it on the dimensions of merit instead of novelty. It is not just a matter of the presence of non-redundant information. Rather users or peripheral members in problem solving communities experience novel issues not foreseen by manufacturers or core members and in many cases the transfer of this use experience is very difficult and expensive, if not impossible. A strong tie between a core developer and a peripheral user does not mean the core will now have the information needed to innovate. Rather the use environment will dictate that such innovation has to be primarily driven by the periphery. Thus the periphery has to first innovate and then transfer the newly created knowledge to the core, regardless of the strength of ties.

2.4.2: Defining Broadcast Search

A radical departure from local search-based problem solving would be for the problem holder to not engage in any problem solving activity at all and instead find ways to interest a heterogeneous set of external actors to attempt solving the problem in their own particular domains. One possible way to access the widely distributed knowledge base and potential solvers is to broadcast problems as widely as possible. I call this shift moving from local search to broadcast search. I define broadcast search as the shifting of problem solving activity away from the center of the R&D organization to the outside periphery.

In broadcast search, problem holders do not seek external knowledge and information in aid of their own local solution creation effort. Instead they distribute their own problem to many heterogeneous domains and allow solvers in those domains to create solutions. Thus the problem holder is transformed into a solution seeker. Instead of doing search through a problem space, the solution seeker defines the problem in a way that can be understood by outsiders and finds a way to access many diverse potential solvers. Problem solving and solution generation shifts to the periphery and potential solvers self-select themselves to participate. Broadcast search depends on the law of large numbers and diversity in the potential solver base for solution creation. It is not enough that many solvers attempt to create a solution, rather, there has to be diversity amongst the solvers so that many different approaches may be attempted and many different solutions created.

The first element in broadcast search is problem definition. Problems need to be defined in such a way to allow for the greatest possible openness of the solution space. This requires that problem definition activity be separated from solution identification effort because often times problem definition is so closely intertwined with solutions that there is a suboptimal lock-in between them (Simon 1973). Second, instead of local search, the key task of is problem broadcasting. To increase the probability of a successful response, problems are broadcast to a large heterogeneous set of solvers not necessarily associated with the problem holders or even in the same scientific and

technical domains. This way problems can draw on multiple and distributed knowledge domains for a solution. Third, to attract solutions to a problem an incentive structure needs to be in place. Incentives can appeal to the solver's extrinsic and/or intrinsic motivations i.e. monetary rewards (e.g.: prizes (Horrobin 1986)) or personal sense of creativity (Csikszentmihalyi 1975; Deci and Ryan 1985). Finally, the solution that meets the problem's resolution criteria needs to be selected.

Broadcast search is different from the traditional problem solving in three distinct ways: 1) It initiates numerous local searches in different domains by leveraging the distributed nature of knowledge. Thus instead of one local search by the problem holder, there are multiple local searches by many solvers. 2) The problem holder's effort shifts from creating solutions to testing completed submission solutions. 3) Because it rewards acceptable solutions only it externalizes the cost and failure-risk of problem resolution.

2.4.3: Overcoming Newton's Folly By Broadcast Search

"And I have told you oftener then once that it [the longitude] is not be found by Clock-work alone. .... Nothing but Astronomy is sufficient for this purpose. But if you are unwilling to meddle with Astronomy (the only right method and the method pointed at by the Act of Parliament) I am unwilling to meddle with any other methods then the right one." – Sir Isaac Newton, 1725 (Andrewes 1996)

Sir Isaac Newton, Western Civilization's most preeminent Natural Philosopher and scientist, boldly asserted that the solution to finding the exact longitude at sea was only to be obtained by the methods of astronomy and not by clockwork. Finding a practical method to obtain the longitude at sea was considered one of the greatest scientific challenges between the 16th and 18th centuries. Three theoretical solutions were proposed: 1) lunar distance method by Werner (1516), 2) clock-based solution by Frisius (1530) and 3) Jupiter moons method by Galileo (1616). Among the scientists who attempted, though un-successfully, to create practical solutions included Cassini, Huyguens, and Haley (Sobel 1996). So severe was the need to obtain the longitude at sea, that in 1714, the British Parliament passed an act establishing a prize of up to £20,000 for anyone who could come up with any reliable method to solve the problem. The parliamentary act defined the range of acceptable solutions (1, 2/3 and ½ degree

accuracies) and established the Board of Longitude to evaluate and select the prize winners (Andrewes 1996).

Newton, as the principal scientific advisor to the Longitude Board repeatedly rejected all other approaches: *"The Longitude will scarce be found at sea without pursuing those methods by which it may be found at land. And those methods are hitherto only two: one by the motion of the Moon, the other by that of the innermost Satellit of Jupiter "*(Andrewes 1996: 190). ***Newton's folly*** was his insistence on an astronomical solution to the longitude problem, based on his own knowledge and experience in astronomy and his rejections of alternative proposals and solutions. His public rejections of any proposals that were based on timekeepers convinced most of the scientific community at the time, to propose and develop solutions that were based on astronomy and ignore the domain of clockwork (Andrewes 1996).

The parliamentary act spurred a tremendous outflow of proposals as it had a handsome reward and scientific prestige. However, contrary to Newton, the ultimate practical solution was based on clockwork and not astronomy and it was developed by an unknown carpenter and clockmaker, John Harrison of Yorkshire, England. Harrison's solution was innovative by two accounts it eschewed astronomy for clockwork and his design was quite different from existing clocks indicating novel understanding of materials science and mechanics. Harrison was also on the distant periphery of the scientific and technical community of the time. Newton's folly was so deeply engrained in the scientific community and the Longitude Board that it took 40 years of effort and a special favor by King George III for Harrison to be declared the winner of the Longitude prize. The longitude episode demonstrates that solutions to difficult scientific problems can emerge from unexpected areas if they are broadcasted to a diverse solver population.

2.4.4: Contemporary Research on Broadcast Search

To my knowledge there are three studies that have examined variants of broadcast search as it relates to technical support for computer and software related problems. Two studies in the late 1980's at Tandem Corporation examined broadcast and public and

private replies to requests for help from field engineers (Constant, Sproull and Kiesler 1996; Finholt, Sproull and Kiesler 2002). Constant et al studied know-how questions – i.e. "How does something work?" about a company's internal products within Tandem's internal email list. They found that on average each question generated replies from eight people but only 8 percent of the solution providers knew the person with the problem. 50% of those posting questions had their problems solved. In addition the mean diversity (geographic location) and resources (being at headquarters or self-admitted expert in the area) of information providers to any one question significantly predicted the probability of a question being answered.

Finholt et al's studied the use of archives of previously broadcast peer-to-peer help discussions and expert-user discussions. They found that the peer-to-peer archives had more stores of knowledge with the mean number of replies and participants to each question were significantly higher than the expert-user discussions. In addition they found that field engineers significantly preferred to use the peer-to-peer archive over the expert-user archive and that this preference increased with distance from headquarters.

Lakhani and von Hippel (2003) studied the provision of technical help in an online voluntary community setting outside the boundaries of any one firm. The main findings indicated that the system was effective in solving problems for users with 63% of participants noting problem resolution through their use of the online system. They also found that the majority (67%) of solution providers were not doing any de-novo problem solving – instead they were simply transferring solution information that they already had. The benefit to cost ratio in terms of time saved to time invested in the problem was approximately nine.

2.5: Setting and Data Sources

I explore the functioning of broadcast search examining its application to a unique data set of 166 discrete scientific problems originating from the research and development laboratories of 26 firms from 10 countries from June 2001 to January 2005[4].

[4] Information on all the problems analyzed are in the Appendix.

The firms were in such diverse industries like agrochemicals, aerospace, biotechnology, chemicals, consumer products and pharmaceuticals. Most firms had previously tried to solve the problem within their own laboratories with some exerting several years of prior effort. The data for the analysis was obtained in cooperation from InnoCentive.com (an independent venture of the Eli Lilly & Company pharmaceutical firm), whose business model is centered around broadcasting science problems. InnoCentive.com (IC) acts like a knowledge broker between "seeker" firms and over 80,000 independent and globally dispersed "solvers" from 160 countries (Sawhney, Prandelli and Verona 2003). IC's business model is contingent upon attracting seeker firms to post internal research problems on its website and encouraging solvers to examine and submit solutions to those problems for a potential monetary award.

Seeker firms work in consultation with IC's scientific operations staff to articulate their internal problems in a form that can be understood by an external scientific audience. Solution requirements for the problems are either "reduction to practice" (RTP) submissions, i.e. either the actual chemical or biological agent or detailed experimental results needs to be provided (original research data has to be provided) or "paper", i.e. a theoretical submission with a validated research proposal needs to be provided. Problems are posted on IC's website along with a pre-set monetary award for the "best" solution and a deadline date for submissions. IC then broadcasts the problem to its entire solver base via email and invites them to participate in solving the problem. IC solvers do not work collectively to solve the problem. Solvers also do not know who else is working on the problem and how many solutions have been submitted. IC screens all submitted solutions to ensure that the problem requirements have been met and then forwards them to the seekers. Scientists from within the originating R&D laboratory assess the submissions and then inform IC if they have found one that meets their criteria. The decision to award the prize money to the best solution rests entirely in the hands of the seeker firm. The seeker firm can chose to not award any prizes or award multiple prizes.

Seekers and solvers remain anonymous to each other throughout the problem solving process. Care is taken to protect the intellectual property (IP) rights of seekers and solvers. When a problem is broadcasted, solvers initially see an abstract of the problem definition. If they are interested in seeing full details and requirements about the problem they have to first agree to a solver agreement which outlines the reward and review period for solutions, confidentiality, and intellectual property transfer clauses for accepted solutions. Solvers that submit solutions give a temporary license to the seeker firm to evaluate their solution. If the solution is deemed acceptable by the seeker firm, the solver then receives the pre-announced award prize and transfers all IP rights to the seeker company. Before the transfer takes place IC contacts the solver's employer to ensure that they release any and all IP claims on the solution[5]. If the solution is not accepted the seeker firm relinquishes any rights to use the information provided in the submission in any future work. This is enforced by contracts between IC and the seeker firm which allow IC rights to initiate audits on the output of the seeker firm's research laboratories.

I conducted two types of analyses to understand the efficacy of broadcast search and its functioning. First, I analyzed the problem solving process used to create a solution and the characteristics and motivations of the solvers. I wanted to understand how solvers came up with a solution and the determinants of a solver being able to successfully create a "winning" solution. Information on solvers and the problem solving process was obtained via an online, web-based survey of individuals who had submitted solutions to problems[6]. The survey was administered in cooperation with IC[7] and took about 20 minutes to complete.

Each solver received a customized email from IC's Chief Scientific Officer requesting them to participate in the survey. The email asked the solvers to respond to

[5] There have been only two cases where the employer of the solver refused to release the IP rights to a solution. I did not consider those two cases in the analyses.
[6] Survey questions are in the Appendix.
[7] I first created a pilot survey to ensure that the questions were understandable by members of the survey population. I administered the pilot survey to two current IC solvers and to three other individuals who had similar backgrounds (PhD in a science discipline) as the IC solver base. The feedback from the pilot survey helped me to fine tune the questions and remove ambiguous language.

the survey by reminding them of a specific problem for which they had attempted a solution along with the date of their submission to IC. Solvers who had created submissions to multiple problems were asked about their most recent submission. Those who had been successful in at least one attempt were asked to respond to the survey with regards to their most recent winning submission. Most solvers also had the ability to review the detailed problem statement and their submission on their personal account space on IC's website[8]. Survey respondents were offered the opportunity to participate in a random drawing for gift certificates upon completion of the survey. The survey was sent to 993 individuals and yielded a relatively high response rate of 35.9% (n = 357) (Sheehan 2001). In all 68% of the winning solvers and 34% of the non-winning solvers responded to the survey.

Second, I analyzed what determined if a problem was successfully solved by examining the characteristics of the problems and the solver base attracted to solve it. IC provided me with all salient information about each of the problems including solution requirements (RTP vs. paper), scientific discipline, seeker firm (anonymized), award value, days a problem was open for submission and size of IC's solver network over time. In addition, IC also provided anonymized information about scientific interests of the solvers who were submitting solutions. At registration time with IC, solvers are asked about their scientific interests from a list of 56 options spanning Chemistry and Applied Sciences and the Life Sciences. The scientific interest information helped me to understand the types of solvers that were being attracted to the various problems and to analyze the intellectual heterogeneity of the solver base.

2.6: Findings

2.6.1: Broadcast Search Performance and Solver Base Profile

Table 2.1 shows the overall performance of broadcast search-based scientific problem solving. Of the 166 problems posted, 49 were deemed solved by the seeker firms yielding a solve rate of 29.5%. A majority 58%, of the problems required a solution that entailed "reduction to practice" (RTP) submission with the remaining asking for

[8] Solvers who had created submissions in 2001 did not have access to their submission data.

theoretical submissions. All problems offered a substantial financial award (Mean: $29,689, Median: $25,000, Range: $2000-105,000 US). Problem solutions had to be delivered within a limited number of days (Mean: 166 days, Median: 108 days, Range: 14 to 554 days). Overall 29.5% of problems (49 out of 166) were solved using this approach.

Table 2.1: Overall Performance Of Broadcast Search By Scientific Disciplines

Discipline of Problems Posted	Number of problems	Paper|RTP (%)	Average Award Value (USD$)	Average Number of People Expressing Interest	Average Number of Submissions	Number of Problems Resolved	Solving Rate (%)
Chemistry and Applied Sciences							
Synthesis	71	30|70	37408	223	9	22	31.0
Formulation	27	66|44	24666	220	10	8	29.6
Analytical	16	50|50	25375	314	13	1	6.3
Polymer	13	54|46	26884	254	8	1	7.7
Materials Science	4	50|50	25000	335	11	3	75.0
Other	5	60|40	22676	464	35	4	80.0
Life Sciences							
Biochemistry	11	27|73	33181	269	5.7	0	0.0
Molecular Biology	7	43|57	15000	116	3	2	28.6
Biology	7	71|29	14571	236	9	5	71.4
Toxicology	3	67|33	12500	80	1	2	66.7
Medicinal Chemistry	2	50|50	14000	228	4	1	50.0
Total	166	42|58	29689	240	10	49	29.5

Winning solutions arrived on average within 84 (sd: 65.5 range: 2-262) days of the problem posting.

On average, 240.7 (s.d.: 195.0, range: 19-1058) individuals examined the detailed problem statement and 9.9 (s.d.: 14.2, range: 0-103) of them submitted solutions. Problems that were solved received more than twice (mean: 16.8, s.d.: 21.5) the number of submitted solutions as non-solved problems (mean: 7.0, s.d.: 8.1) (t=-4.2, p=0.000). 71% of the solved problems had a single "best" solution with one solver getting the award and the remaining 29% had multiple "best" solutions (range: 2-5) with multiple prizes awarded per problem. In all there were 75 winning submissions covering 49 solved problems. In a few cases, independent solvers provided partial non-overlapping solutions to the problem in such a way that the seeker firm could accept all solutions and combine the result into a full solution.

The data shows that the successful problem solvers are widely distributed with 87.5% of winning solvers have one successful submission and 8% have two successful submissions[9]. Two contract research labs were successful three and four times, however, anecdotal evidence suggests that the solvers were different individuals in each case[10]. In all 59 solvers received prize awards. This wide distribution of solvers and the lack of a few "genius" solvers is in sharp contrast to other distributed innovation systems. For example studies of open source software development communities have shown the presence of a power-law distribution of participation where very few people contribute disproportionately to the problem solving effort (Lakhani and von Hippel 2003; Mockus, Fielding and Herbsleb 2002).

In Table 2.2, I provide descriptive statistics on the respondents in the sample. Most of the solvers who responded to the survey were male (89.1%), however there were twice as many female winning solvers (20.0%) as non-winning solvers (10.9%). Solvers

[9] I cannot unequivocally state that there is no power law distribution since I do not have enough observations. It may be possible that given a 1000 attempts at broadcast search a power law distribution may emerge.
[10] Based on discussion with InnoCentive scientific operations staff.

were highly qualified with a majority of solvers reporting having PhD degrees and 20% reporting post-doctoral or habilitation experience. On average, solvers had completed their formal education 16.3 years (s.d.:12.9) prior to responding to the survey with no significant difference between PhD and non-PhD holders. At registration time with IC, solvers indicated their scientific interests from up to 56 options. In the sample solvers reported an average of 11.1 (sd: 11.7, range: 1 - 56) areas of scientific interests with no significant difference between winning solvers and non-winning solvers. The respondents took, on average, 39.9 hours of their own time developing a solution with winning solvers reporting more than twice as many hours (74.1) as non-winning solvers (35.7).

2.6.2: Problem Solving Process Used By Solvers

To gain insight into the nature of the problem solving process, I asked several questions regarding solvers' prior knowledge and experience with similar problems and solutions, source of solution information, match between problem domain and their own expertise and the number of other people involved in the problem solving effort.

Table 2.3 shows that a majority of the solvers had some experience with similar problems before. I asked the respondents: "w*hen you first saw the particular InnoCentive Challenge, what was your experience with similar problems?"* This item was rated on a 7-point scale ranging form 1 – This problem was completely new to me, 4 – I was somewhat familiar to 7 – I had seen the EXACT problem before. The average score was 3.9 (s.d.: 1.6) with approximately 40% of respondents reporting high degrees of experience with similar problems. There was no significant difference in response patterns between winning and non-winning solvers.

Table 2.2: Profile Of Solver Base

	Total Sample	Winners	Non-Winners	t-statistic [p-value]	N
% Female	11.9	20	10.9	-1.66 [0.09]	351
% PhD	65.8	71.7	65	-0.83 [0.40]	351
Mean years since final degree (s.d.)	16.3 (12.9)	16.1(12.3)	16.4(13.0)	0.15 [0.89]	325
Mean number of scientific interests (s.d.)	11.1 (11.6)	10.1 (9.1)	11.3 (11.9)	0.56 [0.56]	327
Mean time spent developing solution in hours (s.d.)	39.9(86.7)	74.1(144.6)	35.7(75.4)	-2.5 [0.009]	342

Table 2.3: Respondents' Familiarity With The Problem

"When you first saw the particular InnoCentive Challenge, what was your experience with similar problems?"	Total Sample (%)	Winners (%)	Non-Winners (%)
1 - This problem was completely new to me	11.6	15	11.1
2	7.9	2.5	8.6
3	11.0	22.5	9.5
4 - I was somewhat familiar	29.0	27.5	29.2
5	25.1	22.5	25.4
6	10.7	7.5	11.1
7 - I had seen the EXACT same problem before	4.8	2.5	5.1
N	355	40	315
Mean (s.d.) rating	3.99 (1.61)	3.73 (1.55)	4.03 (1.61)

I also asked the respondents to assess the distance between the problem and their own field of expertise. Specifically I asked them to rate if the particular challenge was: *"1 – inside your field of expertise, 4 – at the boundary of your field of expertise, 7 – outside your field of expertise.* Table 2.4 shows the response pattern comparisons between winners and non-winners. In both classes a majority of respondents indicated that the problem was either inside or at the boundary of their field of expertise. However, while the mean score for this question was 2.86, the winners' mean rating was 3.35 and non-winners was 2.79, was marginally significantly different ($t = -1.87$, $p=0.06$). Indicating that winners assessed the problems to be more on the boundary our outside of their expertise as compared to non-winners.

Majority of the solvers also reported that upon seeing the problem statement they had already access to all or some of the solution information required to solve the problem. Table 2.5 shows the response pattern to the question about their information situation at time of seeing problem. Only 14.7% of solvers reported that they had no information on the solution upon seeing the problem. Interestingly there was no significant difference in response patterns between winning and non-winning solvers. Lakhani and von Hippel (2003) have reported similarly high pre-existing solution information by solvers for problem broadcasting in an online computer help forum.

Table 2.4: Respondent Mapping of Problem And Their Own Field Of Expertise

"Please tell us if this Challenge was:"	Total Sample (%)	Winners (%)	Non-Winners (%)
1 - Inside your field of expertise	29.9	23.1	30.8
2	19.8	10.3	21.1
3	17.2	23.1	16.4
4 – At the boundary of your field of expertise	15.1	15.4	15.1
5	8.6	15.4	7.7
6	4.1	5.1	4.0
7 - Outside your field of expertise	5.3	7.7	5.0
N	338	39	299
Mean (s.d.) rating*	2.86(1.76)	3.35(1.86)	2.79(1.74)

* $t = -1.87$, $p = 0.061$

Table 2.5: Respondents' Solution Information Knowledge Upon Seeing The problem

"What was your situation when you encountered this Challenge?"	Total Sample (%)	Winners (%)	Non-Winners (%)
1 - I knew the solution right away	20.74	22.5	20.51
2 - I knew where to find the solution	24.72	15	25.96
3 - I had some information but not the exact solution	39.77	35	40.38
4 - I had no information on the solution	14.77	27.5	13.14
N	352	40	312
Mean (s.d.) rating	2.48 (0.98)	2.68 (1.12)	2.46 (0.96)

To investigate the origin of solutions being provided, I asked solvers to what degree their submissions built on pre-existing solutions: *"Sometimes solutions build on previous work. Was your submission to this Challenge based on: 1) A solution you had already developed in your own work with no modifications, minor modifications, major modifications or this was not based on any of my previous work. 2) An existing solution you knew about that could solve the Challenge with no modifications, minor modifications major modifications or this was not based on anyone else's work."* Table 2.6A shows that 55% of winning solvers reported that their submissions relied on previously developed solutions they had developed in their own work with 32.5% doing major modifications to their previous work in the submission generation process. Similarly Table 2.6B shows that 60% of winning solvers also relied on solutions developed by others in their submissions with 47.5% reporting major modifications to the work of others. Table 2.6C shows the overlap in response patterns to both of these questions. Overall I find that 27.5% of the winning solvers reported doing de-novo problem solving by not relying on previous solutions developed by themselves and others. The remaining 72.5% of winning solvers stated that their submissions were partially or fully based on previously developed solutions from their own and/or someone else's work. Table 2.6C also shows that when the level of modifications to previous solutions is considered, regardless of the source, 55% of winning solvers were doing major modifications to previously developed solutions in their submissions. This indicates that broadcast search leverages pre-existing knowledge and the creative (re)combination and transformation of knowledge in the solution generation process. A similar pattern[11] of previous solution usage was also reported by non-winning solvers, thus indicating, that in general, participants in broadcast search rely on previously developed solutions to a large extent in their problem solving efforts.

[11] I tested the response patterns between winners and non-winners using a hierarchical log linear model. The results showed that there was no significant difference in response patterns. Likelihood ratio chi square = 20.678, df = 15, p =0.147; Pearson chi square = 15.105, df = 15, p = 0.444.

Table 2.6A and 2.6B: Source Of Solution Information Used by Winning Solvers in Their Submission
(N = 40)

Table 2.6A	
Information from a solution previously developed by solver with:	Percent
No Modification	5.0
Minor Modification	17.5
Major Modification	32.5
Did Not Use Previously Developed Solution	45.0
Total	100.0

Table 2.6B	
Information from a solution previously developed by someone else with:	Percent
No Modification	0.0
Minor Modification	12.5
Major Modification	47.5
Did Not Use Previously Developed Solution	40.0
Total	100.0

Table 2.6C: Source And Amount of Modification Of Prior Solutions In Creating Present Solution By Winning Solvers (% of winning solvers)

		Solution Previously Developed by Solver				
		No Modifications	Minor Modifications	Major Modifications	Did Not Use Previous	Total
Solution from Somewhere Else	Minor Modifications	0	10	0	2.5	13
	Major Modifications	2.5	5	25	15	48
	Did Not Use Previous	2.5	2.5	7.5	27.5	40
	Total	5	17.5	32.5	45	100

Pearson chi2(6) = 21.534

Pr = 0.001

n = 40 solvers

Only 10.6% of the respondents reported working in teams to solve the problem with 7.5% of winners (n=3) and 11.4% of non-winners (n=36) indicating a team effort. Average team size was 2.8 members (s.d.: 1.6) with no significant difference in team size for winning versus non-winning solvers. A vast majority of solvers (79.6%) also reported that they did not consult others (excluding team members, if any) in the development of their solution with 83.3% of winners and 73.8% of non-winners reporting no consultation with others.

2.6.3: Determinants of a Solver Creating a Winning Solution

I studied the probability of a solver having a winning solution as a function of their motivations to participate and their expertise with the particular problems. Questions regarding motivations to participate were derived from extensive interviewing with IC scientific operations staff and an examination of existing economics (Frey 1997) and psychology (Deci and Ryan 1985) literature. The interviews and literature review suggested that even though winning the award money was the most obvious reason to participate, other extrinsic motivations like career and reputation concerns, and peer and work pressure to submit a solution should not be ignored. Also solvers may also have participated for the challenge and enjoyment of scientific problem solving, thus intrinsic motivations needed to be considered as well. Studies have also shown that being the first to solve a scientific challenge and beat others is a strong motivational driver for scientists (Stephan and Levin 1992). It could also be that solvers were motivated to participate because they either had free time/capacity or were simply bored in their current jobs.

I asked the respondents to rate 16 items on the various motivations for creating a solution and found that 10 of the motivation items loaded onto two separable factors that could be labeled *intrinsic motivations* and *social/career* motivations Table 2.7 shows the result of the factor analysis on the responses. I then derived values for these factors by standardizing[12] the item scores and taking the average score for the corresponding matching factor items. The remaining items weakly loaded or did not load into additional factors. For the regression analysis I retained the *"to win the award money"*, *"having*

[12] Mean = 0 and Standard Deviation = 1 for all items.

Table 2.7: Motivations To Participate in Broadcast Search Problem Solving, Scores and Factor Loadings

Question: There are many reasons for participating in an InnoCentive Challenge. Tell us how true the following statements are for you. Please answer all items. I submitted a solution: (1-Not true at all, 4-Somewhat true, 7-Very true)

	Non-winning Solvers		**Winning Solvers**			**Factor 1 - Loadings**	**Factor 2 - Loadings**
	Mean	S.D	Mean	S.D	Significance	Intrinsic Motivation	Social and Work-Related Motivation
To learn about these types of Challenges	4.31	2.21	3.85	2.08	-	0.62	
Because I enjoy solving these types of Challenges	5.84	1.49	6.45	1.01	*	0.68	
For the intellectual challenge of solving this Challenge	5.09	1.99	6.03	1.62	**	0.71	
To enhance my skills	4.78	2.06	5.20	1.98	-	0.75	
To gain scientific recognition	3.41	2.28	3.21	2.12	-		0.5
Because someone suggested I participate in solving this Challenge	1.50	1.21	1.20	0.79	-		0.5
To enhance my career prospects	3.36	2.32	3.03	2.16	-		0.52
Because others I know have participated before	1.61	1.39	1.43	1.26	-		0.55
Because my boss asked me to work on it	1.13	0.60	1.05	0.22	-		0.57
To impress my colleagues	2.18	1.81	2.10	1.89	-		0.62
Because I had free time available	3.26	1.92	3.98	2.41	*		
Because I already knew how to get the solution	3.75	2.05	3.70	2.15	-		
Because InnoCentive told me about this Challenge	3.80	2.34	2.80	2.33	*		
Because my work/job at the time was not satisfying	2.19	1.79	2.13	1.96	-		
To try to beat other InnoCentive solvers	3.15	2.21	2.58	2.17	-		
To win the award money	5.44	1.83	5.73	1.38	-		
Eigenvalue						3.1	1.76
Percentage of variance explained (two factor solution)						0.69	0.31
Cronbach Alpha						0.78	0.71

Factor analysis. Varimax Rotation. Horst Correction. Loadings <= 0.4 not retained. Stata version 8

* $p < 0.05$. ** $p < 0.01$

Table 2.8: Correlations Between Variables Predicting Solver Being A Winner (N = 295 Respondents)

		1	2	3	4	5	6	7	8	9	10
Control Variables											
1	RTP Problem Type	1									
2	Time to develop solution	0.09	1								
Motivations											
3	Money	0	-0.06	1							
4	Extrinsic motivation	0.01	0.07	-0.11†	1						
5	Intrinsic motivation	-0.05	0.05	-0.21*	0.32***	1					
6	Beating other solvers	0	-0.07	-0.02	0.27***	0.26**	1				
7	Unsatisfactory job	0.07	0.01	-0.01	0.18***	-0.04	0.06	1			
8	Had free time	0.01	-0.13*	-0.08	0.05	0.08	0.03	0.25**	1		
Expertise											
9	Interest Count (at registration)	-0,03	-0,06	0.12**	-0.04	0.04	0.05	-0.02	0.03	1	
10	Problem distance with field of expertise	-0,01	0.11†	-0.03	0.05	0.09	-0.01	0.11†	-0.01	0.13*	1

† significant at 10%; * significant at 5%; ** significant at 1%; *** significant at 0.1%

free time", *"unsatisfactory job"* and "*trying to beat others*" motivational items as separate independent variables[13].

I used standardized values of the number of scientific interests they had previously indicated as a measure of the solvers' generalist (more interests) vs. specialist orientation (fewer interests). I also used the self-rated distance between problem and the solvers' own field of expertise as an additional independent variable relating to the solvers' inherent expertise in the problem. I used the submission requirement of the problem (1 = RTP) and time in hours devoted to the problem solving effort as control variables. Table 2.8 has the correlations for all the control and independent variables[14].

I used probit regression models[15], appropriate when the dependent variable is binary, to understand the correlates for being a winning solver. Table 2.9 contains the results of the regression analysis. Note that the type of submission required, RTP or theoretical, is non-significant throughout the analyses (Models 1-6). Solvers are as likely to win when they respond to a RTP problem as they are to a theoretical challenge. However time devoted to the problem solving is highly significant through all of the analyses, indicating that winners are taking more time to create solutions. Model 2 shows the impact of self-reported extrinsic motivations. As expected, being motivated by winning the award money is a significant predictor of creating a prize winning solution. This effect gets stronger and stays significant throughout the remaining models. Pressure from work and social situations to participate in problem solving has a marginally negative impact on being a winning solver. This result is also consistent through all of the remaining models. Model 3 shows that intrinsic motivation has the largest and most significant effect in predicting who becomes a winning solver. The effect remains highly

[13] I did not retain "being told by InnoCentive about the problem" as a motivation because all the solvers were informed about the problem by InnoCentive and it was not something that I considered as being crucial to the analysis. "Knowing how to get the solution information" as a motivation was also not considered because, in retrospect, it was more a statement of a person's knowledge state and not considered a motivations factor. To test the validity of the decision I ran regression analyses with these items as additional independent variables. The inclusion of these items did not change the significance, sign or magnitude of the main findings.

[14] The appendix contains information on statistical methods and variable construction for this regression.

[15] I also ran the regressions under a logit specification and obtained substantively similar results.

Table 2.9: Probit Regression Predicting Which Solver Submits A Winning Solution (N=295 Respondents)

	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
Control Variables						
RTP Problem Type	0.099	0.096	0.131	0.132	0.161	0.188
	(0.214)	(0.215)	(0.221)	(0.221)	(0.228)	(0.233)
Time to develop solution	**0.002***	**0.002****	**0.002****	**0.002****	**0.003****	**0.002***
	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)	(0.001)
Motivations						
Win award money		**0.163†**	**0.217***	**0.230†**	**0.245***	**0.263***
		(0.096)	(0.097)	(0.122)	(0.107)	(0.107)
Social and work related motivations		-0.141	**-0.256***	**-0.257***	**-0.229***	**-0.243***
		(0.099)	(0.113)	(0.113)	(0.111)	(0.112)
Intrinsic motivations			**0.321****	**0.332****	**0.359****	**0.374****
			(0.098)	(0.113)	(0.107)	(0.115)
Beating other solvers					**-0.209†**	**-0.206†**
					(0.114)	(0.117)
Unsatisfactory job					-0.007	-0.033
					(0.130)	(0.134)
Had free time					**0.255***	**0.284***
					(0.118)	(0.119)
Expertise						
Interest count (at registration)						**-0.179†**
						(0.092)
Problem distance from field of expertise						**0.209***
						(0.104)
Interaction Effect For Motivations						
Money X Intrinsic				-0.025		
				(0.120)		
Log Pseudolikelihood	-98.428	-96.491	-92.711	-92.7	-87.7512	-85.2697
Wald's Chi Square	6.15*	11.48*	22.85***	22.71***	28.29***	32.37***
Df	2	4	5	6	8	10
Pseudo R Square	0.0281	0.0473	0.0846	0.0847	0.1336	0.1581

Robust standard errors in parentheses

† significant at 10%; * significant at 5%; ** significant at 1%; *** significant at 0.1%

significant through the remaining analyses and also remains the strongest predictor. It is interesting to note the significant presence of both money motivation and intrinsic motivations in the remaining models. Model 4 shows that the interaction effect between money and intrinsic motivations is non-significant. Thus these motivations, in the context, are neither complementary nor are they crowding each other out. Instead, the significantly negative correlations (Table 2.8) between the two variables indicates that the solver population has individuals that are driven by either money or intrinsic motivations but not necessarily both. Both of those motivations are significant correlates of success in creating winning solutions. Model 5 indicates, not surprisingly, that having free time is a significantly positive correlate of being a winning solver. This is also consistent with the finding that winners tend to spend more time developing solutions. Observe that being motivated by beating other solvers is marginally and negatively correlated with creating a winning solution.

Model 6 introduces variables related to solvers' self-assessment of the distance between the problem and their field of expertise and their overall orientation as a generalist or specialist. Counter-intuitively, I find that the self-assessed distance between the problem and the solvers' field of expertise is positive and significant. Thus winning solvers, on the margin, are solving problems that are further away from their own field of expertise then non-solvers. I expected that winning solvers were solving problems that were directly in their field of expertise. But this finding along with the high reuse of previously developed solutions (Table 2.6) suggests that winning solvers are able to re-apply and modify solutions developed in different fields to the problem at hand. This model also shows that winning solvers tended to have fewer scientific interests as compared to non-winning solvers thus those with a specialist intellectual orientation did better at problem solving than generalists.

2.6.4: Determinants of a Problem Being Successfully Solved

What predicts whether a problem will be successfully solved or not? Recall that the success rate of obtaining a winning solution for a problem was 29.5%, even though each problem received, on average, 10 potential solution submissions. The next set of

analyses tries to unpack the characteristics of the problems and the solver base that are related to the probability of a problem being solved. Once again I used Probit regressions to conduct the analyses.

Table 2.1 shows that there is quite a bit of heterogeneity in solving rates for the various types of problems. Therefore problem characteristics like scientific discipline, RTP vs. Paper solution requirements, award size, and the number of days a problem was open were included in the analyses. Seeker firms posting multiple problems may have built up experience and learning about problem articulation and decomposition, proper award size and time limits that may be effective in a broadcast search setting. Thus the number of previous problems broadcasted by a particular firm may have an effect on current problem solving potential.

Broadcast search aims to initiate problem solving effort by a relatively large set of people in heterogeneous domains. Therefore the overall size of the solver network and the number of submissions received per problem are relevant independent variables in the analyses. The likelihood of the problem being solved may also be dependent on the diversity of scientific interests in the solver base attracted to the problem. Some problems may attract solvers with similar scientific interests and others with substantially different interests. For these purposes I used the number of distinct scientific interests indicated by solvers as a measure of the intellectual diversity attracted to solve the problem. Similarly problems may attract generalist or specialist problem solvers to engage in creating a solution. A generalist solver base may be able to leverage many different scientific domains and apply them to the problem at hand, whereas, a specialist solver base may be more easily able to use their deep understanding of a few scientific fields to create a unique solution. I measured the generalist/specialist orientation of the solver base attracted to the problem by taking the average number of interests represented by the solvers submitting a solution. Table 2.10 contains the correlations of the variables considered in the analyses.

Table 2.10: Correlations Between Variables Predicting Problem Being Solved (N = 132 Problems)

	1	2	3	4	5	6	7	8
Problem Characteristics								
1 RTP Problem Type	1.00							
2 Award Value	0.69***	1.00						
3 Days Problem Open	0.38***	0.43***	1.00					
Seeker Firm Experience								
4 Previous problems posted by seeker firm	0.07	0.03	0.24**	1.00				
Solver community								
5 Solver base size	-0.16†	-0.18*	-0.18*	-0.18*	1.00			
6 Number of submissions	-0.31***	-0.2*	-0.12	-0.04	0.13	1.00		
Types of Solvers Attracted								
7 Distinct scientific interests attracted	-0.22**	-0.14	-0.16†	-0.12	0.57***	0.39***	1.00	
8 Generalist orientation of solvers	0.07	0.10	-0.25***	-0.15†	0.41***	-0.13	0.59***	1.00

† significant at 10%; * significant at 5%; ** significant at 1%; *** significant at 0.1%

Table 2.11: Probit Regression On Problem Being Solved
(N=132 Problems)

	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7
Problem Characteristics							
RTP Problem Type	**-0.295***	-0.07	0.002	-0.008	0.189	0.305	0.246
	(0.130)	(0.184)	(0.191)	(0.199)	(0.211)	(0.224)	(0.223)
Award Value		**-0.370†**	-0.28	-0.273	-0.288	-0.256	-0.362
		(0.217)	(0.229)	(0.225)	(0.237)	(0.247)	(0.238)
Days Problem Open			**-0.430***	**-0.565***	**-0.659****	**-0.956****	**-0.782****
			(0.188)	(0.220)	(0.225)	(0.278)	(0.243)
Seeker Firm Experience							
Previous problems posted by seeker firm				**0.316†**	0.319	**0.362†**	**0.367†**
				(0.186)	(0.198)	(0.204)	(0.195)
Solver community							
Solver base size					-0.295	-1.044	-0.755
					(0.597)	(0.637)	(0.617)
Number of submissions					**0.692****	0.041	**0.457***
					(0.225)	(0.175)	(0.199)
Types of Solvers Attracted							
Distinct scientific interests attracted						**1.260****	**0.565****
						(0.367)	-0.212
Generalist orientation of solvers						**-0.915****	
						(0.326)	
Log Pseudolikelihood	-69.86	-68.01	-65.03	-63.44	-58.06	-50.52	-54.16
Wald's Chi Square	22.62*	27.87**	39.99***	37.99***	56.99***	57.02***	57.04***
df	12	13	14	15	17	19	18
Pseudo R Square	0.16	0.18	0.22	0.24	0.30	0.39	0.35

Robust standard errors in parentheses, Includes controls for years and scientific disciplines (not shown for space reasons)

†significant at 10%; * significant at 5%; ** significant at 1%; *** significant at 0.1%

Table 2.11 shows the results of the probit regressions[16]. All of the variables are standardized, allowing for comparisons of strength of effects. Models 1-3 introduce the variables related to the overall characteristics of the problems. I do find that in Model 1, when I only consider the scientific disciplines and year effects with the solution type requirements, that paper problems are significantly more likely to be solved. However in Model 2, when I include award value, the solution type requirement is no longer significant and surprisingly the coefficient on award value is negative and marginally significant. In Model 3, I find that that the type of solution required (RTP vs. Theoretical) and the pre-announced award are not significant predictors of the ability of the solver community to create a winning solution. One would have expected that paper problems were more likely to be solved as compared to RTP and that larger cash awards would create more incentives for solvers to create a solution. Instead the only significant and negative correlate is the pre-defined time window in which a problem is open and solutions accepted. The negative effect of time window stays significant throughout the remaining models. Discussions with IC's scientific operations staff indicated that the time window could be considered a proxy for problem complexity as viewed by the seeker firm. Thus more complex problems get a larger time window for resolution and are also less likely to be solved[17].

Model 4 brought in the previous number of problems posted by the firm as an independent variable[18]. I find marginally positive significance for this effect. The effect was barely non-significant ($p = 0.101$) in Model 5 and then marginally significant ($p = 0.076$) in the final model. I also ran a fixed effects regression model[19], comparing within firms, to determine if prior experience with broadcast search impacted firm learning about appropriate solution type, reward size, and solution time windows. The results

---

[16] I also ran the regressions under a logit specification and obtained substantively similar results.
[17] I also created and tested various composite variables that had different combinations of solution type, award value and time window. The overall magnitude and significance of the main effects and findings were not impacted by the inclusion of these variables.
[18] Problems posted within 30 days prior to the posting of the focal problem were not included in this measure.
[19] This was done on the full model under a logit specification. Stata Version 8 does not allow for fixed effects in a probit specification.

indicated consistently non-significant effects for solution type and reward size and a more significant effect (p =0.025) for the number of previous problems posted.

Model 5 shows the impact of the total solver base and number of submissions received on the likelihood of a problem being solved. I find that the total size of the solver base is not a significant predictor, but as expected, the number of solution submissions received is a significant positive predictor. Model 6 includes independent variables related to the types of solvers attracted to the problem. I find that the greater the number of distinct interests as reported by the solvers who are submitting solutions the higher the probability of the problem being solved. This is the most significant and strongest effect in the model. I also find that problems that attract relatively more specialized solvers are more likely to be solved. Note that in this model the number of submissions received is now non-significant. This may be due to the relatively high degree of correlation ($r = 0.59$, $p < 0.1\%$) between number of submissions received and the average number of interests per solver per problem (the generalist orientation). Model 7 shows that number of submissions is significant again with the removal of the generalist orientation of the solvers variable.

2.7: Discussion

2.7.1: Cross-fertilization of Scientific Fields

One of the most counter-intuitive findings was the positive and significant impact of the self-assessed distance between the problem and the solvers' field of expertise on the probability of being a winning solver. This finding implies that the further the solvers assessed the problem from their own field of expertise, the more likely they were to create a winning submission. A concern with this self-assessment may be the ability of the solvers to accurately rate the distance between the problem and their field of expertise. This concern can be mitigated by recalling that a majority of the solvers have doctorates in scientific disciplines and winning solvers spent, on average, over 70 hours creating a submission to the problem. Given their specialized background and training and the real effort expended in creating a solution, their self assessment is probably an accurate judgment of the distance between their own expertise and the problem. I reason

that the significance of this effect may be due to the ability of "outsiders" from relatively further fields seeing problems with fresh eyes and applying solutions that are novel to the problem domain. In many instances seeker firms had exhausted internal problem solving efforts with subject matter experts on these problems and were seeking alternative solutions. Therefore the likelihood of a similar type of external expert creating a winning solution can be expected to be lower as compared to external experts from other fields attempting to solve the problem. A recent example saw an aerospace physicist, a small agribusiness owner, a transdermal drug delivery specialist and an industrial scientist all submitting solutions to the same scientific question and each winning an award for their solution. In another case a seeker firm's research and development laboratory did not understand the toxicological significance of a particular pathology that they had observed in an ongoing research program. They broadcasted their problem via IC and it was eventually solved by scientist with a PhD in protein crystallography using methods common in her field. This particular solver would normally not be exposed to toxicology problems or solve such problems a routine basis; however, in this case, she successfully applied knowledge from crystallography to toxicology.

An extreme interpretation of this finding would articulate that the best way to solve problems is to have experts from vastly different fields attempt solutions. Our claim would be that even if the scientific fields are different, for example crystallography and toxicology, there are probably explicit and implicit overlaps between them. In the explicit case, since both these fields are related to biology, we can expect overlap in training and methodological experience amongst the practitioners so that a person from a neighboring discipline can make the leap and create a solution. In the implicit case, modern finance theory offers a good example of cross fertilization with physics. The development of modern finance theory is directly linked to the insights of M. F. M. Osborne, a physicist in the U.S. Navy, who realized in 1959 that financial market prices followed the equations of Brownian motion that Albert Einstein and Norbert Wiener had developed many years earlier (Chance and Peterson 1999; Osborne 1959). Furthermore, the original solution to the Black-Sholes option pricing model relied on a extremely complex calculation of parabolic partial-differential equation based on the premise that

stock prices exhibit Brownian motion. However, Black later realized that the complex equation could be easily transformed into the heat-diffusion equation of thermodynamics, for which the solution was already well known and understood(Chance and Peterson 1999).

Underlying this ability to transform knowledge from one field to another is the role of analogy in reasoning (Dunbar 2001). "In-vivo" studies of scientists working in laboratory settings have shown extensive use of analogies while problem solving. Dunbar has found that biologists use analogies for formulating theories, designing experiments, giving explanation to other scientists and when dealing with unexpected findings. Dunbar measured the distance between the source and target of the analogies and found that the vast majority came from either highly similar fields or fields from common superordinate categories. Recall, however in this context, that solvers were attempting to create solutions to problems that were, on average, at the boundary of a their own field of expertise. Thus I can anticipate that the solvers were able to analogize a solution based on their expertise in their own native field and the problem's field. Contrary to laboratory-based experimental findings on the failure of analogies in problem solving, analogies in naturalistic settings work because problem solvers utilize much richer information on structural features and higher order relations as compared to superficial similarity between source and target (Dunbar 2001). The ability to generate an analogy is not necessarily dependent on having subject matter expertise on both target and source fields. Instead the problem solving context must highlight the structural relations which can then be equally used by novice or experts in creating the analogy and the ultimate solution. Note that in this context, winning solvers could be considered relative novices in the target field but experts with the source field.

Our findings show that a central feature of broadcast search is the relatively high degree of reuse and recombination of previously developed solutions in the creation of new submissions. A majority of the winning solvers (55%) had indicated that they had made major modifications to pre-existing solutions in their submissions with 17.5% reporting a direct port of previously developed solutions with little or no modifications.

In light of the above discussion on the distance between the solvers' field of expertise and the problem, I speculate that broadcast search exposes potential solvers to problems that they do not routinely encounter and that it triggers creative analogizing between their own expertise and knowledge base and the "new" distant problem. This burst of creativity causes the solvers to either significantly modify and recombine pre-existing solution knowledge that they already had or come up with entirely new ideas (as done by 27.5% of the solvers) for effective problem resolution[20]. The net benefit of the juxtaposition of a solver and a distant problem is the creation of a solution that is both novel to the seeker and also to the solver. It also implies that problem broadcasting to heterogeneous solvers can trigger creative cross-fertilization of scientific disciplines and knowledge.

2.7.2: Attracting Many Heterogeneous Solvers

My basic framework for broadcast search posited that increasing number of solvers and the *intellectual* heterogeneity in the solver base would positively contribute to a problem being solved. I have empirical support for both conjectures. I find that increasing number of submissions is significantly and positively related to a problem being solved. This also consistent with a central maxim in open source communities, Linus' Law (Raymond 1999), states that: "given enough eyeball, all bugs are shallow." Thus the larger the solver population attempting solutions the more likely an appropriate solution will be found. I also find that problems that are able to attract submissions from a solver base with a more heterogeneous set of intellectual scientific interests are significantly more likely to be solved. However, the average number of interests per solver per problem is significantly and negatively correlated with solvability. Thus, problems that attract specialized solvers (those that select a lower number of interests) are more likely to be solved. The findings on the specialist orientation of solvers is further corroborated by noting that likelihood of a solver creating a winning solution was negatively and significantly effected by the number of scientific interests selected at registration time with IC by the solver. This indicates that an important inherent characteristic of solved problems is the ability to attract specialist solvers from

[20] I would like to thank Teresa Amabile for pointing out this interpretation of the findings.

heterogeneous fields. It remains a puzzle as to why diverse specialists are attracted to a certain set of problems? It could be that the problems that attract diverse specialist solvers are somehow more easier to solve than those that attract more homogenous specialists. Or these problems could be simply be more interesting to a solver base with heterogeneous interests thus inducing more solution effort. It may also be that solutions for these problems could be more amenable via a variety of different fields thus generating a diversity of solutions and approaches of which one would meet the requirements.

2.7.3 Solver Motivations to Participate

Broadcast search in scientific problem solving exhibits the ability to attract solvers who have varied motivations to participate. The probability of being a winning solver was significantly correlated with both a desire to win the award money as well as intrinsic motivations like enjoying problem solving and being intellectually challenged. Surprisingly, even though there was a substantial prize award for creating the best solution, the effect of intrinsic motivation was stronger and more significant. The stronger effect of intrinsic motivation is consistent with theory and empirical findings which indicate that scientists have a "taste" for science (Stephan and Levin 1992; Stern 2004). I did not find the interaction between intrinsic motivation and a desire to win the award money to be a significant predictor of being winning solver. The significant negative correlation between intrinsic motivation and the financial motivation indicates that broadcast search attracts both financially driven and intrinsically motivated solvers with no crowding out between the motivations. This finding is consistent with results from studies of scientist working in inside of commercial and university laboratories (Stephan and Levin 1992) and other distributed innovation systems, for example open source software, where multiple motivations have been reported to have co-exist (Hertel, Niedner and Herrmann 2003; Lakhani and Wolf 2005).

2.7.4: No One Genius Solver

It also appears that solvers have a unique interaction between the context of the problem and their prior experience and solution that allows them to (re)combine existing solutions or create brand new submissions into a submission for a broadcast search

problem. Since there are no "super solvers" in the population, i.e. individuals who are successfully solving a majority of the problems, I speculate that the uniquely prepared mind of the winning solver is a matter of circumstance and timing and not a general purpose skill. The lack of high numbers of repeat solvers shows that solution knowledge is widely distributed and no-one person or organization can claim exclusive access to it or be successful in solving all the problems.

2.7.5: Seeker Learning and Structuring Ill-Structured Problems

Finally note that seeker firm experience with broadcast search is positively and significantly correlated with the probability of a problem being solved. Seeker firms engaging in broadcast search must do the following four activities: 1) Develop an appropriate articulation and decomposition of the problem and the solution requirements, 2) Determine reward size, 3)Determine time window for solution acceptance and 4) Assess submission and select, if appropriate, a winning entry. These activities are not routinely conducted in a firm's R&D labs and thus firms may learn over time with more problems how best to do all of them. I found an overall marginally positive effect of firm learning (as measured by the number of problems previously posted) on the probability of a problem being solved. The fixed effects regression model, which isolates within-firm variation, showed that firms with multiple broadcast search problems exhibited an even more significant learning effect of prior problems on current problem solvability. Within the fixed effect model, the coefficient on the reward size was non-significant, leading me to speculate that the primary seeker firm learning were related to firm specific skill development in problem articulation and decomposition. IC scientific operations staff, in discussing this finding, have told me that scientists inside seeker firms value the upfront and separated problem and solution definition stage. For example, a seeker firm, while searching for its next generation commercial product had come to a significant technical impasse on further directions for research. They decided to use IC as a means to overcome the impasse. In developing the problem statement, in collaboration with IC, for broadcast, the firm realized that molecules with the ability to spontaneously break protein cross-links were critical to the solution and their technical impasse. Posting this problem statement on IC eventually provided them with a new solution and approach on how to

break such cross-links. The solution provided both a novel mechanism for initiating such breaks and a new class of molecules that operated via this mechanism (Raynor and Panetta 2005). Generally speaking, the critical first step for success in a broadcast search environment is to provide structure to ill-structured problems. I speculate that such an explicit problem structuring process is not routine inside of firms and thus experience with this approach may allow for learning over time on how to accomplish it. Furthermore, this learning around problem articulation may be the driver that allows specialists from different fields to make the connection between their own domain knowledge and the problem.

2.7.6: Value to Seekers

Overall, in this context, broadcast search enables 29.5% of problems to be considered solved by solution seekers. At this point, I am not able to judge the performance of this solution rate with those of firm-based R&D or university-based science laboratories. I have been unable to find any comparable published data which allows me to make comparisons. Most labs are loath to publicize their micro-performance data due to concerns around intellectual property and competition for internal and external funding. Anecdotal conversations with firm and university-based laboratory directors indicates that the performance of broadcast search is equivalent to or exceeds traditional internal only problem solving. However the 29.5% resolution rate may be quite remarkable, if we consider that many of the seekers posting problems had originally been unsuccessful in creating internal solutions[21].

The interest and effort exerted by outsiders in solving broadcasted problems is also quite noteworthy. Most seekers would not be able to find an internal problem solving process that generated 240 scientists to examine a problem statement and 10 of them to actually create solutions. In rare cases there may be two or three parallel attempts at solution creation for the same problem. But it would be highly unlikely to get between three to five times the number of parallel attempts on a routine basis. Translating

[21] Secrecy and intellectual property concerns typically prevented the firms from disclosing to IC – the true extent of problem solving effort within their internal labs. Often times, due to job protection concerns, scientists inside of firms would broadcast "very difficult" and hard to solve problems to IC's solver network.

this into actual time spent shows that solved problems and unsolved problems are, on average, inducing approximately 700 and 245 hours of solver effort respectively. Wage data[22] from the US Bureau of Labor Statistics indicates that the mean annual wage for a scientist working in life and natural sciences is approximately $75,000 or $36/hour. The total hourly cost of a scientist given benefits and lab space requirements could be conservatively estimated at $72/hour. Thus the implied cash benefit to seeker firms is in the range of $20,711 ($50,400 of effort less $29, 689 average award size) for solved problems and $17,460 for unsolved problems.

We can expect that the value of the solved problems is most likely higher than the difference between the implied effort exerted by the solvers and the average award size. Most seeker firms have not provided IC with a downstream assessment of the value of the solved solution due to commercial proprietary concerns. However, one seeker firms, on the condition of anonymity, did allow a downstream value assessment of 12 solved problems (24.4% of solved problems) by an independent consulting company. Majority of the problems analyzed required a reduction to practice solution with an average award of $30,000. The study indicated that the gross downstream value creation of solved problems was $10.3 million. The seeker awarded $333,500 in prize money and incurred additional internal administrative costs of $60,000 resulting in a return on investment that exceeded 2, 175% (Raynor and Panetta 2005).

There is also significant value in unsolved problems. While the firm cannot put into practice the knowledge from submitted solutions from unsolved problems, just knowing that on average seven external people attempted solutions and a further 228 individuals examined the problem statement should inform the seeker about the "solvability" of a problem. Some problems may simply not be solvable given the current state of scientific and technological knowledge and broadcast search provides seekers with an external and unbiased evidence of problem solvability. This information can be very valuable because it allows seekers to shift resources away from "unsolvable" problems to other more amenable areas. In addition, in this model, the marginal cost of

[22] Available from http://www.bls.gov/oes/current/naics5_541710.htm#b19-0000, visited on July 26, 2005

this negative information is zero because firms only pay out the reward money if they have find an acceptable solution.

2.8: Implications

In this paper, I set out to explore the efficacy and functioning of broadcast search in scientific problem solving. Broadcast search inverts the typical problem solving process by focusing the efforts of the problem holders into attracting "complete" solutions from external sources instead of creating solutions themselves. Problem holders transform from problem solvers to become solution seekers by defining and broadcasting problems, attracting many potential solvers from different domains and evaluating solution submissions. Broadcast search overcomes the limitation of local search by distributing the problem into different domains and inducing many solvers to engage in problem solving based on their own expertise and skill sets. I discuss implications for distributed problem solving, for organizations engaged in science and technology work and for the general practice of science.

2.8.1: Implications for Distributed Problem Solving

Broadcast search may be applicable only under specific problem solving conditions. Foss & Foss (2004) have argued that two of Simon's seminal papers "The Architecture of Complexity" (1969) and "The Structure of Ill Structured Problems" (Simon 1973) provide a unique lens on possible problem solving approaches in organizations. Simon's first paper (1969) created a classification of complex systems as being either *decomposable, non-decomposable and nearly decomposable*. With decomposability being a function of the level of interactions amongst subsystems within a system. Nickerson and Zenger (2004) transform this taxonomy to problem classification. In their perspective problem decomposability is a function of the level of interactions amongst knowledge sets within a solution landscape. Decomposable problems are those that do not have a large number of interactions across various knowledge domains and that these problems can be sub-divided so that each of the smaller problem can draw on exclusive knowledge domains. Non-decomposable problems are those where there is a very high degree of interaction amongst distinct knowledge set and attempts to sub-divide the problem is no better than a random search

across the solution space. In between are nearly decomposable problems where the interactions are weak but not negligible.

In Simon's second paper he made the strong contention that: *"In general, the problems presented to problem solvers by the world are best regarded as [ill-structured problems] (ISPs). They become [well-structured problems] WSPs only in the process of being prepared for the problem solvers. It is not exaggerating much to say that there are no WSPs, only ISPs that have been formalized for problem solvers."* Thus WSPs are the result of a problem-defining process for ISPs (Foss and Foss 2004) and that there is a continuum of problem types between ISPs and WSPs.

Foss and Foss (2004) reason that when real-world problem-solving processes are considered the two problem dimensions of decomposition and structured 'ness are interdependent instead of independent. Problem solving typically proceeds with decomposition and often in the process of decomposition the nature of interdependencies within a problem space is brought to light (Schaefer 1999). Thus the act of decomposition enables problem structuring as well. Similarly, one way to move a problem from ISP to WSP is to impose constraints on the problem. Simon argues that constraints help define a problem, example when we consider building a house we may impose a constraints like style, size and location, thus creating both a structure but also the related sub-problems that need to be tackled. The interdependent nature of both decomposition and structure is typically resolved through successive iterations of problem solving effort – where the first stage in the solution search yields new information on both constraints and sub-problems – meaning that a fully decomposed problem cannot be laid out from the beginning and a joint process of problem definition and solution generation needs to take place (Foss and Foss 2004).

In our context, broadcasted problems were discrete, with well-defined constraints and requirements for solutions, and were articulated in a way that was accessible to diverse outside solvers. In other words broadcasted problems were both nearly "well-structured" and suitably well decomposed. Thus finding a "cure for cancer" would not be

a suitable broadcast problem but finding "a method for isolating a specific gene" might be reasonable. Therefore a significant limitation for broadcast search is the ability of the problem holder to engage in up-front problem definition without investing in solution generation. This implies that only certain problems that fit the criteria of decomposition and structure will be suitable for broadcast. Understanding how this occurs and its applicability to all types of problems would be a significant contribution to the literature on distributed problem solving.

Closely related to problem decomposition and structure is the actual articulation of the problem so that solvers in heterogeneous domains can create possible solutions. The principle challenge here is overcoming the tendency to use native codes in communication so that outsiders cannot access them (Katz and Kahn 1966). The challenge here is three-fold; problem articulations need to be general enough so that individuals outside organizations, in different scientific fields and in different countries can understand the challenge and create a solution.

Broadcast search relies on solvers' willingness to take on a problem solving task without a guaranteed reward. Hence it is also important to consider both the intrinsic motivations and extrinsic incentives for the potential solvers — in this case both intrinsic motivations and monetary rewards were significant correlates of being a winning solver. This may not be possible in other contexts. For example, non-profit organizations may not be able to offer significant financial rewards for external problem solving but could benefit from the help and solutions from outsiders. They could spur participation by appealing to potential solvers' sense of altruism and social contribution (Frey 1997). Similarly an overtly monetary focus may be counter-productive as some psychology studies have shown that extrinsic rewards may actually dampen intrinsic motivations resulting in lower productivity (Deci, Koestner and Ryan 1999). Other types of rewards may include simple recognition of the submission by non-winners, feedback on performance to improve learning by comparing winning solutions and non-wining entries and creating a sense of community so that solvers can self-identify with others and

participate for the social aspects (Lindenberg 2001) of belonging to a global solver network.

2.8.2: Implications for Organizations

In this paper I am not claiming that broadcast search should be the only problem solving strategy to be used by organizations. Rather I propose that organizations need a portfolio of search strategies and that they decide during the problem solving process which search strategy may be most appropriate. The portfolio approach implies that managers understand that the bias in the organization will be to rely on local search. Local search and its variants may be appropriate for the vast majority of problems encountered inside the firm. However, problems that appear novel to the organization or are not easily solved by internal experts, may be appropriate for a broadcast search approach. Determining which strategy and its timing may not be trivial matters for both mangers and scholars. In contrast, Nickerson and Zenger (2004) propose a knowledge-based theory of the firm where the complexity type of the problems faced by the organization, as defined by Simon, drives the design of the firm. In their theory, fully decomposable problems are best suited for local search via a market mechanism, near decomposable problems can utilize within-firm local search and/or cognitive search (Gavetti and Levinthal 2000) in an authority-based hierarchy and non-decomposable problems require cognitive search exclusively in a consensus-based hierarchy. They propose these choices as being exclusive at any one point in time, i.e. a firm can only take on form. For them, firm structure is based on the types of problems faced by the firm and that the managers in the firm should optimize their organization to the problem type. Their theory does not allow for multiple search strategies within the same firm. I contend that the likelihood of real firms facing only one problem type is going to be fairly low and that most firms are better off considering a portfolio approach to their problem solving challenges.

The absorptive capacity literature has equated problem solving and learning as being indistinguishable from each other (Cohen and Levinthal 1990; Zahra and George 2002). As I have discussed, broadcast search separates problem definition from solution

finding, with the solving component relegated to outsiders. This may raise the concern that organizations that are overly reliant on broadcast search may not get the essential learning experiences that arise from engaging fully in a problem solving process. An organization's capacity for future problem solving in related fields may suffer if it is utilizing the submitted solutions as "black box" modules within its development efforts. This black box approach may create holes in the essential knowledge structures of the organization that may be hard to replicate or even comprehend. The risk of broadcast search without serious effort at knowledge absorption is that the firm may eventually hollow out its future problem solving capability.

Organizations may also consider doing broadcast search-based problem solving within its own boundaries and/or with trusted suppliers and partners. Research has shown that search and transfer of knowledge within firms is a function of the relationship between relationship strength and knowledge codification. Hansen (1999) has shown that, within a multi-unit organization, weak ties help to identify novel sources of knowledge but strong ties are required for effective transfer of non-codified knowledge. This creates a paradox for the problem solver/searcher because often they will not know ex-ante what type of knowledge is required and from where. However, it may be possible to utilize broadcast search within organizations so that potential solvers match themselves to problems instead of the problem holder searching for solutions. Given a large enough organization, there may be sufficient diversity of intellectual interests and abilities that broadcasting with the bounds of the organization may prove to be fruitful. Concerns around incentives to share knowledge and to solve problems for others will likely remain, however, in principle large diverse organizations may be able to utilize broadcast search within their confines. Alternatively, Malone (2004) has proposed bringing markets and quasimarkets inside of organizations in order to achieve the benefits of dispersed knowledge and decentralization for a range of activities from funding new ideas to joining new project teams and allocating manufacturing capacity.

2.8.3: Implications for Science

Thomas Kuhn theorized that scientific progress occurred through scientific revolutions where existing paradigms are overturned and replaced by new paradigms (Kuhn 1996). Kuhn's mechanism for revolution within a scientific discipline was a sense of dissatisfaction with long held assumptions by a few scientists in the discipline. These scientist then declare a crisis with "normal science" in their field and then engage in anomaly finding and new theorizing that results in a revolutionary science. The revolutionary science typically does away completely with existing theories and instead posits new theories for understanding in the field. Kuhn felt that progress only occurred when normal science was completely replaced with revolutionary science. Quoting Max Planck, Kuhn noted that: "a new scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die, and a new generation grows up that is familiar with it" (Kuhn 1996: 151).

This view of scientific progress via revolutions that overturn existing paradigms can be reasonably augmented by considering intellectual hybridization between various disciplines and paradigms (Dronamraju 1989). The history of science has shown that innovative solutions to difficult scientific problems can arise when knowledge from one scientific discipline is applied to another (Heisenberg 1962) . For example, Erwin Schrödinger's public lectures (Schrödinger 1951) on the nature of the gene and the potential relationship between physics and biology have been credited with spurring physical scientists to study biology and biologists to apply physical principles to life sciences, resulting in an important contribution towards the initiation of molecular biology (Dev 1990). Similarly the modern field of human genetics was not created by revolution, rather a gradual adaptation of new concepts and methods from diverse fields like biochemistry, cytology, mathematics and embryology set in motion the development of current understanding (Dronamraju 1989). Thus instead of waiting for revolutions, a systematic inclusion of diverse perspectives and heuristics may offer advantages over within-field attempts at problem solving which may be yielding "normal science" results. Broadcast search is one such mechanism that can enable intellectual hybridization across disciplines for current scientific problems. It provides a way for scientists in many

different fields to examine problems from other fields and to investigate the possibility of creating hybrid solutions. Furthermore, since one cannot apriori predict which field or person will be able to make an impact on a particular problem, the wide broadcasting of the problem statement allows scientists to self-select themselves into becoming solvers, instead of the problem holder investigating unknown fields.

Broadcast search is based on the premise of sharing of internal scientific problems with outsiders. It implies that the boundaries between various scientific disciplines need to be more permeable and the flow of individuals, problems and knowledge across boundaries actively encouraged. While openness, sharing, and rejection of secrecy are strong institutional norms in science (Dasgupta and David 1994; Merton 1973), most of this occurs ex-post, i.e. upon publication of results, once the scientific problems have been solved and results obtained. Ex-ante, during the process of scientific problem solving and discovery, there is a high degree of secrecy due to concerns over priority (Hagstrom 1974) and more recently commercial gains (Walsh and Hong 2003). A recent empirical study investigating secrecy, measured as unwillingness to discuss ongoing research with those outside the research group, found that only 14% of experimental biologists were willing to talk openly about their current research (Walsh and Hong 2003). Another survey found that 47% of academic geneticists who asked other researchers for additional information, data or materials regarding ***published research*** reported that at least one of their requests has been denied in the preceding three years (Campbell et al. 2002). Application of broadcast search by the larger scientific community will then require the invention of innovative institutional structures that lower the barriers to ex-ante sharing. Such an example can be found in the domain of software, where the creation of the General Public License (GPL) by Richard Stallman, has been credited with breaking the commercially imposed secrecy amongst computer programmers. The GPL is an intellectual property regime which ensures both priority and credit of authorship and substantially curtails private hoarding of collectively developed innovations. Similar systems for the sciences need to be developed so that practitioners will feel comfortable in both sharing problems with outsiders and solving problems for others. In the context of broadcast search the promise of a substantial

financial and the transfer of IP rights does create a substitute institutional structure, however, this may limit applicability of broadcast search to either commercial firms only or well-funded non-profit research laboratories.

2.9: Future Research

This study on broadcast search opens several possible avenues for research on distributed innovation systems and their use by various types of organizations like firms, non-profits and communities. Research is needed to understand how problem articulation, decomposition and structured' ness occurs. I have noted above that this is most likely a critical skill in being successful at broadcasting problems and attracting solutions. Discovering the elements that allow internal problem holders to translate and transform (Carlile 2004) internal problems across boundaries would be an important step in understanding both the mechanisms involved and the types of problems that may be amenable to this method. In addition an understanding of the problem articulation process and the untangling of the problem definition and problem solution steps may be of general benefit to practical problem solving and the literature on innovation.

A direct comparison between broadcast search and local search based problem solving efforts would help both theory and practice. Problems that were broadcasted in this study were not being simultaneously being worked on by internal scientists. Indeed most of the problems in this study had some degree of internal problem solving effort, mostly failed, prior to being broadcast to outsiders. Indicating a possibly higher level of difficulty for broadcasted problems as compared to internal problems. I propose conducting a single-blind experimental design where the same problem statement is used within a firm as outside and to directly compare effectiveness and approaches used by internal and external problem solvers. In addition the study should also include independent evaluation of problem statements and solutions submitted to eliminate any organization-specific biases.

A further study could examine the consequences to the firms after a problem has been broadcasted and solutions submitted. For problems that were solved, it would be

interesting to explore how the solution was utilized within the organization and the process by which it was commercially exploited. Of particular interest would be if the internal personnel were able to absorb the knowledge embedded in the solution for future solution generation and if they changed their own problem solving routines as a result of the external solution. In other words, was the solution used as a black-box to solve a particular problem or did the solution cause changes in how future problems were solved. For problems that were not solved it would be interesting to investigate if the organizations used that information as signal of problem insolvability or if they were ultimately able to create a working solution.

Broadcast search requires an audience of potential solvers for the receipt of the problems. In this context, IC as the knowledge broker, invested heavily in developing the solver network over 80,000 scientists from around the world and an information technology infrastructure to host the problems and solutions. Research is needed to determine alternative ways of aggregating potential solvers and recipients of broadcasted problems. Are there other means of identifying and reaching potential solvers that do not require a significant investment in infrastructure, for example, could journal citation searches help identify likely candidates for problem resolution? In addition I also need to understand if there are threshold effects in the number of individuals needed in the potential solver network for successful broadcast search. Thus determining if a critical mass of potential solvers is needed for success and the range of growth and new entrants required for sustained solving rates.

Broadcast search may be extended by incorporating elements of "commons-based peer production" (Benkler 2004) where individuals openly collaborate in joint problem solving activity. In this specific application of broadcast search, solvers worked independently and did not share their knowledge and solutions with each other. It may be advantageous to bring problem solvers together and encourage them to collaborate on solutions that leverage multiple knowledge domains. Mathematical modeling and computer simulations have indicated that groups of diverse problem solvers can outperform groups of high-ability problem solvers (Hong and Page 2001; Hong and Page

2004). Experience from open source software projects (e.g.: Linux and Apache) shows that transparent broadcast search-like communication and problems solving norms can effectively self-organize communities creating competitive and robust software (Feller et al. 2005). Thus research that examines collective problem solving in domains outside software would significantly add to our understanding of how such distributed innovation systems work and their application to a broad range of problems.

References

Adamson, R. E. 1952. "Functional fixedness as related to problem solving: a repetition of three experiments." *Journal of Experimental Psychology* 44:288-291.

Adler, P.S., and Kim B. Clark. 1991. "Behind the learning curve: A sketch of the learning process." *Management Science* 37:267-281.

Alchian, Armen. 1963. "Reliability of Progress Curves in Air-frame Production." *Econometrica* 31:679-693.

Allen, Thomas J. 1966a. "Performance of communication channels in the transfer of technology." *Industrial Management Review* 8:87-98.

Allen, Thomas, J. 1966b. "Studies of the Problem-Solving Process in Engineering Design." *IEEE Transactions on Engineering Management* 13:72-83.

Allen, Thomas J. 1970. "Communication networks in R&D labs." *R&D Management* 1:14-21.

Allen, Thomas, J. 1977. *Managing the flow of technology*. Cambridge, MA: MIT Press.

Allen, Thomas J., and Stephen I. Cohen. 1969. "Information Flow in Research and Development Laboratories." *Administrative Science Quarterly* 14:12-19.

Allen, Thomas J., A. Gerstenfeld, and P.G. Gerstberger. 1968. "The problem of internal consulting in research and development organization." *MIT Sloan School of Management Working Paper Series* 319-68.

Allen, Thomas, J., and D. G. Marquis. 1964. "Positive and negative biasing sets: The effects of prior experience on research performance." *IEEE Transactions on Engineering Management* 11:158-161.

Almeida, Paul, and Bruce Kogut. 1999. "Localization of knowledge and the mobility of engineers in regional networks." *Management Science*:905-917.

Anderson, Philip, and Michael L. Tushman. 1990. "Technological Discontinuities and Dominant Designs: A Cyclical Model of Technological Change." *Administrative Science Quarterly* 35:604-633.

Andrewes, William J. H. 1996. "Even Newton could be wrong: The story of Harrison's first three sea clocks." Pp. 189-234 in *The quest for longitude*, edited by William J. H. Andrewes. Cambridge, MA: Collection of Historical Scientific Instruments, Harvard University.

Argote, L., and D. Epple. 1990. "Learning Curves in Manufacturing." *Science* 247:920-924.

Arrow, Kenneth J. 1962. "The economic implications of learning by doing." *Review of economic studies* 29.

Baron, Jonathan. 1988. *Thinking and deciding*. New York: Cambridge University Press.

Benkler, Yochai. 2004. "Commons-based strategies and the problems of patents." *Science* 305:1110-1111.

Berul, L. H., M.E. Elling, A. Karson, A. B. Shafrity, and H. Sieber. 1965. "Department of Defense User needs study." Philadelphia: Auerbach Corporation.

Birch, H. G., and H. S. Rabinowitz. 1951. "The negative effect of previous experience on productive thinking." *Journal of Experimental Psychology* 41:121-126.

Brown, Shona L., and Kathleen M. Eisenhardt. 1995. "Product Development: Past Research, Present Findings, and Future Directions." *Academy of Management Review* 20:343-378.

Campbell, E. G., B. R. Clarridge, N. N. Gokhale, L. Birenbaum, S. Hilgartner, N. A. Holtzman, and D. Blumenthal. 2002. "Data withholding in academic genetics - Evidence from a national survey." *JAMA-Journal of the American Medical Association* 287:473-480.

Carlile, Paul. 2004. "Transferring, translating, and transforming: An integrative framework for managing knowledge across boundaries." *Organization Science* 15:555-568.

Chance, Don M. , and Pamela P. Peterson. 1999. "The new science of finance." *American Scientist* 87:256-264.

Christensen, Clayton M, Fernando F Suarez, and James M Utterback. 1998. "Strategies for survival in fast-changing industries." *Management Science* 44:S207-S220.

Chubin, Daryl E. 1976. "The Conceptualization of Scientific Specialties." *The Sociological Quarterly* 17:448-476.

Clark, Kim B. 1985. "The interaction of design hierarchies and market concepts in technological evolution." *Research Policy* 14:235-251.

Cockburn, Iain M., and Rebecca M. Henderson. 1998. "Absorptive Capacity, Coauthoring Behavior, and the Organization of Research in Drug Discovery." *Journal of Industrial Economics* 46:157-182.

Cohen, Wesley M., and Daniel A. Levinthal. 1989. "Innovation and Learning: The Two Faces of R & D." *The Economic Journal* 99:569-596.

—. 1990. "Absorptive Capacity: A New Perspective on Learning and Innovation." *Administrative Science Quarterly* 35:128-152.

Constant, David, Lee Sproull, and Sara Kiesler. 1996. "The kindness of strangers: The usefulness of electronic weak ties for technical advice." *Organization Science* 7:119-135.

Crane, Diana. 1969. "Social Structure in a Group of Scientists: A Test of the "Invisible College" Hypothesis." *American Sociological Review* 34:335-352.

Csikszentmihalyi, Mihaly. 1975. *Beyond Boredom and Anxiety: The Experience of Play in Work and Games*. San Francisco: Jossey-Bass, Inc.

Cyert, Richard M., and James G. March. 1963. *Behavioral theory of the firm* N.J.: Englewood Cliffs.

Dasgupta, P, and P A. David. 1994. "Towards a new economics of science." *Research Policy* 23:487-524.

Deci, Edward L, R Koestner, and Richard M Ryan. 1999. "A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation." *Psychological Bulletin* 125:627-688.

Deci, Edward L, and Richard M Ryan. 1985. *Intrinsic motivation and self-determination in human behavior*. New York, NY: Plenum Press.

DeMonaco, Harold J., Ayfer Ali, and Eric von Hippel. 2005. "The Major Role of Clinicians in the Discovery of Off-Label Drug Therapies." *MIT Sloan School of Management Working Paper Series*.

Dosi, Giovanni. 1982. "Technological paradigms and technological trajectories." *Research Policy* 11:147-162.

Dronamraju, Krishna R. 1989. *The foundations of human genetics*. Springfield, Ill: Charles C. Thomas.
Dunbar, Kevin. 2001. "The analogical paradox: Why analogy is so easy in naturalistic settings, yet so difficult in the psychological laboratory." in *Analogy: Perspectives from Cognitive Science*, edited by D. Gentner, K.J. Holyoak, and B. Kokinov. Cambridge, MA: MIT Press.
Duncker, K. 1945. "On problem solving." *Psychology Monographs* 58.
Edge, David O, and Michael J Mulkay. 1974. "Case studies of scientific specialties." University of Edinburgh, Science Studies Unit.
Feller, Joe, Brian Fitzgerald, Scott Hissam, and Karim R Lakhani (Eds.). 2005. *Perspectives on Free and Open Source Software*. Cambridge: MIT Press.
Finholt, Thomas, Lee Sproull, and Sara Kiesler. 2002. "Outsiders on the Inside: Sharing Know-How Across Space and Time." Pp. 357-380 in *Distributed Work*, edited by Pamela Hinds and Sara Kiesler. Cambridge, MA: MIT Press.
Foss, Kirsten, and Nicolai J Foss. 2004. "Simon on problem solving: Implications for new organizations forms." *Copenhagen Business School Working Paper*.
Frey, Bruno. 1997. *Not just for the money: an economic theory of personal motivation*. Brookfield. VT: Edward Elgar Publishing Company.
Gavetti, Giovanni, and Daniel Levinthal. 2000. "Looking Forward and Looking Backward: Cognitive and Experiential Search." *Administrative Science Quarterly* 45:113-137.
Gordon, W. J.J. 1961. *Synectics: The development of creative capacity*. New York: Harper and Row.
Granovetter, M. 1973. "The strength of weak ties." *American Journal of Sociology* 78:1360-1380.
Hagstrom, Warren O. 1974. "Competition in Science." *American Sociological Review* 39:1-18.
Hansen, Morten T. 1999. "The search-transfer problem: The role of weak ties in sharing knowledge across organization subunits." *Administrative Science Quarterly* 44:82-111.
Hayek, F. A. 1945. "The use of knowledge in society." *American Economic Review* 35:519-530.
Heisenberg, W. 1962. *Physics and philosophy; the revolution in modern science*. New York: Harper.
Hertel, Guido, Sven Niedner, and Stefanie Herrmann. 2003. "Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel." *Research Policy* 32:1159-1177.
Hong, Lu, and Scott E. Page. 2001. "Problem Solving by Heterogeneous Agents." *Journal of Economic Theory* 97:123-163.
—. 2004. "Groups of diverse problem solvers can outperform groups of high-ability problem solvers." *PNAS* 101:16385-16389.
Horrobin, D. F. 1986. "Glittering prizes for research support." *Nature* 324:221.
Katz, D., and R. Kahn. 1966. *A Social Psychology of Organizations*. New York: McGraw-Hill.
Katz, E. 1960. "The two-step flow of communication." in *Mass communications, 2nd edition*, edited by W Schramm. Urbana: University of Illinois.

Katz, E., and P.F. Lazarsfeld. 1955. *Personal influence*. New York: Free Press.
Kuhn, Thomas. 1996. *The structure of scientific revolutions - 3rd Edition*. Chicago: University of Chicago Press.
Lakhani, Karim R, and Robert Wolf. 2005. "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects." in *Perspectives on Free and Open Source Software*, edited by Joe Feller, Brian Fitzgerald, Scott Hissam, and Karim R Lakhani. Cambridge, MA: MIT Press.
Lakhani, Karim R., and Eric von Hippel. 2003. "How Open Source Software Works: Free User to User Assistance." *Research Policy* 32:923-943.
Lazarsfeld, P.F., B. Berelson, and H. Gaudet. 1948. *The people's choice*. New York: Duell, Sloan, Pierce.
Lee, Denis M. S., and Thomas J. Allen. 1982. "Integrating New Technical Staff: Implications for Acquiring New Technology." *Management Science* 28:1405-1420.
Levinthal, Daniel A., and James G. March. 1993. "The Myopia of Learning " *Strategic Management Journal* 14:95-112.
Levitt, Barbara, and James G March. 1988. "Organizational learning." *Annual Review of Sociology* 14:390-340.
Lewin, K. . 1936. *Principles of Topological Psychology*. New York: McGraw-Hill Book Company, Inc. .
Lieberman, Marvin, B. 1984. "The learning curve and pricing in the chemical processing industries." *The Rand Journal of Economics* 15:213-228.
Lindenberg, Siegwart. 2001. "Intrinsic motivation in a new light." *Kyklos* 54:317-342.
Lovett, Marsha C., and John R. Anderson. 1996. "History of Success and Current Context in Problem Solving:
Combined Influences on Operator Selection
" *Cognitive Psychology* 31:168-217.
Luchins, A. S. 1942. "Mechanization in problem solving: the effect of Einstellung." *Psychology Monographs* 54.
Luchins, Abraham, S., and Edith Luchins, H. 1959. *Rigidity of Behavior. A variational approach to the effects of Einstellung*. Eugene, Oregon: University of Oregon Books.
Malone, Thomas W. 2004. *The future of work : how the new order of business will shape your organization, your management style, and your life*. Boston: Harvard Business School Press.
March, James G, and Herbert Simon. 1958. *Organizations*: Wiley.
Marples, D. L. 1961. "The Decisions of Engineering Design." *IRE Transactions on Engineering Management*:55-71.
McPherson, J Miller. 1983. "An ecology of affiliation." *American Sociological Review* 48:519-532.
McPherson, J. Miller. 1981. "A Dynamic Model of Voluntary Affiliation." *Social Forces* 59:705-728.
McPherson, J. Miller, and Lynn Smith-Lovin. 1987. "Homophily in Voluntary Organizations: Status Distance and the Composition of Face-to-Face Groups." *American Sociological Review* 52:370-379.

Merton, Robert K. 1973. *The sociology of science: Theoretical and empirical investigation.* Chicago: University of Chicago Press.
Meyers, Sumner, and Donald C Marquis. 1969. "Successful industrial innovation." Washington DC: National Science Foundation.
Mockus, Audris, Roy Fielding, and James Herbsleb. 2002. "Two case studies of open source software development: Apache and Mozilla." *ACM Transactions on Software Engineering and Methodology* 11:1-38.
Mowery, David C., Joanne E. Oxley, and Brian S. Silverman. 1996. "Strategic Alliances and Interfirm Knowledge Transfer." *Strategic Management Journal* 17:77-91.
Nagarajan, Anuradha, and Will Mitchell. 1998. "Evolutionary Diffusion: Internal and External Methods Used to Acquire Encompassing, Complementary, and Incremental Technological Changes in the Lithotripsy Industry." *Strategic Management Journal* 19:1063-1077.
Nelson, Richard R., and Sidney G. Winter. 1982. *An evolutionary theory of economic change.* Cambridge, MA: Belknap Harvard.
Newell, Allen, and H.A. Simon. 1972. *Human Problem Solving.* Engelwood Cliffs, New Jersey: Prentice-Hall INC. .
Nickerson, Jack A, and Todd A. Zenger. 2004. "A knowledge-based theory of the firm - the problem solving perspective." *Organization Science* 15:617-632.
Nonaka, Ikujir D., and Hirotaka Takeuchi. 1995. *The knowledge-creating company : how Japanese companies create the dynamics of innovation.* New York: Oxford University Press.
Osborne, M.F.M. 1959. "Brownian motion in the stock market." *Operations Research* 7:145-173.
Popielarz, Pamela A, and J Miller McPherson. 1995. "On the edge or in between: Niche position, niche overlap, and the duration of voluntary association memberships." *American Journal of Sociology* 101:698-720.
Raymond, Eric. 1999. *The Cathedral and the Bazaar: Musings on Linux and Open Source from an Accidental Revolutionary.* Sebastopol: CA: O'Reilly and Associates.
Raynor, Michael E, and Jill A. Panetta. 2005. "A better way to R&D?" *Strategy & Innovation: A newsletter from Harvard Business School Publishing and Innosight* 3:14-16.
Riggs, William, and Eric von Hippel. 1994. "Incentives to innovate and the sources of innovation: The case of scientific instruments." *Research Policy* 23:459-469.
Rosenkopf, Lori, and Paul Almeida. 2003. "Overcoming Local Search Through Alliances and Mobility." *Management Science* 49:751-766.
Rosenkopf, Lori, and Atul Nerkar. 2001. "Beyond local search: Boundary-spanning, exploration and impact in the optical disk industry." *Strategic Management Journal* 22:287-306.
Saugstad, P. 1955. "Problem-solving as dependent upon availability of functions." *British Journal of Psychology* 46:191-198.
Sawhney, Mohanbir, Emanuela Prandelli, and Gianmario Verona. 2003. "The Power of Innomediation." *MIT Sloan Management Review*:6.
Saxenian, AnnaLee. 1994. *Regional advantage : culture and competition in Silicon Valley and Route 128.* Cambridge, Mass.: Harvard University Press.

Schaefer, Scott. 1999. "Product design partitions with complementary components." *Journal of Economic Behavior and Organization* 38:311-330.
Schilling, C. W., and J. Bernard. 1964. "Informal communication among Bioscientists. Biological Sciences Communication Project." Washington D. C.: George Washington University.
Schrödinger, Erwin. 1951. *What is life? The physical aspect of the living cell.* Cambridge, UK: Cambridge University Press.
Sheehan, Kim. 2001. "E-mail Survey Response Rates: A Review " *Journal of Computer Mediated Communication* 6.
Simon, H.A. 1969. *The Sciences of the Artificial.* Cambridge: Massachusetts Institute of Technology.
Simon, H.A., and Allen Newell. 1962. "Computer Simulation of Human Thinking and Problem Solving." *Monographs of the Society for Research in Child Behavior* 27:137-150.
Simon, Herbert A. 1973. "The structure of ill structured problems." *Artificial Intelligence* 4:181-201.
Sobel, Dava. 1996. *Longitude : the true story of a lone genius who solved the greatest scientific problem of his time.* New York: Penguin.
Sorensen, Jesper B, and Toby E. Stuart. 2000. "Aging, Obsolescence, and Organizational Innovation." *Administrative Science Quarterly* 45:81-112.
Staats, A. W. 1957. "Verbal and instrumental repose-hiearchies and their relationship to problem-solving." *American Journal of Psychology* 70:442-446.
Stephan, Paula E., and Sharon G. Levin. 1992. *Striking the mother lode in science : the importance of age, place, and time.* New York: Oxford University Press.
Stern, Scott. 2004. "Do scientists pay to be scientists?" *Management Science* 50:835-854.
Stuart, Toby E., and Joel M. Podolny. 1996. "Local search and the evolution of technological capabilities." *Strategic Management Journal* 17:21-38.
Tushman, Michael L. 1977. "Special boundary roles in the innovation process." *Administrative Science Quarterly* 22:587-605.
Tushman, Michael L., and Ralph Katz. 1980. "External Communication and Project Performance: An Investigation into the Role of Gatekeepers." *Management Science* 26:1071-1085.
Utterback, James M, and Fernando F Suarez. 1993. "Patterns of industrial evolution, dominant designs and firm's survival." *Research on Technological Innovation, Management and Policy* 5:47-87.
Utterback, James M. 1971. "The Process of Technological Innovation within the Firm." *Academy of Management Journal* 14:75-88.
von Hippel, Eric. 1978. "Successful industrial products from customer ideas." *Journal of Marketing* 42:39-49.
—. 1982. "Get New Products from Customers." *Harvard Business Review* 60:117-122.
—. 1988. *The Sources of Innovation.* New York, NY: Oxford University Press.
—. 1989. "New Product Ideas from "Lead Users"." *Research Technology Management* 32:24-27.
—. 1994a. "'Sticky information' and the locus of problem solving: Implications for innovation." *Management Science* 40:429-439.

—. 1994b. "Sticky Information and the Locus of Problem Solving." *Management Science* 40:429-439.
—. 1999. "Economics of product development by users: Impact of "sticky" local information." *Management Science* 44:629-644.
—. 2005. *Democratizing Innovation.* Cambridge, MA: MIT Press.
von Hippel, Eric, and Marcie J Tyre. 1995. "How learning by doing is done: Problem identification in novel process equipment." *Research Policy* 24:1-12.
Walsh, John P., and Wei Hong. 2003. "Secrecy is increasing in step with competition." *Nature* 422:801-802.
Ward, T.B. 1995. "What's old about new ideas?" Pp. 157-178 in *The creative cognition approach*, edited by S.M. Smith, T.B. Ward, and R.A. Finke. Cambridge, MA: MIT Press.
Weiman, Gabriel. 1982. "On the Importance of Marginality: One More Step into the Two-Step Flow of Communication." *American Sociological Review* 47:764-773.
Wright, T.P. 1936. "Factors affecting the cost of airplanes." *Journal of Aeronautical Sciences* 3:122-128.
Yelle, L. E. 1979. "The Learning Curve: Historical Review and Comprehensive Survey " *Decision Sciences* 10:302-328.
Zahra, Shaker A, and Gerard George. 2002. "Absorptive capacity: A review, reconceptualization, and extension." *Academy of Management Review* 27:185-204.

Appendix

1 – Statistical Methods and Variable Construction

Statistical Methods

I used probit regression models to determine the size and strength of relationship between dependent and independent variables. A probit model regression model is appropriate when the outcome variable is binary and is categorical (i.e. the problem was solved or not solved; solver had a winning solution or not a winning solution). The probit model is non-linear with an assumption that errors are normally distributed with a variance of the errors equal to one. Regressions were computed using robust estimates for the standard errors thus allowing our estimates of the standard errors to be "robust" to failure to meet assumptions of normality and homogeneity of variance of the residuals.

Construction of variables in Table 9 – "Who becomes a winning solver?"

All variables were standardized except for time to develop solution.

Dependent variable: Who becomes a winning solver. This data was available from the InnoCentive Database per problem. For each problem there were a number of submissions and the data indicated which person(s) had a winning solution or not

Independent Variables:

RTP Problem Type: Value = 1 if the solution requirement for the problem was a reduction to practice submission. Value = 0 if the solution requirement was a paper submission. Obtained from InnoCentive.com

Time to develop solution: Time in days as reported by solvers required to create a solution. Obtained from web survey of solvers

Motivations: Please see Table 7 – for details on how the motivation variables were derived. The two factors, intrinsic motivation and social and work-related motivation, that were developed from multiple items were constructed by first standardizing (transforming them so that mean = 0 and variance = 1) each of the items and then added and averaged and then further standardized. Obtained from web survey of solvers.

Interest Count: Number of scientific interests indicated by solver when first registering with IC – from a list of 56 options. Obtained from InnoCentive.com.

Problem distance from field of expertise: Based on the answer to the following survey question: Is the particular challenge: "1 – inside your field of expertise, 4 – at the boundary of your field of expertise, 7 – outside your field of expertise. Respondents could choose any value between 1 and 7.

Construction of variables in Table 11 - "Which Problems Get Solved?"

Dependent variable: Binary variable = 1 if a solution reward was given out | = 0 if no solution award is given out. Data from InnoCentive.com

Dependent variables
RTP Problem Type: Value = 1 if the solution requirement for the problem was a reduction to practice submission. Value = 0 if the solution requirement was a paper submission. Data from InnoCentive.com

Award Value: Actual value in US dollars for the award money for the problem being successfully solved. Data from InnoCentive.com

Days Problem Open: The time window in days between the broadcast of the problem and the deadline for submissions. Data from InnoCentive.com

Previous problems posted by seeker firm: The total number of previous problems broadcasted by the firm on IC. Summed from 30 days prior to the post of the current problem. Data from InnoCentive.com

Solver base size: Total number of registered users on IC website at the time of the posting of the problem. Data from InnoCentive.com

Number of submissions: Number of submissions received at the end of the time window of a problem. Data from InnoCentive.com

Distinct scientific interests attracted: At registration time with IC, solvers were asked about their scientific interests from a list of 56 options spanning Chemistry and Applied Sciences and the Life Sciences. This variable consists of counting the total number of distinct (unique) scientific interests from the solvers who submitted a solution to the problem. Double counts of same the scientific interests by different solvers were eliminated. The higher the number the more unique scientific interests represented in solving the problem. Data from InnoCentive.com

Generalist orientation of the solver: At registration time with IC, solvers were asked about their scientific interests from a list of 56 options spanning Chemistry and Applied Sciences and the Life Sciences. This variable consists of first summing the raw count of scientific interests indicated by the solvers who submitted a solution to the problem. And then dividing this sum by the total number of solvers who submitted a solution. Thus creating the average number of scientific interests per solver per problem. The higher the number the larger the average number of interests per solver per problem and the more generalist an orientation of the solver community that is creating a solution. Data from InnoCentive.com

2 - Solver Survey

InnoCentive Solvers Survey

About the InnoCentive Challenge:

1 **When you first saw the particular InnoCentive Challenge, what was your experience with similar problems?**

1 ○ This problem was completely new to me 2 ○ 3 ○ 4 ○ I was somewhat familiar 5 ○ 6 ○ 7 ○ I had seen the EXACT problem before

2 **Tell us about your experience with these types of Challenges. How true are the following statements?**

a. I have had experience with these types of problems professionally

1 ○ not true at all 2 ○ 3 ○ 4 ○ somewhat true 5 ○ 6 ○ 7 ○ very true

b. I have had experience with these types of problems as a hobby

1 ○ not true at all 2 ○ 3 ○ 4 ○ somewhat true 5 ○ 6 ○ 7 ○ very true

	c. I have had experience with these types of problems as a student	1 ○ not true at all 2 ○ 3 ○ 4 ○ somewhat true 5 ○ 6 ○ 7 ○ very true
3	**Please tell us if this Challenge was**	1 ○ Inside your field of expertise 2 ○ 3 ○ 4 ○ At the boundary of your field of expertise 5 ○ 6 ○ 7 ○ Outside your field of expertise

About your Submission to InnoCentive:

4	**What was your situation when you encountered this Challenge?**	-----select a response from this pulldown menu-----
5	**Sometimes solutions build on previous work. Was your submission to this Challenge based on:**	
	a. A solution you had already developed in your own work with:	1 ○ No modifications 2 ○ 3 ○ 4 ○ Minor Modifications 5 ○ 6 ○ 7 ○ Major modifications ○ NA – This was not based on any of my previous work
	b. An existing solution you knew about that could solve the Challenge with:	1 ○ No modifications 2 ○ 3 ○ 4 ○ Minor Modifications 5 ○ 6 ○ 7 ○ Major modifications ○ NA – This was not based on anyone else's work
6	**How much time did it take you to develop your submission (please estimate hours of effort)**	☐ hours

7	**How much money (not including your own labor) did you spend in developing your submission (e. g.: Money spent on Reagents, Equipment etc)**	☐ --------select currency--------

8	**Tell us about any additional resources you needed as you worked on this Challenge. How true are the following statements? I had to acquire access to:**												
	a. More laboratory equipment than I normally have access to	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
	b. More software than I normally have access to	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
	c. More library and literature resources than I normally have access to	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
	d. More specialty databases than I normally have access to	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true

9	**Did you solve the Challenge as an individual or as a team?**	individual: O team: O
	9a If you solved the Challenge as a team – how many people were on your team?	☐
10	**How many other people did you consult with in your problem solving effort (excluding those that were on your team in question 9a)?**	☐
11	**In your estimation, how many others could have developed a submission similar to yours?**	----------select a response from this pulldown menu----------

Your reasons for participating in the InnoCentive Challenge:

12	**There are many reasons for participating in an InnoCentive Challenge. Tell us how true the following statements are for you. Please answer all items. I submitted a solution:**	
	a. To win the award money	1 O not true at all 2 O 3 O 4 O somewhat true 5 O 6 O 7 O very true

b. Because others I know have participated before	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
c. Because someone suggested I participate in solving this Challenge	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
d. Because my boss asked me to work on it	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
e. To try to beat other InnoCentive solvers	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
f. Because my work/job at the time was not satisfying	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
g. Because InnoCentive told me about this Challenge	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
h. Because I enjoy solving these types of Challenges	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
i. To enhance my skills	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
j. To enhance my career prospects	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true
k. To impress my colleagues	1 ○ not true at all	2 ○	3 ○	4 ○ somewhat true	5 ○	6 ○	7 ○ very true

l.	Because I already knew how to get the solution	1 ○ not true at all 2 ○ 3 ○ 4 ○ somewhat true 5 ○ 6 ○ 7 ○ very true
m.	For the intellectual challenge of solving this Challenge	1 ○ not true at all 2 ○ 3 ○ 4 ○ somewhat true 5 ○ 6 ○ 7 ○ very true
n.	To learn about these types of Challenges	1 ○ not true at all 2 ○ 3 ○ 4 ○ somewhat true 5 ○ 6 ○ 7 ○ very true
o.	To gain scientific recognition	1 ○ not true at all 2 ○ 3 ○ 4 ○ somewhat true 5 ○ 6 ○ 7 ○ very true
p.	Because I had free time available	1 ○ not true at all 2 ○ 3 ○ 4 ○ somewhat true 5 ○ 6 ○ 7 ○ very true
q.	Other	
13	**Would you have attempted to solve this Challenge if there was no financial reward offered?**	1 ○ Most definitely Not 2 ○ 3 ○ 4 ○ Maybe 5 ○ 6 ○ 7 ○ Most definitely Yes
14	**Will you attempt to solve an InnoCentive Challenge in the future?**	1 ○ Most definitely Not 2 ○ 3 ○ 4 ○ Maybe 5 ○ 6 ○ 7 ○ Most definitely Yes
15	**How satisfied were you with your experience with InnoCentive**	1 ○ Highly Satisfied 2 ○ 3 ○ 4 ○ Neither satisfied or dissatisfied 5 ○ 6 ○ 7 ○ Highly dissatisfied
16	**Any thing else you may want to tell us about your experience with InnoCentive?**	

Your Background:

17	**What is your Gender?**	male: ○ female: ○
18	**What is the highest academic qualification you have received?**	----------select----------
19	**What year did you receive your highest degree?**	
20	**What is the name of the institution where you got your highest academic qualification?**	
21	**What is the field in which you have received your highest qualification?**	
22	**What city were you living in at the time of your submission?**	
23	**What was your occupation at the time of the submission of the Challenge (including student or retired)?**	
24	**May Innocentive contact you via email if the MIT-CBS Research Team has any further questions on this topic?**	Yes: ○ No: ○

Submit Survey

3 – List of Problems in Analysis

#	Problem Description	Scientific Discipline	Country of Originating Lab	Award Value (USD)
2062	Cyclohexaneacetic acid	Synthesis	USA	30000
2068	Challenge #2068	Synthesis	USA	80000
2071	Substituted Piperazine	Synthesis	Belgium	50000
3076	Substituted Cyclopentaneacetic Acid	Synthesis	USA	30000
3079	1-Bromo-6-fluoronaphthalene	Synthesis	UK	45000
3082	Chiral 2-Methyl-4-piperidone	Synthesis	Belgium	55000
3085	2-Bromo-6-fluoronaphthalene	Synthesis	UK	45000
3088	Substituted indole	Synthesis	USA	65000
3091	Substituted pyridine	Synthesis	USA	50000
3094	Cyclopentenone	Synthesis	USA	25000
3097	Challenge # 3097	Synthesis	USA	100000
3100	Novel Synthetic Route	Synthesis	USA	50000
3103	Challenge # 3103	Synthesis	USA	90000
3106	Challenge # 3106	Synthesis	USA	65000
3109	4-(4-Hydroxyphenyl) butanoic acid	Synthesis	USA	25000
3112	Challenge # 3112	Synthesis	USA	80000
3115	Substituted thiophene	Synthesis	USA	70000
9517	Challenge # 9517	Synthesis	USA	2000
9520	Efficient Synthetic Strategy	Synthesis	USA	2000
9523	Novel Synthetic Route	Synthesis	USA	2000
24384	Deazaguanine ester	Synthesis	USA	90000
25580	Fmoc-L-Neo-Trp	Synthesis	USA	60000
32295	Fmoc-D-2-Me-Trp	Synthesis	USA	75000
32312	Fmoc-L-2-Me-Trp - enzymatic	Synthesis	USA	105000
39949	(5-aza-benzofuran-7-yl) acetic amide	Synthesis	USA	75000
40742	7-Formyl-Indole	Synthesis	USA	75000
40964	Challenge # 40964	Synthesis	USA	65000
43165	D-glucopyranose	Synthesis	USA	40000
44076	D-xylopyranose	Synthesis	USA	40000
55195	Substituted isoquinoline	Synthesis	USA	20000
59214	Chiral Hexose-nucleoside	Synthesis	USA	60000
62163	4-AZIDO CHIRAL HEXOSE	Synthesis	USA	50000
78991	4-hydroxypyrimidine	Synthesis	USA	15000
79007	Preserved Parenteral Suspension Placebo	Formulation	USA	100000
96229	Regio- and stereocontrolled tricyclic alcohols	Synthesis	USA	5000
113827	Malononitrile - stable label	Synthesis	USA	15000
113837	4-nitroacetophenone - stable label	Synthesis	USA	25000
124675	Surfactant Analysis	Analytical	USA	40000
196513	cis-PTAP	Synthesis	USA	40000
200908	Sulfoethoxylates	Synthesis	USA	2000
205709	Oxidation of parrifins	Synthesis	USA	2000

207090	Branched alcohols	Synthesis	USA	2000
211113	BTCA	Synthesis	USA	2000
216128	Protein crosslinks	Medicinal Chemistry	USA	3000
243737	Filtration of a Formulation	Formulation	USA	3000
245802	Properities of CMC	Formulation	USA	3000
258354	Stimulus to Elicit Urination by Untrained Rats of Either Sex	Biology	USA	2000
258382	Paracrystalline Arrays	Toxicology	USA	5000
258387	Megamitochondria	Toxicology	USA	2500
260715	1-Azabicyclo [3.2.2] nonan-3-one	Synthesis	USA	70000
271659	1-Azabicyclo [3.2.1] octan-3-one	Synthesis	USA	65000
279618	Vacuum Blood Collection System	Biochemistry	USA	10000
280053	N-Boc-7-azabicyclo [2.2.1] heptene	Synthesis	USA	60000
281657	Yeast molecular genetics (1)	Molecular Biology	USA	2000
281662	Yeast molecular genetics (2)	Molecular Biology	USA	3000
286857	Alkyl phenyl alkanols	Synthesis	USA	2000
297790	Low Surface Energy Particles for Reduction of Friction	Material Science	USA	5000
301277	Efficient synthetic route	Synthesis	USA	50000
301277.1	Efficient synthetic route	Synthesis	USA	50000
349446	A-MOE	Synthesis	USA	50000
349450	G-MOE	Synthesis	USA	50000
371122	In vitro Bone Formation Assay	Biology	USA	5000
526670	Chitosan Life Sciencespolymer	Synthesis	Belgium	75000
592959	Picolinic acid (Derivative 2)	Synthesis	USA	25000
592963	Picolinic acid (Derivative 1)	Synthesis	USA	25000
609131	Yeast molecular genetics	Biochemistry	USA	5000
648798	Crosslinking Polysaccharides and Polycarboxylic acids	Synthesis	USA	50000
648808	Crosslinking Polysaccharides and Polycarboxylic acids	Synthesis	USA	3000
663856	Polymer analysis in surfactant matrices	Polymer	USA	5000
672016	Trifluoro-lactate Derivative	Synthesis	UK	7000
672025	Pyrrolo-pyrimidine	Synthesis	UK	10000
716076	Seeking Small Molecules Libraries (I)	Chemical Diversity	USA	18380
756921	Analytical Method for Active Ingredient	Analytical	USA	20000
757740	Procedure to Develop Artificial Human Fluid	Formulation	Italy	15000
776827	Stabilization of liquid formulation	Formulation	Germany	5000
795123	Purification of silicone based solvents	Technology	USA	10000
845617	Life Scienceslogical Targets for Inflammation	Biology	USA	5000
845646	Life Scienceslogical Targets for AntiLife Sciencestics	Biology	USA	5000
845675	Life Scienceslogical Targets for Obesity	Biology	USA	5000

846919	Life Scienceslogical Targets for Insulin-Releasing Compounds	Biology	Germany	30000
861088	Substituted Propionic Acid	Synthesis	USA	30000
861093	Amino Indanol	Synthesis	USA	30000
861628	Incomplete Release of Active Ingredient	Formulation	USA	5000
864721	Novel colorant materials #1	Formulation	USA	5000
865198	Novel colorant materials #2	Formulation	USA	15000
875122	Elasticity improvement in textiles	Analytical	Germany	5000
980996	Synthesis of dipalmityl- or distearyl-diketene	Synthesis	Germany	10000
991141	Synthesize hexamethylene-1	Synthesis	Germany	10000
995380	TMBA (3	Synthesis	Italy	5000
995385	Gallic Acid	Synthesis	Italy	5000
995443	DNA Extraction Method	Molecular Biology	USA	10000
997488	Burst Release Formulation	Formulation	USA	5000
997602	Plant Selectable Marker	Molecular Biology	USA	35000
997637	Plastid Selectable Marker	Molecular Biology	USA	10000
998108	Crosslinking Reaction for Polymers	Polymer	Germany	12000
1058346	Calcium carbonate nanoparticles in water	Nanotechnology	Germany	35000
1083568	Immortalized Preadipocyte Cell Line	Molecular Biology	USA	35000
1116521	Microbial strain for the production of an amino acid	Biochemistry	Germany	40000
1116609	Compounding Method	Polymer	Germany	7500
1116662	Compounds forming hydrogen-bonds	Polymer	Germany	5000
1132113	Cerium containing organic solution	Formulation	Germany	10000
1160777	New Chem and Applied Sciencesical routes to a substituted benzaldehyde	Synthesis	France	25000
1186758	additives	Material Science	Missing	50000
1201783	Diagnostic test for Interstitial Cystitis.	Analytical	USA	40000
1222113	Particle Size Reduction	Formulation	USA	55000
1223905	Controlled Encapsulation and Release of Electrolyte	Formulation	USA	30000
1223933	pH Modification	Analytical	USA	30000
1225582	Full-Length cDNA Isolation	Molecular Biology	USA	10000
1230129	DNA inverted repeat analysis	Biochemistry	USA	20000
1239328	Food-Grade Polymer	Polymer	USA	35000
1335287	Decrease of Cr (VI) concentration	Formulation	USA	15000
1351513	Bubbling Action	Formulation	Germany	10000
1351519	Formulation for a proLife Sciencestic powder	Formulation	USA	45000
1375732	Stable form of tetrasodium pyrophosphate	Formulation	USA	35000
1414246	Synthesis of an acrylic acid polymer (2)	Polymer	Germany	15000
1414254	Synthesis of an acrylic acid polymer (1)	Polymer	Germany	60000
1470313	Lowering of CO levels	Material Science	USA	35000
1470520	Substituted Benzenes	Synthesis	USA	35000
1470565	Hedonics of Oral Chem and Applied	Biochemistry	USA	55000

	Sciencesesthesis			
1480505	Synthesis of 2	Synthesis	USA	15000
1508173	Porous carbohydrate resin	Polymer	USA	40000
1594697	Gel-forming polymer	Formulation	USA	40000
1629166	Water vapor barrier glue	Polymer	Germany	35000
1650978	Lactose Polymerization	Polymer	Denmark	25000
1654057	2- Specific lipase of microbial origin	Biochemistry	Denmark	50000
1654065	Non toxic inhibitor for lipases	Medicinal Chemistry	Denmark	25000
1712465	Platelet Aggregometry Device	Technology\Knowledge Aggrigation	USA	25000
1742333	Additive to alter surface properties	Analytical	USA	40000
1747283	Enzyme Stabilizer	Formulation	USA	65000
1748963	Flash point elevation	Formulation	USA	45000
1749231	Ethanol absorbents	Formulation	USA	30000
1820210	Retort stable form of Vitamin C	Formulation	USA	15000
1841332	Water Absorbent Material	Polymer	USA	50000
1862965	Method for peptide bond synthesis	Synthesis	USA	15000
1877388	Supplier for MgO	Formulation	USA	5000
1894778	High-throughput format for a Life Scienceslogical assay	Biology	USA	50000
1900564	Selective removal of a protecting group	Analytical	Germany	12000
1908792	Method for Addition of a Salt	Technology/Chem Eng	USA	25000
1913185	Non-fluorinated oil and water repellent	Formulation	Missing	45000
1927291	Improving Solution Appearance with Novel Dyes	Formulation	USA	30000
1931590	Visual Modification of an aqueous dispersion	Formulation	USA	20000
1949133	Life Sciencesfilm Indicator	Analytical	USA	25000
1963197	Water Vapor Permeability	Analytical	USA	30000
1980008	Synthesis of 3-difluoromethyl-1-methyl-4-pyrazole carboxylic acid	Synthesis	USA	10000
2014338	Preservative Degradation	Formulation	USA	10000
2051972	Alternate material to cyclododecane	Material Science	UK	10000
2052063	Iminium ion synthesis from tertiary amines	Synthesis	USA	10000
2052165	New Phase Change Materials	Analytical	Germany	10000
2160537	Separation of tolualdehyde-acid adducts	Synthesis	USA	15000
2160626	New applications for silane-functionalized polyolefins	Polymer	USA	15000
2171115	Photo and Chem and Applied Sciencesical Passivation of Titanium Dioxide Nanopart	Analytical	USA	10000
2241934	Thiophene formation	Synthesis	USA	10000
2242777	Metals removal from heavy petroleum fractions	Analytical	USA	10000
2257439	Film-forming polymer	Polymer	USA	45000

2265185	Gametogenesis Inhibitor	Analytical	Canada	100000
2284488	Efficient synthesis of a Resorcinol Derivative	Synthesis	USA	15000
2318298	Method to avoid skin sensitization	Biochemistry	USA	15000
2374481	Seeking anti-nitration additive	Analytical	UK	15000
2417892	Seeking ion channel enzyme inhibitors	Biochemistry	USA	100000
2421769	Reduce viscosity of a salt formulation	Analytical	USA	4000
2456223	Chlorine Removal	Analytical	UK	15000
2487239	Seeking formulation development partners from China and India	Formulation	USA	5000
2489565	Detection of DNA sequences	Biochemistry	USA	10000
2489751	Analytical Assay for Phytate	Biochemistry	USA	30000
2496322	Inositol phosphate derivatization	Biochemistry	USA	30000
2566610	Insect mutant line	Toxicology	USA	30000

Chapter 3 - The Primacy Of The Periphery In A Distributed Problem Solving Community

3.1: Introduction

How does complex product development occur in voluntary and virtual online communities? The sustained ability of Free/Open Source software (F/OSS) communities to produce, maintain, and distribute, for free, complex software products raises two interesting questions; 1) Why people participate in such endeavors? ***and;*** 2) How are they able to innovate and problem solve in a virtual and distributed setting?

Considerable scholarly attention has been paid to the question of incentives and motivations of participants in F/OSS communities. Lerner and Tirole (2002) first asked the question: "Why would thousands of top-notch programmers contribute freely to the provision of a public good?" Subsequent empirical research has shown that contributors to F/OSS projects participate to satisfy both intrinsic and extrinsic motivations. Participants contribute to projects because they have a direct need and use for the software (Hertel, Niedner and Herrmann 2003; Lakhani and Wolf 2005; von Hippel 2001), they enjoy the process of software creation (Lakhani and Wolf 2005), and/or the norms of sharing and openness are paramount with their primary identification with the hacker community(Ghosh et al. 2002).

Significantly less attention has been paid to the question of how the effort of individuals is harnessed for distributed problem solving in these communities (exceptions include Lee & Cole (2003) and von Krogh, Spaeth & Lakhani (2003)). This is surprising given studies of new product development in general and software development in particular have identified effective problem solving as one of the most enduring challenges faced by managers responsible for delivering products to their various stakeholders (Brown and Eisenhardt 1997; Clark and Fujimoto 1991; Cusumano 1992; Kraut and Streeter 1995; MacCormack, Verganti and Iansiti 2001; Powell, Piccoli and Ives 2004). And in a more general sense, a central concern for organization theory has been how collective action is achieved in complex organizations (Galbraith 1973;

Lawrence and Lorsch 1967; Perrow 1986; Simon 1976; Stinchcombe 1959; Thompson 1967; Weber 1947).

The focus of this chapter is on explaining how distributed and voluntary F/OSS communities solve technology problems using inputs from hundreds, if not thousands of participants. The emerging empirical literature on F/OSS communities indicates that a majority of code writing and communication activity is concentrated with a few individuals, the "core" (Gallivan 2001), yet these communities allow and encourage wide scale participation by anybody in their community, the "periphery." My research aim is two fold:

1. To explain the division of labor amongst the core and periphery in a distributed problem solving community and in essence to determine the value of the periphery to the core;
2. To develop a grounded theory-based explanation of how the core and periphery accomplish technical problem solving and to develop a theoretical perspective on the social practices used by the community to achieve collective action.

The data from my study are based on a one year "historical" and "virtual" ethnography of the community that produces the open source PostgreSQL database management system. I had access to the majority of technical discussions that occurred in the community along with all of the changes in the software code from November 2002 to December 2003[23]. This allowed me to create a unique analytical tool, the innovation process history, that showed the detailed micro-interactions behind the creation of a complex software product. I used this analytical tool to drive my analysis of the value of the periphery to the community and an practice-based explanation of the practices that the community members use to get work done.

[23] This time line corresponded with one major software release cycle in the community – between versions 7.3 and 7.4. Practicaly speaking I had access to technical discussions and source code changes from 1997 – forward. I chose the November 2002 to December 2003 time frame as it conicided with my research window.

I begin by first defining the term "community" as it applies to my empirical context (Section 2). I then examine the literature on "periphery" and its impact on innovative outcomes (Section 3). In Section 4, I address the elements that go into creating a robust theory of distributed social action and practice. In Section 5, I provide details on the research setting, data and methods used in my analysis. Section 6 is dedicated to a quantitative assessment of the value of the periphery in a distributed problem solving setting. Then in Section 7, I present three detailed vignettes about social action and collective problem solving. I then use the data from vignettes in Section 8 to develop a practice-based theory of collective problem solving, Section 9 discusses at length the quantitative and qualitative findings. I conclude this chapter by discussing the implication of my research for the literature on innovation and product development and organizations (Section 10).

3.2: Defining Community

The term "community" has been used as both an aspiration for belonging and doing good with others and an analytical sociological variable (Calhoun 1980). In this section I discuss the traditional concept of community and its expansion to include instrumentally oriented communities. I then consider the application of this definition to organizational communities as represented by communities of practice and then offer a definition of problem solving communities as exemplified by open source communities.

3.2.1: Traditional Communities

The German sociologist, Ferdinand Tönnies' ([1887] 1957) theoretical essay, *Gemeinschaft and Gesselschaft*, first made the distinction between community and society. Tönnies argued that the community (Gemeinschaft) represented the childhood of humanity and society (Gesselschaft) its maturity (Brint 2001). Tönnies viewed community, ideally represented by a family or village, as grounded in common ways of life with concentrated ties and frequent interactions with small numbers of people who were familiar with each other and had long standing emotional bonds. In contrast, society, as represented by a city, is one where commerce dominates life of people and where rules and rationality and exchange relations dominated a person's interactions with

other individuals. Tönnies viewed individuals in the city as being distant from one and other and needing external rules and regulations to enforce order and proper behavior.

Tönnies represented the concept of community as an ideal form of living where communities offered participants a "sense of familiarity and safety, mutual concern and support, continuous loyalties, even the possibility of being appreciated for one's full personality and contribution to group life rather than for narrower aspects of rank and achievement" (Brint 2001: 2). Tönnies' conceptualization of community has not been without its critics. Empirical studies have shown the presence of inequity in community relations, power stratification and the discovery of self-interested community leaders who provide a façade of spontaneously generated consensus (Brint 2001). Tönnies' approach either invited rebukers of the concept or overly acritical, romantic descriptions of community resulting in a lack of rigorous analytical approaches to the concept (Calhoun 1980) and an inability to explain how collective action or coordination is achieved in communities (Gläser 2001).

Emile Durkheim pioneered another conceptual view of community. He recognized the importance of community for some types of human action but he stepped away from a sentimental approach to one based on variable properties of human interaction found as much in small villages as in large cities (Brint 2001). Durkheim's emphasis was on extracting an element or process associated with communal relations and its impact on behavior. Thus in Suicide (Durkheim 1966), he argued that dense and absorbing ties were an antidote to the danger of egoism and the resulting deviant social behavior. Similarly he noted that joint ritual experience was the basis of a common definition of the sacred amongst a community of believers (Durkheim [1911] 1965). Durkheim's disaggregating approach to collective human behavior has yielded narrowly defined variables from the community concept. Steven Brint proposes that sociologists who have followed Durkheim's methods have derived six properties of community relations that are important as separate variables for sociological analyses (Brint 2001: 3-4). The first four are structural variables including: 1)dense and demanding ties; 2) social attachments to and involvements in institutions; 3) ritual occasions; and 4) small group

size. The last two are cultural variables: 5) perceptions of similarity with others (physical, expressive style, way of life, historical experience); and 6) common beliefs in an idea system, a moral order, an institution or a group. Brint argues that since all six of these properties are rarely found in all communities, it makes good analytical sense to focus on them individually instead of comparing the specific community contexts within which they are found.

Brint judges the community studies tradition in sociology to be a failure (Brint 2001). Citing Hillery (1955), he notes that as early as the 1950s there were 94 separate definitions of the word community. He complains that the field has been stuck at the descriptive level and has primarily gained attention by either supporting or debunking the traditional concept of Gemeinschaft. He further argues that community studies have disappeared from contemporary sociology because of a failure to develop generalizations about human social behavior. In an attempt to rescue the concept, Brint (2001: 8) attempts to develop a new generic definition of communities as *"aggregates of people who share common activities and/or beliefs and who are bound together principally by relations of affect, loyalty, common values, and/or personal concern (i.e., interest in the personalities and life events of one another)."*

Although Brint acknowledges that material and social interests (Bourdieu [1972] 1977) can be the basis for some interactions in a community, he argues that the primary basis for relations in a community are affect, loyalty, shared values or personal involvement in the lives of others. Brint's emphasis on the affectual-relationship basis for community leads him to discount the existence of communities within work settings and voluntary interest organizations (2001: 9). He acknowledges that these entities may display some of the qualities associated with communities, however, he disqualifies them from the definition because of their instrumental and rational interest orientations.

3.2.2: Instrumentally Oriented Communities

In contrast, Jochen Gläser (2001) aims to develop a definition of community that explicitly includes an instrumental orientation. His motivation is the disconnect between

traditional definitions of community (including Brint) and the long held sociological view that scientists belong to a worldwide collectivity (Merton [1942] 1973) also known as a "scientific community." Merton conjectured that the collectivity was governed by specific scientific ethos consisting of the norms of communalism, universalism, disinterestedness and organized skepticism. Thomas Kuhn (1970) challenged the emphasis on norms as a governance mechanism by proposing the role of scientific paradigms as regulating the collective behavior of scientists thus implying that knowledge instead of values held together the scientific community (Gläser 2001).

More radically, ethnomethodolgists in the sociology of scientific knowledge have denied the existence of scientific communities and explicitly excluded them from their sociological analyses (Knorr-Cetina 1982). Gläser however persists in showing that the empirical work of the sociologists of science and scientific knowledge tends to support the notion that scientists work within a wider collective community. He shows that most citation analyses of scientific work have shown that scientists are actively "accepting, adding, rejecting, and changing the knowledge claims of their colleagues" (2001: 4). He also shows that both the early and more recent works of some of the leading scholars in the SSK tradition can be interpreted as showing a collective, community-based knowledge production. For example, Knorr Cetina's 10 year ethnography of CERN includes distinctions like "post-traditional communitarian structures," "the erasure of the individual as an epistemic subject," and the presence of "confidence pathways and gossip circles" (Knorr Cetina 1999).

Gläser then develops a parsimonious definition of community as follows (2001: 6): "A community is an actor constellation that consists of individuals who perceive to have something in common with others, and whose actions and interactions are at least partially influenced by this perception." An actor constellation is simply a collectivity of actors whose interests and action potentials overlap (Scharpf 1997); meaning that there is some intersection between individuals, their idiosyncratic interests and their action as related to those interests. Gläser contends that in order to understand and compare and contrast various communities one needs to consider the following analytical categories:

1) basis for relationship amongst participants; 2) rules for how membership is established; 3) ways in which coordination of action is achieved; and 4) institutions that support the collective.

In the case of the scientific community, Gläser states that the basis for relationship among participants is the collective knowledge production amongst the scientists. Membership is determined by a scientist's own perception of belonging to the community since most attempts at determining scientific community membership have been strongly biased by the sociometric measures used (Woolgar 1976). This also implies that membership in a scientific community "is on a continuum with a few active core members and many less active, less visible and even invisible members" (Gläser 2001: 6). The primary coordination mechanism is the actual collective work itself. Scientists engage in decentralized mutual adjustment (e.g., via citations) based on the common subject matter of the work and through the collective objects that they may produce, for example, particle accelerators (Knorr Cetina 1999). The institutions that support the scientific community are formal ones like peer review and informal ones like methodological and technical rules for experimentation, disclosure of results and citations of previous work. Ultimately, Gläser labels scientific communities as *producing communities*, because their purpose is to produce knowledge.

3.2.3: Communities of Practice

An emerging stream of research within organization theory has shown the significance of communities inside of organizations as a locus of non-canonical organizational learning and innovation (Brown and Duguid 1991; Brown and Duguid 2001) . The concept of communities of practice originated with Lave and Wenger's findings that suggested alternatives to formal school-based learning theories (Lave and Wenger 1991). They were intrigued by the historical practice of apprenticeship and wanted to develop a theory of learning that took into the account the trajectory of the journey of a new apprentice to a master, from the margins of a community to its center. Their theory was grounded in empirical examples of tailors in Liberia, Mayan midwives, US supermarket meat cutters, US navy quartermasters and non-drinking alcoholics. For

them learning is a social practice which occurs in the everyday activity of a community: "learning is not something about digestion of facts, rather it along with thinking and knowing are constituted in relations among people who are engaged in activity in the socially and culturally constructed world" (Lave and Wenger 1991: 51). A community of practice is one such locale for learning and participating in a particular occupation (e.g., being a butcher) or set of interdependent work activities (e.g., navigating a ship).

Central to their perspective on learning is legitimate peripheral participation (LPP) of newcomers in an already established community of practice. Learning occurs through LPP in a community. Newcomers in a community start by participating in a set of practices of the community and it is this participation that makes them legitimate in the eyes of the community. Initially, practices may be minor or peripheral to the community, but over time the novices take on more and more complex practices enabling them to move from the periphery of the community to its core. Once here, they become masters. Key to LPP and the journey to full membership is the ability of newcomers to gain access to the practices of the community and to observe how full members perform their practices. Transparency of the workings of the community becomes a key analytical variable in studies of communities of practice.

Wenger (1998) states that a community of practice is not equivalent to or synonymous with group, team or network. Thus membership in a formal or imagined social group does not constitute membership in a community of practice. The network of social ties also does not yield a community of practice, and neither does geographical proximity (Wenger 1998: 74). Instead, Wenger defines a community of practice on the basis of the following three dimensions: 1) mutual engagement; 2) joint enterprise; and 3) shared repertoire (Wenger 1998: 73). The dimensions themselves are not independent, instead they constitute each other and form the basis of the community. Mutual engagement implies that individuals in a community are continuously engaged in actions whose meanings they negotiate with one and other. The mutual engagement is around a joint enterprise that entails not just accomplishing the canonical work but also identity and meaning as well as belonging. Pursuing a joint enterprise together also builds a

shared repertoire amongst the community members. Such a repertoire includes "routines, words, tool, ways of doing things, stories, gestures, symbols, genres, actions, or concepts that the community has produced or adopted in the course of its existence and which have become part of its practice" (Wenger 1998: 83)

Finally Wenger takes a non-romantic and pragmatic view of communities of practice. He does not fall into the Tönniesian trap of equating community with morally good actions. Instead he states that communities of practice are neither intrinsically beneficial or harmful and can yield both positive and negative effects for both participants and the surrounding environment (Wenger 1998: 85).

It is important to note that the empirical basis for the communities of practice literature has emerged from studies of relatively stable occupations and professions, e.g., butchers, tailors and insurance adjusters. This does not mean that there is a lack of novelty or challenge in their work. For example, while most childbirth situations are routine, there maybe emergency instances where situations require creative problem solving by midwives. Similarly a navy ship entering an unfamiliar harbor may require novel and improvised problem solving by its quartermasters. However, the central imagery is one of reproductive practices enabling a newcomer to become part of an existing community and its practice (Osterlund and Carlile 2003).

The concept of communities of practice inside of organizations was introduced by Brown and Duguid's (Brown and Duguid 1991) use of Orr's (1996) ethnography of Xerox repair technicians and the basic theoretical framework as outlined by Lave and Wenger. The empirical setting is one where Xerox repair people create and share narratives about problematic photocopiers and customers. The primary problem solving approach is one of narration where repair people integrate various facts about the problem via conversation and storytelling. The stories connect the individual and collective memories of the repair people along with various tests and machine diagnostics so that a potential diagnosis and repair can be made. Through ongoing conversations these stories become repositories of a collective (and local) knowledge about what it

takes to repair machines. The stories are about machines, people and techniques for common problems. The stories are generated by and linked to the shared practice of fixing the machines. They are also highly situated and improvised and constantly evolving as the actors and machines change. The distinguishing feature of this community of practice is its eschewing of the canonical knowledge represented by Xerox repair manuals and headquarters support staff and the use of non-canonical narration and storytelling as a way to learn about repairing machines. This context shifts the focus of the community of practice from one of a newcomer attaining full membership to that of finding solutions to technical problems while also reinforcing identity and shared meaning. Practices are now reproductive (stories allow for the assimilation of new technicians) and also productive (stories allow for novel problem solving).

3.2.4: Distributed Problem Solving Communities

Free and open source software (F/OSS) communities represent a type of a community structure with an explicit orientation to innovate, i.e., continuously create (and support) complex software. This form of community organizing is unique for four reasons: 1) a mission to create technological products; 2) organization of work and workers that is loosely coupled and staffed by distributed volunteers in a virtual setting; 3) an intellectual property regime that protects rights of anyone to distribute the outputs of the community, instead of excluding non-members; and 4) the creation of "free" software products that directly "compete" with products produced by highly paid development teams inside of firms (Feller et al. 2005; Lakhani and von Hippel 2003; Weber 2004).

The mission of F/OSS communities to create software-based products put them on the instrumental end of the instrumental – affect spectrum of community orientation. F/OSS communities are deemed successful if they can create and ship software products (Weber 2004). The empirical literature on product development has identified creative problem solving as a critical component of successful innovation and new product development efforts (Allen 1977; Brown and Eisenhardt 1995; Clark and Fujimoto 1991;

von Hippel 2005). Thus a primary objective of participants in F/OSS communities is to solve ongoing and specific technical problems as they relate to their software offering and the use of that software in production environments.

The other distinguishing organizational feature of F/OSS communities is the distributed, voluntary and virtual nature of individual participation. Several empirical studies of various F/OSS communities studies have highlighted the presence of between hundreds and sometimes thousands of participants in the software development and support process (Hertel, Niedner and Herrmann 2003; Koch and Schneider 2002; Lakhani and von Hippel 2003; Lee and Cole 2003; Markus, Mannvile and Agres 2000; Mockus, Fielding and Herbsleb 2002; von Krogh, Spaeth and Lakhani 2003). Moreover participation does not occur in a centralized and physical location, instead public email lists and source code repositories on the Internet enable distributed participation by community members from around the world. Thus F/OSS communities can be classified and generalized as distributed problem solving communities.

Applying Gläser's analytical categories for community identification helps in locating and comparing distributed problem solving communities with other community types. Table 1 (adapted and modified from Gläser (2001: 7)) shows the relationship between the various communities discussed so far. In his analysis, Glaser puts F/OSS communities in the same category as scientific communities and labels them as producing communities. I make the distinction that any specific F/OSS community is very different from any specific scientific community because of the need to jointly produce working technological artifacts. As Kuhn argues, scientific communities work within paradigms, but within those paradigms there is no need for ex-ante cooperation and problem solving between community members. The knowledge transfer and joint work occurs after publication when individual scientists or laboratories have solved particular problems. In the case of F/OSS, as I will show later in the vignettes section, the need to produce a jointly developed artifact results in ex-ante information sharing and collective problem solving activity.

The basis for relationships between actors within distributed problem solving communities is the jointly produced artifact, software in the case of F/OSS. The need to use the software is the most frequently reported motivation for joining F/OSS communities with very few participants acknowledging the presence of pre-existing ties with other community members (Lakhani and Wolf 2005; Lakhani and von Hippel 2003). Membership is established by participating in the community's e-mail discussion lists and by contributing in many large and small ways (software code writing, testing, quality assurance, documentation, website development etc) to the product development effort. There is no settled agreement in the literature on the basis for coordination of action in distributed problem solving communities. Weber, using a political economy lens on F/OSS, has argued that a combination of individual incentives, cultural norms and leadership practices enable distributed development (Weber 2004: 159). An alternative conceptualization would be to consider coordination by mutual adjustment as information about the ongoing development and use of the software artifact is made available to community members. The institutional framework for distributed producing communities consists of the sharing of intellectual property which enable free use and modification by anyone and the norms and convention around credit and attribution (Raymond 1999).

Table 3.1: Subtypes of Communities[24]

	Subtypes of Communities				
	Problem solving communities	Knowledge Producing communities	Communities of Practice	Social Movements	"Traditional" Communities
Relation to other actors	Jointly produced artifact	Common subject matter of work	Common activity and occupation/profession	Common goal	Common norms and values
Basis of membership	Joint participation in artifact creation or support of others	Perception of having something in common with others	Shared practices and legitimate peripheral participation	Perception of having something in common with others	Perception of having something in common with others and location
Coordination	Incentives, norms, leadership | Mutual adjustment via changes to artifact and use of artifact	Common subject matter of work	Partially by institutions	Institutions and ad-hoc organizations	Norms and values
Institutions	Sharing of intellectual property, credit and attribution	Standards and procedural rules for interactions	Standards for individual conduct of common activity, procedural rules for interactions	Rules defining situations that require action, rules coordinating collective action	Dependent on what values and norms are shared
Examples	Free/Open Source Software Communities	Scientific communities	Xerox service technicians community	Environmental and peace movements	Religious communities

24 Modified and adapted from Gläser (2001:7).

3.3: The Periphery and its Value to the Community

The Tönniesian inspired *Gemeinschaft* tradition has the potential to lead many to assume that communities are egalitarian, unstratified and open to all members. However a stable empirical finding in the traditional communities literature has shown the emergence of status hierarchies and other forms of stratification (Brint 2001: 15). The reasons for stratification can include: differing levels of contribution, differential skills, knowledge and abilities related to community activities; social or physical attributes; and as a means of enforcing social control.

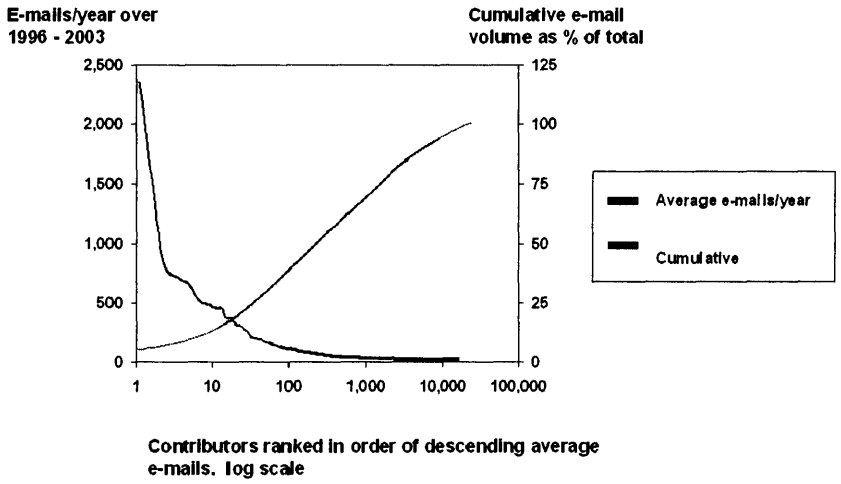
Similar stable findings about stratification have been found in instrumentally oriented communities in general and distributed problem solving communities in particular. Participation, as defined by contributing code to the F/OSS community, has been found to be highly skewed and to follow a power distribution, where majority of the code writing effort is done by a very small minority of contributors (Koch and Schneider 2002; Lee and Cole 2003; Mockus, Fielding and Herbsleb 2002; von Krogh, Spaeth and Lakhani 2003). An excerpt from a recent speech by Andrew Morton[25], one of the core developers in Linux, starkly highlights this stratification of participation in code writing:

> "If we look at the Linux kernel project: 38,000 Changes in three years. Approx 1,000 individuals. But top 20 contributors provided about half of the changes. Heavily skewed contribution rate."

Similarly skewed distribution rates are also observed in the communication patterns, between 1996-2003, of the Linux Kernel email list, the main location for development activity. Figure 3.1 shows that approximately 75% of all emails (approximately 350,000 messages) were sent by just 11% of individual authors between 1996-2003.

[25] Available from http://www.tech-forum.org/upcoming/transcripts/Transcript_OpenSource_07-15-04.pdf

Figure 3.1 – Skewed Participation in Linux Kernel Email Discussion



These empirical findings raise several questions about the value of the peripheral community members in a disturbed problem solving community. If we consider just code writing in Linux, we need to determine what is the contribution and value of the remaining 980 participants? At any one time there are approximately 3000 individuals who are subscribed to the Linux kernel email list. Thus we need to consider the value of the additional 2000 members who have not contributed code. Even more starkly when we consider a broader time horizon of participation (1996-2003) on the email list we find that approximately 20,000 individuals have participated in discussing some aspect of the Linux kernel. What is the value of the these 20,000 individuals to the extremely small number of code writers in the community? An economically rational view of development and participation may argue that much of the periphery is generating noise in the system and that the top 20-100 developers should instead create a private discussion list to get the work done and ignore the largely "non-contributing" members.

In this section I discuss how the lower strata of the community, the periphery, has been defined and the potential role it plays in achieving community objectives. I first discuss the different conceptualizations of the core and periphery and how they relate to distributed problem solving communities. I then propose an analytical definition of the

core and periphery for distributed problem solving communities, and use that to inform my subsequent analyses. Finally I discuss the relevant research questions in this study.

3.3.1: Defining the Periphery

Core/periphery interaction structures have been identified in a number of social science fields. The imagery is prevalent in fields as diverse as political geography (Gottman 1980), economics (Krugman 1996), sociology (Borgatti and Everett 1999), and organization studies (Faulkner 1987). Although there is no settled definition of core and periphery amongst the various fields (Borgatti and Everett 1999), the two most common ways to distinguish core and periphery are a statistical approach based on the density of ties amongst participants (Borgatti and Everett 1999), and an approach that examines authority and knowledge structures in a community and assigns individuals with authority to the core and those without to the periphery (e.g. Lave and Wenger (1991)).

The field of social network analysis has pioneered statistical approaches to determining core-periphery structures in human communication networks. The basic approach is to gather data on ties[26] between individuals, often based on some measure of communication patterns, and then to use matrix algebra techniques to determine cliques, cores and periphery. The basic intuition is that there are two classes of nodes in a network, a cohesive set of nodes where actors are "densely" connected to each other, the core, and set of nodes with actors more loosely connected to the cohesive core and each other, the periphery (Borgatti and Everett 1999).

Lave and Wenger's (1991) concept of legitimate peripheral participation in a community of practice informs us of the authority/knowledge approach to determining core and periphery community structure. Peripheral members in a community of practice lack both specific knowledge about the community's practices and are also in subordinate authority positions, that is apprentices working with and for masters. The point of LPP is to enable peripheral newcomers to become skillful in the practices of the community by legitimate participation in those practices over time. The acquisition of skill and fuller

[26] Ties can also be based on citations in scientific articles or patents.

participation in the community also provides access to differential power positions in the community and thus the journey of a periphery member via LPP is also empowering (Lave & Wenger 1991: 36).

However Lave and Wenger (1991: 36) also emphasize that: "There is no place in a community of practice designated as "the periphery," and most emphatically, it has no single core or center." The metaphorical point that they try to emphasize is that peripheral participation leads to *full participation* and not necessarily central or complete participation. Lave and Wenger's stance of a denial of "the periphery" and "the core" in a community of practice is understandable but puzzling. They may have fallen into the trap of Gesselschaft trap of only positive characterization of community. They do however later acknowledge and recognize the presence of power struggles, conflicts and the denial of LPP in communities of practice. Thus I would claim that a core and periphery do exist in communities of practice.

Lave and Wenger's distinctions do help us to think about how to define core and periphery in other communities. One approach would be to consider the relevant knowledge base of the practitioners in a community. Those in the core could be considered more knowledgeable about the community-specific knowledge structures and practices and have greater ability to demonstrate that to periphery members. Hence by default periphery members may have less of the relevant community-specific knowledge. This does not mean that peripheral members lack in general knowledge or are not knowledgeable about other matters. Indeed as Granovetter (1973) has shown, if we conceive of the periphery as being weakly tied to the core of the community, weak ties in a network, can be a source of important non-redundant information to the community.

A related but analytically separate approach would be to consider formal and informal authority in a community setting. Those with authority can be considered core and those without authority, the periphery. There are two type of authority-related actions to consider. The first one is the master-apprentice relationship or in traditional organizations, supervisor-subordinate relationship. In this type of authority,

superordinate individuals can compel action by subordinate individuals on the basis of their status in the "organizational" hierarchy. In the case of communities of practice, the apprentices are under the authority of full members and learn by following their instructions and demonstrations. Another form of authority is having formal decision rights in a community. The ability to set policy and guide the general direction of the community. Core members have decision rights on important community-specific dimensions and periphery members do not have such rights. In this case the authority is more indirect. While a core community member will not able to directly compel a peripheral member to do something, there may be instances where their wielding of their decision rights can change the trajectory of actions from peripheral participants.

3.3.2 The Periphery as the Locus of Innovation and Source of Novelty

The community of practice lens views peripheral participants as trying to learn the practices of the community and to eventually become full and skillful members of that community. Peripheral members are learning and seeking knowledge about the specific ways in which the community acts and performs in the world from the existing full members. However if, the mission of the community is not just to reproduce existing practices but create new ones, that is, to innovate and to produce new knowledge and artifacts (as in the case of producing communities and problem solving communities), then the interactions between core and periphery members need to be viewed in a different light. It may be that peripheral members bring forth experiences and knowledge that can be recombined in interaction with each other and core members so that new forms of knowledge and innovation can result. Hence, the role of the periphery is not just to learn about the practices of the community but to fully participate in novel problem solving and to be a potential source of innovation in a community.

One stream of research in the sociology of science has argued that the flow of ideas and innovation in scientific communities is centripetal instead of centrifugal (Chubin 1976), that is, the margins of the scientific community are the drivers of change and progress. Thus Crane (1969: 349) in her study of the "invisible college" in the natural sciences speculated that "outsiders" were a likely source of new ideas and innovation:

"Most problem areas are open to influence from other fields. The desire for originality motivates scientists to maintain contacts with scientists and scientific work in areas different from their own in order to enhance their ability to develop new ideas in their own areas."

A review of six scientific disciplines, (radio astronomy, bacteriology, psychology, phage group, physical chemistry, x-ray protein crystallography) by Edge and Mulkay (1974) (cited by Chubin 1976) showed that innovations from the margins and the mobility of scientists across fields were the only consistent factors in scientific innovation and specialty development across these fields. Edge and Mulkay did express concern that very little was known about the social process underpinning their findings: "If we are correct in suspecting that many major scientific innovations come from the outside, or from the margins of, established research communities (either from applied research contexts, or by migration between research networks), then it is surprising that so little is known about this process" (Edge and Mulkay (1974) cited in Chubin (1976: 457)).

Research on the sources of technological innovation has repeatedly highlighted that novel innovations arise when problem solving activity is decentralized (von Hippel 1988; von Hippel 2005). Users have been shown to innovate in a variety of consumer, industrial and scientific settings (von Hippel 2005), often preceding and initiating firm-based efforts (von Hippel 1978; von Hippel 1982; von Hippel 1988; von Hippel 1989). Here users are on the periphery and the firms that commercialize and sell products which have innovations are the core. Thus in the field of scientific instruments, Riggs and von Hippel (1994) found that 44% (n=64) of the innovations emerged from users dispersed in industry, universities and government laboratories, while the remaining 56% emerged from a handful of manufacturers. They further found that the vast majority of functionally novel innovations, that is, enabling new technical capability in the equipment, were developed by dispersed users and "dimension of merit" improvements, i.e. convenience or reliability, were developed by manufacturers. More recently, DeMonaco, Ali & von Hippel (2005) have shown that in pharmaceuticals industry, 76% (n=29) of the new drugs introduced in 1998 had significant "off-label", that is, novel uses

not in the original drug approval process, applications. They found that 59% (85/144) of the "off-label" drug therapy innovations were discovered by distributed and peripheral practicing clinicians via field discovery as compared to the scientists working inside of the pharmaceutical companies.

Within sociology, Weiman's (1982) study of the flow of information and influence in the personal network of an Israeli kibbutz community also shows "the importance of marginality" or peripheral participation. Weiman gathered sociometric data from 270 members of the kibbutz regarding conversational ties with other members of the community yielding 2511 conversation ties. Weiman then used matrix algebra to determine cliques in the community and then derived a network position of each individual based on the number of times a person was chosen as a conversational tie by someone else. "Centrals" and "Marginals" were then determined by using the upper and lower quartiles of the choice distribution in a clique. As expected centrals, dominated in all types of communication patterns. In addition centrals, were more efficient in the flow of information. Information originating from centrals flowed more faster, was deemed more accurate and more credible than the information activated by the marginals. However, marginals were key for inter-group or inter-clique communication. Marginals were both receivers and transmitters of information amongst the 16 distinct groups in the kibbutz. Weiman showed that marginals were the importers of new information across groups and that centrals then served as the transmitters of that information within groups. Implying that centrals rely on marginals for imported information while the marginals required the enlistment of centrals for spreading the information in the group.

There are two theoretical perspectives underpinning the findings related to the importance of the periphery. Granovetter's (1973) seminal article on the strength of weak ties posits that weak ties amongst individuals allow for the transfer of non-redundant and novel information amongst colleagues as opposed to strong ties amongst friends. Strong ties imply that the information flow amongst strongly connected individuals will be homogenous and already known, while weak ties may enable the transfer of new and heterogeneous information. Thus those on the periphery of a

community are more likely to be weakly tied to the core, while they may serve as "bridges" between other communities and thus transfer novel information amongst them. This theoretical perspective is the basis for both Weiman's empirical findings about the importance of marginality and Chubin's assertion regarding the centripetal flow of novel information in science communities. Although not mentioned by Granovetter, Hayek's (1945) central insight about the unequal and distributed nature of knowledge in society explains why non-redundant information may exist in the first place. If knowledge is both spatially and intellectually distributed – then gaining access to this knowledge via weak ties may be one mechanism by which peripheral members provide advantages to communities.

The other theoretical perspective on the value of the periphery arises from von Hippel's findings about the critical role of users in the innovation process. Here the theoretical perspective is the relative stickiness of information. Von Hippel argues that the locus of innovation shifts to where the information is the most stickiest (von Hippel 1994a; von Hippel 1994b; von Hippel 1999). Thus users innovate in areas where they have needs not met by manufacturers, typically in using technologies in novel ways, while manufacturers innovate in areas where they have pre-existing expertise, typically manufacturing the technology or improving it on the dimensions of merit instead of novelty. It is not just a matter of the presence of non-redundant information. Rather users or peripheral members in problem solving communities experience novel issues not foreseen by manufacturers or core members and in many cases the transfer of this use experience is very difficult and expensive, if not impossible. A strong tie between a core developer and a peripheral user does not mean the core will now have the information needed to innovate. Rather the use environment will dictate that such innovation has to be primarily driven by the periphery. Thus the periphery has to first innovate and then transfer the newly created knowledge to the core, regardless of the strength of ties.

In the case of distributed problem solving communities we can anticipate that both theoretical perspectives are applicable depending on the sophistication and use characteristics of both core and peripheral members.

3.3.3 Defining core and periphery in F/OSS communities

The emerging empirical studies on various F/OSS projects have identified the presence of multiple roles amongst the participants in those communities. These roles include: 1) project leaders who assume overall "responsibility" for the output of the community; 2) developers who contribute code; and 3) other individuals who while using the software identify bugs, submit problem reports and test the code (Gallivan 2001). Statistical analysis of contribution patterns in terms of code writing and e-mail communication show a small set of participants who contribute a lot and a much wider set of individuals with significantly less participation for both coding and emailing (Koch and Schneider 2002; Lee and Cole 2003; Mockus, Fielding and Herbsleb 2002; von Krogh, Spaeth and Lakhani 2003). This has led many researchers to make a distinction in participation between "core developers" and "peripheral developers" (Gallivan 2001: 293).

However this core – periphery distinction as currently used in the literature has conflated statistical findings with decision rights in the community. For example, many of the studies use historical logs from the community's public source code repository to track contribution rates by developers in the community. Contributions are summed by individual and then rank-ordered to determine the underlying distribution and patterns of participation. Statistical tests, such as cluster analyses are then used to determine who is core and who is peripheral e.g., Koch et al. 2002. However, this procedure ignores the fact that all the individuals listed as contributors in the logs of the source code repository have full decision rights to commit code to the project repository. Regardless of their contribution rates and importance, all individuals in the project can commit code *any where* in the repository. Mockus et al. 2002 describe the process in the open source Apache web server community: "*Each AG* [Apache Group] *member is expected to use his/her judgment about committing code to the base, but there is no rule prohibiting any AG member from committing code to any part of the server. Votes are generally reserved for major changes that would affect other developers who are adding or changing functionality.*" Thus once given the decision right to commit code to the repository, participants have a lot of latitude and freedom to make changes. In addition this decision

right also allows them to commit code from individuals who do not have these rights. So a high frequency contributor may simply be playing a functional bridging role between the core and the periphery by committing code from others. So while we observe a statistical distinction between high and low frequency contributors to the code writing effort, from a community point of view, all of them may be seen to have achieved core status.

Different communities have different rules about how someone gets the right to make official changes to the source code repository. Von Krogh et al. 2003 have identified the presence of a joining script whose repeated enactment by newcomers in a community helps predict their trajectory towards the core group. In some cases, the community is fairly laissez faire about who gets source code access, for example in GNOME, a highly sophisticated Linux graphical user interface desktop product, there were 280 individuals (German 2005) who had full access to the repository in 2002. Here decisions rights are given as soon as a new individual is submits acceptable code changes at a high rate (i.e. he/she is continuously asking others for his/her "good" code to be added to the repository). While in other cases, there may be a formal mentoring and voting system amongst the current core members as to who gets "promoted" and given full access to the repository (Mockus, Fielding and Herbsleb 2002). In either situation, the key element is gaining the decision right to commit code to the repository.

Generalizing to other distributed problem solving communities, the core can be considered as having the ability to make permanent changes to the joint work output of the community. The core may serve other functions as well, for example media liaison, public relations, distribution coordination etc. However, the ability to make changes to the focal technological object of the community is a status that can be used to differentiate core and peripheral participation. In my particular empirical context, I therefore identify core members as those who have full access to make changes to the source code repository, and peripheral as those who do not.

3.4: A Practice View of Distributed Problem Solving Communities

A majority of the academic research on problem solving has focused on the individual as he/she goes about solving a problem, and most of that research has involved the use of discrete "puzzles" in artificial domains instead of complex real-world situations (Dunbar 1998). Two important streams in the individual/cognitive problem solving domains are the Gestalt-based tradition on important role of insight in problem solving and the Simonian view of problem solving as a search in a problem space.

The Gestalt tradition, as best represented by the work of Duncker (1945), argued that solutions occurred when the individual gained "insight" into the problem and not through a process of trial and error learning (Dunbar 1998). Followers of the Gestalt tradition referred to four stages of problem solving: preparation, incubation, insight and verification. Sudden insight enabled problem solvers to discover a crucial part of the problem and once that was discovered, all other parts fell into place automatically and the problem was solved.

Duncker (1945) and colleagues (Adamson 1952; Birch and Rabinowitz 1951) identified the existence of "functional fixedness" - where problem solvers have difficulty using familiar tools in novel ways as a major source of insight blockage. In one of Duncker's experiments, he created five problems which could only be solved by applying a new way of using a tool. The first of the two groups of experimental subject saw the tool being used in a usual way while the second group did not. The result of the exercise was that subjects were more likely to solve the problems requiring a novel way of using the tool if they had not observed how that tool was used in the usual way. The problem solving success of subjects that had previously observed the tool being used as usual was hampered.

In Duncker's terms the subjects were "fixated" on the tools' normal function and could not re-conceptualize it in a way that permitted them to solve the problem. The way around functional fixedness and the generation of new insight is for the individual solver

to work on restructuring the problem so that it is amenable to insight generation. Restructuring of problems could be facilitated via hints to the problem solver or by the problem solver rethinking the constraints of the problem description and solving it in non-obvious ways. In either case it was the individual and his/her own cognitive ability that determined success in problem solving.

The seminal work of Newell and Simon (Newell and Simon 1972; Simon and Newell 1962) has described the problem solving process as an attempt to get from the present to a desired situation through a process of "searching through a large maze." The maze depicts the problem space, the nodes of the problem space represent situations, and the paths joining one node to another are the actions that will transform one situation into another. A problem space has an initial state and a goal state and a set of means that allows a solver to move from one state to another. Problem solving is an act of stepwise search through the problem space (Dunbar, 1998) and decisions by problem solvers are taken under significant uncertainty (Simon, 1969: 68).

Newell and Simon (1972) characterize the differences in problem solving search processes as "trial and error," "hill-climbing," and "means-end analysis." The three approaches form a succession in which the latter processes requires information the former does not. Trial and error problem requires solvers to only recognize that they have reached their goal; hill climbing requires that solvers can assess the relative closeness of their position to the goal; and means-end analysis requires that the solver be able to discern the type of difference between the current state and the goal state (Baron 1988).

Understanding how core and peripheral participants, many of them strangers in a distributed community, collectively solve problems and create complex technological artifacts puts the emphasis on the social aspects of the problem solving process. In particular, collective problem solving implies a relational view of the problem solving process, that is, how do the various participants relate to each other and what are the specific individual and collective activities that they undertake so that so that they can move forward and develop solutions? While the individual approaches may be relevant

and applicable in a community setting, there is also a need to identify the collective mechanisms that enable communities to innovate and solve problems.

A relational view of problem solving implies the application and development of a theory of social practice. A theory of social practice emphasizes the relational interdependence of individuals with each other, their ongoing activity and their interactions with technology and tools as they jointly solve problems (Lave and Wenger 1991: 50). A social practice theory goes further than other theories by focusing on interactions and relations and it places everyday activity as the locus of production and reproduction of relations (Osterlund and Carlile 2003: 3). Thus a theory about problem solving in a collective setting is not about a sudden burst of insight or an individual search in a problem space, rather it is the articulation and explication of the activities taken on by the collective in their problem solving process. The focus is not on the attributes of individuals or groups, rather it is on the relations between people and objects and how those relations come about. In other words, if we believe systems theorists that "the whole is greater than the sum of the parts" (Langlois 1983), then understanding the linkages between parts becomes important.

As is to be expected of an interpretive theory building tradition, "there is no unified practice approach" (Schatzki 2001: 2). However a key tenet of the diverse approaches to practice theory identifies practices as "embodied, materially mediated arrays of human activity centrally organized around shared practical understanding" (Schatzki 2001: 2). Central to this perspective is the primary focus on human activity, that is, the concrete actions taken by people. Practices depend on activity, and activity amongst and between individuals relies on the ongoing development of shared understandings and agreements. Finally activities are often mediated through material means, meaning, individuals utilize material objects and relate to material objects to perform their activities and to develop shared understanding. Practices are thus dependent on and are constitutive of the material world.

Barnes hence concurs with the above view of practice theory by alluding to Kuhn's proposition about paradigms in the natural sciences (Barnes 2001). Citing Kuhn he asserts that one can view paradigms as practices and not "theories": "accepted examples of actual scientific practice – examples include law, theory, application and instrumentation together; they are examples selected as model achievements, ways of solving problems known to work in one case and available to guide practice in other cases" (Kuhn 1970: 10). Hence paradigms represent shared activities constituting practice and in any "normal" science field they are an indication of agreement about practices. However, this agreement can be about joint activities alone and thus it is "perfectly possible for them to press forward cooperatively on the basis of this agreement, whilst being in radical disagreement with each other at the level of "philosophy" or in their abstract theoretical ideas" (Barnes 2001: 20). Barnes' supposition has been empirically supported by Gallison and colleagues (Galison 1999; Galison and Stump 1996) who have shown that the sciences in many fields are "united in their disunities," even while producing practical and "proven" outcomes.

Developing a social practice theory grounded in the actual activities of the actors can provided an analytic lens on how social order and task accomplishment occurs. Practice theory has been used to study identity formation (Wenger 1998), communally centered distributed knowledge (Brown and Duguid 1991; Brown and Duguid 2001; Lave and Wenger 1991; Orr 1996) and distributed organizing (Orlikowski 2002). Developing a practice theory implies developing a general and abstract account of a particular empirical phenomenon (Schatzki 2001: 3). Schatzki notes that: "Systems of generalizations (or universal statements) that back explanations, predictions, and research strategies are theories. But so, too, for example, are typologies of social phenomena; models of social affairs; accounts of what social things (e.g., practices, institutions) are; conceptual frameworks for depicting sociability; and descriptions of social life – so long as they are couched in general, abstract terms" (2001: 4). Hence one of my aims here is to develop a general and abstract account of distributed problem solving communities.

3.5: Research Setting, Data and Methods

3.5. 1 Research Setting

The PostgreSQL community produces the open source PostgreSQL database management software (PG from now on). Databases are a critical component of the modern information infrastructure as they serve as the repositories of much of the data produced and consumed in the information economy and society. The PG software and community have a had lengthy evolution and a history going back to the middle of 1970s [27]. In the mid 1970s' Michael Stonebraker at University of California, Berkeley started an academic research project to create a "modern" DBMS called Ingres. In 1982, Stonebraker left Berkeley to commercialize Ingres, but eventually returned to academia in 1985. Upon his return, Stonebraker started a "post-Ingres" project to address the problems with contemporary database systems that had become increasingly clear during his commercial experience in the 1980s. Hence the term "Postgres."

Starting in 1986, Stonebreaker's academic team released a number of computer science papers (see Stonebraker, Rowe, & Hirohama (1990) for citations) describing the basis of the system, and by 1988 the project had a prototype version up and running. That year, Stonebraker released the prototype under a the Berkeley Software Distribution (BSD) license, which enabled free copying and modifications by anyone. Over the next five year's Stonebraker's team released three version of PG. By 1993 a large number (over 1000) of users existed and began to overwhelm the project with requests for support and features. After releasing a Version 4 of the software, primarily to cleanup existing issues, the project ended.

Although the Postgres project had officially ended, the BSD license (under which Berkeley had released Postgres) enabled Open Source developers to obtain copies and to develop the system further. In 1994, two UC Berkeley graduate students, Andrew Yu and Jolly Chen, added a SQL language interpreter to replace the earlier non-standard system, creating Postgres95. The increasing user demand for improvements to the system resulted

[27] I obtained details on the history from this wikipedia entry : http://en.wikipedia.org/wiki/Postgresql

in a decision, in 1996, by Yu and Chen to create an external email list and to decentralize the software development effort.

Bruce Momjian, one of the earliest core developers in the community describes the transition from an academic project to a distributed community[28]:

> In the summer of 1996, it became clear that the demand for an open source SQL database server was great, and a team was formed to continue development. Marc G. Fournier, in Toronto, offered to host the mailing list and provide a server to host the source tree. One thousand mailing list subscribers were moved to the new list. A server was configured, giving a few people login accounts to apply patches to the source code using cvs.
>
> By this point Jolly Chen had stated, "This project needs a few people with lots of time, not many people with a little time." With 250,000 lines of C code, it was easy to understand what he meant. In the early days, there were four people heavily involved: Marc Fournier in Canada, Thomas Lockhart in Pasadena, California, Vadim Mikheev in Krasnoyarsk, Russia, and me in Philadelphia, Pennsylvania. We all had full-time jobs, so we did this in our spare time. Calling this a challenge was an understatement.
>
> Our first goal was to scour the old mailing list, evaluating patches that had been posted to fix various problems. The system was quite fragile then and not easily understood. During the first six months of development, there was fear that a single patch would break the system, and we would be unable to correct the problem. Many bug reports had us scratching our heads, trying to figure out not only what was wrong, but how the system even performed many functions.
>
> We inherited a huge installed base. A typical bug report was, "When I do this, it crashes the database." We had a whole list of them. It became clear that some organization was needed. Most bug reports required significant research to fix, and many were duplicates, so our TODO list reported every buggy SQL

[28] http://www.oreillynet.com/pub/a/network/2000/06/16/magazine/postgresql_history.html

query. It helped us identify our bugs, and made users aware of them too, cutting down on duplicate bug reports.

We had many eager developers, but the learning curve in understanding how the back-end worked was significant. Many developers got involved at the edges of the source code, like language interfaces or database tools, where things were easier to understand. Other developers focused on specific problem queries, trying to locate the source of the bug. It was amazing to see that many bugs were fixed with just one line of C code. Postgres had evolved in an academic environment and had not been exposed to the full spectrum of real-world queries. During that period, there was talk of adding features, but the instability of the system made bug fixing our major focus."

Table 2 shows the growing complexity of the PG software, in terms of lines of the source code written since it started to operate in a distributed problem solving community. Over the seven years of releases the code-base for the project has approximately tripled in size and 3,393 individuals have participated in the development email list for the community.

Table 3.2: Release History of PostgreSQL Software

Release Date	Release Version	Lines of Code (LOC) in "C"	Days Between Release Cycles	LOC / Release Cycle	LOC/Day per Release Cycle
10/27/1996	1.09	178,976	n.a.	n.a.	n.a.
6/8/1997	6.1	200,709	224	21,733	97
10/2/1997	6.2	225,848	116	25,139	217
3/1/1998	6.3	260,809	150	34,961	233
10/30/1998	6.4	297,918	243	37,109	153
6/9/1999	6.5	331,278	222	33,360	150
5/8/2000	7	383,270	334	51,992	156
4/13/2001	7.1	410,500	340	27,230	80
2/4/2002	7.2	394,274	297	-16,226[29]	n.a.
11/27/2002	7.3	453,282	296	59,008	199
11/17/2003	7.4	508,523	355	55,241	156

PG development occurs outside the boundaries of any "formal" organization[30]. The community is virtual in both the literal and physical sense. There is no formal organization driving the development of the software and there is no physical space where the developers work together. PG by itself does not have any employees. However, many developers do have jobs that allow them to work on PG software – full time and part-time. The community congregates at three major Internet-based rendezvous points:

1) The PG website (http://www.postgresql.org) is a portal into the community with areas for developers, users, and downloading source code;
2) There are 22 e-mail lists related to the various aspects of the project. The majority of the technology development activity occurs on the "hackers" and

[29] In this PG release the community worked to rearchitect the system and not to remove redundanices in the software code and thus we observe a reduction in the linese of code from the previous release.
[30] Some open source projects have incorporated as non-profit organizations (for legal protection purposes) and often have a formal board of directors.

"patches" e-mail list. All of the lists are open except for one which is reserved for five members of the PG steering committee for their community-related administrative activities (described below);

3) Public source code repository, also referred to as CVS (Concurrent Versioning System). Anyone in the world has read access to CVS, that is, they can download all or some of the files. At the time of my study 11 individuals had decision rights for "write" access to this repository, that is they had the ability to commit changes to the software and make these official for the rest of the community.

The "steering committee" is historical and self-appointed and consists of very active developers. At time of my study the steering committee included the following five individuals: Peter Eisentraut, Aachen, Germany; Marc G. Fournier, Wolfville, Nova Scotia, Canada; Tom Lane, Pittsburgh, Pennsylvania, USA; Bruce Momjian, Philadelphia, Pennsylvania, USA; and Jan Wieck, Germany. Interviews with the steering committee members (Momjian, Wieck) revealed three primary functions: 1) to determine the release schedule for software, that is, to declare dates when the software is in beta testing and when the software is ready for release; 2) to discuss people issues, that is how to handle unruly individuals in the community (rare) and who to give commit access to in the community (more common); and 3) to represent the public face of the community to media and commercial entities looking to establish relationships with the community. The steering committee members also emphatically noted that none of the technical problem solving and design discussions happen on their private list. All members of the steering community have full time jobs with firms that have products or services which relate to the PG software.

3.5.2 Data and Methods

My research objectives are in the spirit of "bringing work back in" (Barley and Kunda 2001) for the development of grounded theories of distributed problem solving. I chose to study one complete technology development cycle as the PG community's software release went from version 7.3 to 7.4. This entailed collecting data on one year of technology development activity in the community from November 27, 2002 (the 7.3

release date) to November 17, 2003 (the 7.4 release date). During this time period the PG community added 55, 241 new lines of code to its collective product and created 241 new features and improvements over the 7.3 code base. As table 2 shows, this amount of code production is consistent with the previous nine community-based releases. F/OSS communities are relatively modest in advertising their technological achievements and the PG community is no exception. Thus the point upgrade from 7.3 to 7.4 should not be mistaken for an incremental improvement in technology. In a commercial setting equivalent achievements of code writing and new feature development would merit an increase of the software version number to an "8.0." Following the entire technology development cycle allowed me to observe a majority of the situations and conditions that the community faced during the various phases of the software development process: design, development, testing, beta release, and user feedback.

I used multiple data collection methods to achieve theoretical triangulation and insight into the workings of the community. Data collection began after the 7.4 release was completed. Two unique features of F/OSS communities in general and the PG community in particular enabled me to "directly observe" the interactional and problem solving aspects of community activity: 1) all of the technology development occurs via public e-mail lists and software source code repositories; and the 2) majority of the technically-related communications and interactions, and related source code changes are publicly archived. Thus in the 7.4 release cycle there were 20,129 email messages exchanged between 798 individuals in two development-related emails lists and 2,402 changes committed by 11 individuals to the source code repository. The publicly archived discussions and changes to the source code repository allowed me to reconstruct the entire development history of the community as they went from the 7.3 to 7.4 version and in essence enabled me to conduct detailed historical analyses and to use ethnographic methods on the interaction history of the community.

I analyzed the problem solving process in the PG community by creating an analytic tool called an "innovation process history." In the innovation process history, I matched concrete software features (for example spell checking in a word processor) to

actual changes in the software source code repository that enabled the features, and to related public interactions (on e-mail lists) in the community during that feature's development.

The innovation process history for PG was created in the following manner. Once a development cycle is completed, the PG community creates a "release note" for the user population that lists all the new features and improvements in the newly released version of the software. Each item in the release note provides details on the new feature or change in the software along with an acknowledgement of the individuals associated with that change. The release notes for PG 7.4 listed 241 new features and changes. In essence, the release note provides an indication of all the relevant technological problems that are solved by that change. The features and changes in the release note are enabled via the changes made to the software code repository, CVS. The software repository is a versioning source code management system that keeps track of all code changes along with following meta-information: 1) date and time of change; 2) identity of the person making the commit; 3) names of existing files changed and lines of code added and subtracted per file; 4) new files added; and 5) any comments made by the person making the commit, including acknowledgement of the person(s) who wrote the code if it not the committer.

I worked with a periphery member, Neil Conway, who was an active participant and coder during the 7.4 release cycle, to match the 241 software feature in the 7.4 release note to the 2402 changes in the software repository. This resulted in the direct matching of the 574 source code changes to the 241 software features. An analysis of the remaining 1828 residual changes in the source code revealed that they consisted of documentation, translation, and technical features that were still under development and not complete. I choose not to include these residuals in my analysis as they were not directly related to the technical solving problem effort of completed features.

The matching of the features to the source code changes provided me with an approximate timeline of development along with key words and names of individuals

associated with each feature. I was then able to search the email archives of the two development related email lists to track all the related technical discussions underpinning the software changes and ultimately the features. This then enabled me to track the problem solving process enacted by the participants as they discussed, developed and jointly converged on a solution. The innovation process history thus provided me with quantitative and qualitative data on the core-periphery interaction and problem solving process used in the community. For each feature developed, I was able to determine who initiated the development process, who provided confirmatory evidence for similar needs, who participated in the problem solving process, the number and sources of the potential solutions proposed, what was the configuration and history of the actual solution developed, what feedback was provided, and what were the overall timing and community dynamics.

I also worked with Bruce Momjian, a long-standing core developer in the community, to categorize the types of features and changes developed in the 7.4 release cycle. One of his tasks in the PG community is to create the release notes for new PG community software releases. Each release note is developed collaboratively by the community with Momjian leading the email discussion. Thus he has good ability and a broad view of all the changes in any particular release note. Lientz and Swanson (1980) in their study of 487 information technology organizations have identified three reasons for making changes to software programs: adaptive, corrective and perfective. Adaptive changes add new features to the software program. They alter the software to meet newly changing and discovered use requirements. Corrective changes are made to fix problems with existing features; that is, bug fixing. A bug is defined as a variance between the expected behavior of the system and its actual behavior in test or use conditions (Crowston 1997). Finally perfective changes are those related to improving existing performance of the system. In this change the problem is not unexpected behavior rather a desire for improved performance of the same behavior (e.g.: making software run faster). Momjian exclusively assigned each of the 241 items in the release notes to one of the Lientz and Swanson categories. For reasons of parsimony and comparability I then transformed the Lientz and Swanson software change categories to an innovation

taxonomy developed by Riggs and von Hippel (1994). Their taxonomy consists of classifying innovations as either being ***functionally novel***, i.e. enabling new technical capability, or a ***dimension of merit improvement*** i.e. performance improvement or fixing problems. Thus adaptive changes were reclassified as functionally novel and corrective and perfective changes were reclassified as dimension of merit improvements.

As I was generating the innovation process histories of all 241 new features[31], I paid paying close attention to the ongoing interactions between community participants and how they were engaging in joint problem solving activity. As is to be expected, a detailed exploration into the one year history of development in a community of 798 individuals will yield information on other relevant issues beyond just my narrow focus on problem solving. Thus I found the community grappling with issues related to assigning credit to contributors, ways to speed up and streamline the software development process, competitive information and how to deal with commercial pressures, explaining to novices how the development process worked and why it was so different from commercial firms, etc. All of these were interesting and could be fruitful sources of future reflection and research. However, I chose to remain focused on exploring the practices of collective core-periphery problem solving.

Once the innovation process histories were completed, I then developed 32 detailed vignettes of the problem solving process enacted in the PG community. Half the vignettes related to functionally novel changes and the other half related to dimension of merit improvements. Depending on the scale, scope and length of the change, the vignettes took up between 5 to 40 pages of notes each, with an average of 15 pages. The vignettes exhaustively detailed the community discussion underlying the feature development and included my interpretations of the problem solving process. I consulted with Conway and Momjian when I encountered interactions amongst individuals which were not clear and areas that I found to be confusing. Furthermore, since I had email contact information on all the participants, when I found areas in which I needed more clarification or context I would reach out to the appropriate person and have an email

[31] The appendix contains a list of all the 241 changes and their respective innovation category.

exchange with him/her on the particular topic. Most of the time this related to understanding better their expertise, how they came to be developing solutions and their reflections on experiences in the PG community. In all, I conducted 25 e-mail based interviews to further enhance the vignettes.

I used inductive qualitative techniques to analyze the vignettes (Eisenhardt 1989; Eisenhardt 1991; Glaser and Strauss 1967; Strauss and Corbin 1990) with a focus on discerning the activities that enabled distributed problem solving. My analysis was geared towards understanding the commonality and differences in activities that were being performed by the participants in the vignettes. I used Orlikowski's (2002: 256) definition of activity; "what members did every day as part of the complex and distributed product development work;" to guide my open coding of the activities involved in collective problem solving. Initially I attempted to use the individual problem solving literature and the traditional software development literature as a guide to assist me in creating categories and themes. However, I found that to be quite constraining and not reflective of the actual work being done in the community. I thus abandoned those categories and decided to let the categories emerge via multiple readings of the vignettes, combined with examination of the innovation process histories and related development email discussions, and reflections on my interview notes with the various participants.

I used Orlikowski's (2002) distinction that practices are both individual (performed by actors in their everyday action) and institutional (they shape and are shaped by collective norms and structures) to focus on activities that an individual did and the activities that occurred at a collective level. Individual level activities as those that were done by participants on their Of course in the empirical setting these are intertwined in the sense that actors have full agency over their activities, however there are a set of activities that occur in response and in light of the collective and others that reflect individual action. Using the individual and collective practice distinction, I then grouped and combined activities into practices and then created appropriate labels for those activities and practices.

Once my initial set of activities and practices were developed. I individually consulted with Ben Hyde and Stefano Mazzocchi from the open source Apache Community to get their reactions and feedback on my categorization. I chose them as collaborators in fine tuning the practices because they have extensive experience in building and leading open source communities (Hyde was president of the web server project within Apache, which has over 60% global market share, Mazzocchi has started and led four different and successful open source communities and is currently on the Board of the Apache Software Foundation). Their experience in a different but related empirical setting helped me to develop categorizations that were robust across distributed problem solving communities. Hyde and Mazzocchi provided feedback that made me revisit my categories and to re-examine the data from the vignettes in a new light. Their input led to a further refinement of the practices and activities and helped me to reframe the number of practices developed from ten to six. I then presented the activities and practices to my PG informants, five other members of the PG community and a leader from one other open source project, GNOME, for their reactions and feedback. Their commentary helped to refine my interpretations and representations of the distributed problem solving practices and helped in creating suitable generalizations across communities.

3.6: The Primacy of the Periphery

In this section I quantitatively analyze the participation patterns of core and peripheral community members on the development email list ("hackers") and their contribution to the new software features developed between the 7.3 and 7.4 release cycle. I used the innovation process histories, where I matched new features in the 7.4 release note to source code changes and email discussion, to generate the data underlying the core – periphery contribution the software development process.

3.6.1: Community participation structure

Table 3.3 shows that during the 7.4 release cycle there were 792 individuals who participated in the PG community's "hackers" email list. Eleven of these participants had CVS commit access, that is, the decision right to make changes to the community's software repository, and as discussed above were considered core community members.

Table 3.4 shows that core members dominate the email message traffic on a per person basis. While 781 peripheral members were responsible for 61% (16,797) of the emails generated, two core members, Tom Lane and Bruce Momjian generated 30% of the email list traffic. On average, core members were generating between 196 times (for Lane and Momjian) and 13 times (remaining core members) the email traffic from the peripheral participants.

Email discussion take the form of "threads," where messages are "threaded" together around a common topic. McDaniel, Olson and Magee offer a useful definition (1996: 41) "We define a thread as 'a stream of conversation in which successive contributions continue a topic, following an initial contribution which introduces a new topic.'" Yates, Orlikowski and Woerner (2003) have shown threads are used to establish and maintain continuity, coherence, and coordination in the collaborative work of a virtual team. Table 3.5 shows that on an overall basis, peripheral participants were responsible for initiating 76% of the discussion threads on the development email list. On a per person basis the core members were still responsible for initiating 23 times the number of discussions as compared to peripheral participants. However, it is worth considering that a large majority of the discussions were being triggered by peripheral members. Starting a discussion implies giving direction and perhaps setting the intellectual agenda for a conversation. Thus, in this context the distributed peripheral members help to lead the agenda setting activity for the community. In addition, given that on a per-person basis the number of discussions imitated is fairly low, three threads/member, the implied diversity of the discussions initiated can be expected to be quite high. Many different individuals initiate conversations in the community and this provides new sources of information into the development activity.

Table 3.3: Community Structure during the PG 7.4 Release Cycle (November 2002 - November 2003)

Community Role	Number of Participants
Core	11
Periphery	781
Total	792

Table 3.4 - Core Members Dominate Email Message Traffic

Source of Email Message	Number of Messages	% of Messages	Messages / Person
Core Group 1 (Lane & Momjian)	5098	30%	2549
Core Group 2 (Remaining 9 members)	1473	9%	164
Periphery (781 Members)	10226	61%	13
Total	16797	100	

Table 3.5 - Periphery Drives Community Discussion

Community Role	Number of Threads Initiated	% of Threads Initiated
Core	790	24%
Periphery	2542	76%
Total	3332	100%

3.6.2: Contribution to software development and problem solving process

Table 3.6 shows the breakdown by innovation type of the 241 features developed by the PG community and the attribution to the core and peripheral participant in the release note. Recall that Bruce Momjian, one of the core developers, categorized these features according to the software change taxonomy developed by Lientz and Swanson (1980) as adaptive, corrective and perfective. I then took those ratings and transformed them into the Riggs and von Hippel innovation categorization as functionally novel (adaptive) and dimension of merit improvement (corrective and perfective). [32] Consistent with findings from Lientz and Swanson (1980), in their analysis of improvements to existing software program, a majority of the work done in a mature software project like PG relate to corrective and perfective changes. Note again that the 11 core developers appear to do a majority of the work and have been credited for 55% of the software

[32] The appendix to the paper contains a listing of all the features and their respective categorization.

coding effort, whereas 74 peripheral members were credited with developing the remaining 45% of the features.

Table 3.6 also reveals that when considering the innovation categorization of the features that were developed, as a percentage core members specialize in creating dimension of merit improvements (62% of dimension of merit features) and peripheral participants dominate in creating functionally novel changes (62% of functionally novel features). Thus members from the distributed periphery are a significant source of novelty and new functionality into the community. This finding is consistent with Riggs and von Hippel's findings that distributed users are the source of functionally novel improvements in the chemistry analyzer industry. The primacy of the periphery in creating a majority of the functionally novel innovations is most likely related to them being embedded in heterogeneous use and technical environments. Peripheral community members encounter needs and problems that are not experienced by the core developers and thus they innovate to solve their own local problems or issues.

Table 3.6: Feature Type Author by Member Status [33]

	Member Status		
Innovation Categorization	Core	Periphery	Total
Functionally Novel (Adaptive)	27 (38%)	44 (62%)	71
Dimension of Merit (Corrective and Perfective)	105 (62%)	65 (38%)	170
Total	132 (55%)	109 (45%)	241

[33] Author here refers to the person given credit on the PG release note.

Table 3.7 considers the source of the original initiation of the new feature by member status. I developed this analysis by tracking down the source of the first email that triggered work on a particular feature. In this case we see that the peripheral members, regardless of the final coding attribution in the release, are the largest source of new feature development in the community with 70% of all features being triggered by peripheral participants. Peripheral members also lead in triggering both functionally novel and dimension of merit changes. These results corroborate the data from table 4 where discussion initiation is widely distributed in the peripheral membership. Hence the peripheral membership, besides being credited as the majority authors of the functionally novel code, are also key in triggering the majority of coding activity.

Table 3.7: Source of Initiation of Feature by Community Member Status

	Member Status		
Innovation Categorization	Core	Periphery	Total
Functionally Novel (Adaptive)	16 (22%)	66 (71%)	71
Dimension of Merit (Corrective and Perfective)	56 (34%)	113 (66%)	170
Total	72 (30%)	179 (70%)	241

Besides writing novel code and triggering coding activity, peripheral members also play an important role in the problem solving process that underlies any particular feature. In creating the innovation process history I examined the role of the periphery in:

- triggering the feature development;
- in confirming the need information;
- in helping to formulate the exact problem;
- in proposing potential solution information; and
- in providing information that was used in the final solution

Table 3.8 shows that peripheral members play an equally important role in the entire problem solving process. As noted above 70% of the changes are triggered by peripheral participants. In 59% of the cases peripheral participants confirmed that they had the need

for the feature and in 58% of the cases the peripheral participants worked with core and other peripheral participants to help formulate or diagnose the exact problem. In 51% of the cases the peripheral participants provided potential solution information for the problem at hand and in 41% of the cases – the solution information was utilized in the final software code.

Table 3.8: Role of Periphery in Community Problem Solving

Actions Taken by the Peripheral Participants	% of Changes
Triggered feature development	70
Confirmed need information	59
Helped formulate exact problem	58
Proposed solution information	51
Provided information used in final solution	42

N = 241

In my innovation process history I also tracked the timing of the first software code submission for each feature. Table 3.9 shows that 39% of the 109 features developed by the peripheral members arrived "pre-made" to the community, that is, the peripheral developer submitted code at the time of the first problem statement. This indicates that a significant portion of the peripheral developers first solve problems locally and then participate in the larger community. This does not mean that the software does not evolve or change once it arrives at the community. However, the arrival of the code, does signal to the rest of the community that a peripheral participant is motivated enough to make a serious attempt at problem resolution.

Table 3.9: Code Developed by Periphery Arrives "Pre-Made"

	% of Features developed by Periphery
Code submitted at time of problem statement	39
Code submitted after problem statement	61

N = 109

3.7: Vignettes of Distributed Core-Periphery Problem Solving

In this section I present three vignettes of core-periphery interactions and collective problem solving effort in creating a complex technological product. The vignettes are based on analyzing the development process in three distinct phases: 1) problem definition and solution exploration; 2) solution development; and 3) use and refinement. These phases are my categorization of the development process and are for analytic convenience. In reality, the boundaries of the phases are quite blurry and community participants may not, while in the midst of their work, think of these categories as being salient.

I chose the vignettes to illustrate representative problem solving and feature development episodes in the PG community. The vignettes show the complete feature development history from the time the feature need was first mentioned on the development list to its entry into the release note. As discussed above, the vignettes were developed via the innovation process history analysis of PG's 7.4 release note. Two of the vignettes illustrate novel feature development and one illustrates a major dimension of merit improvement to the software.[34] The first vignette illustrates peripheral community members dominating the problem solving and code writing effort for a new feature. The second vignette shows core members leading the problem solving and code writing related to a dimension of merit improvement (getting better performance for an existing feature). The third vignette shows significant collaboration between core and peripheral members as they work towards creating new functionality.

3,7.1: Vignette 1 – Periphery Members Develop Novel Feature

The development of the Auto Vacuum (AV) feature in the PG community demonstrates how the periphery in a community can take the lead in all aspects of the software development process. Vacuuming is a routine maintenance function required in databases so that the system will continue to maintain high performance levels. Databases provide access to stored data and the ability to update the stored data with new data. During an update transaction, the PG system creates new entries for the data in the

[34] The innovation characterization of the vignettes was based on Bruce Momjian's coding discussed in Section 5.

transaction and then makes them available once the transaction is completed. Meanwhile, parallel access queries for the same data see the old data for consistency of results. When the transaction is completed the database now contains both the new updated data and the old data – with the old data being inaccessible to users. However this old data takes up memory and hard disk space and can lead to performance degradations if it is not removed via vacuuming.

There were two significant vacuuming issues with previous PG releases. First, vacuuming imposed a performance penalty on the database while it was in use. Database administrators had to carefully time the vacuum function as the entire system would slow down when vacuuming was initiated. In addition, the database administrator had to manually determine the areas of the system that needed vacuuming as the operation took a significant amount of time and would lock out other operations while in process. Second, many new users did not know that vacuuming was required and would not initiate the function, and would later complain about performance issues. As one experienced PG user mentioned[35]:

> "We at Company[XYZ] often hear that "postgressql is very slow, and the files are getting larger" and then "wow! it's so much faster now that we're regularly vacuuming!" after we let them know about this need (the RPM install of PostgresSQL is so easy that most people don't read any docs)."

Auto Vacuum itself was not on the project's TODO list or anticipated before-hand by the core team. PostgreSQL users voted AV functionality as most "favorite new feature" in the 7.4 release cycle. The development of this feature involved participation by 23 community members over a one year period of time. The feature was initiated, developed and refined by three peripheral participants with active engagement by core and other peripheral developers.

[35] In the email discussion around the creation of the AV feature.

Problem Definition and Solution Exploration (September 3, 2002)

The problem definition and solution exploration phase of this feature took place over a 24 hour period and comprised eight individuals from six different countries. This phase in the development cycle illustrates the emergence of a collective problem solving process where different contributors participate by modifying and changing ideas generated by others. No one person was actively guiding the discussion or its outcome.

The need for automatic vacuuming was first articulated during the 7.3 beta testing cycle by Mario Weilguni, a peripheral community member from Germany:

> Subject: possible vacuum improvement?
> Date: 9/3/2002 2:55 AM
> I know everyone is busy with the 7.3beta, but maybe this is something to think of before releasing the beta. Currently VACUUM will vacuum every table, but sometimes it's desireable to leave tables untouched because the're mostly static or protocol tables.
> <snip>[36]
> VACUUM on the 4GB table needs a long long time and no improvements, it just hurts performance and fills OS buffers.
> If pg_class would have a field for storing misc flags (e.g. a bitfield). This would allow to set a flag like NO_AUTO_VACUUM and modify the vacuum code to leave that tables untouched if not specified by hand. <snip>

Weilguni outlined the problem he was facing with the current implementation of PG's vacuum feature and also proposed a potential work-around solution. He clearly identified his problem by detailing why and how the current implementation of PG vacuum did not satisfy his use conditions. Weilguni recognized that "everyone is busy with the 7.3 beta[37]," meaning that the likelihood of attracting attention to his problem was going to be low, however he still attempted to get other developers to mobilize around him and recognize the importance of the problem and initiate coding during the beta testing period.

---

[36] <snip> indicates that I have removed text from the original email message relating to the technical and software coding details. These details did not directly realte to my social practice analysis of the problem solving in the community.

[37] Typically new features are not submitted during the beta test period.

Shridhar Daithankar, a peripheral member from India, with an alternative solution that he had implemented for a similar problem. Daithankar's post indicated that his solution of selective vacuuming gave him a significant performance advantage and he proposed a solution that was based on what he had done in a client environment. Weilguni, responded to Daithankar's solution by stating that his solution was not practical for large databases that he encountered. Daithankar immediately responded by agreeing with the criticism but stating that there were no practical alternatives. A few hours later, Rod Taylor, a peripheral member from Toronto, responded to Weilguni's criticisms of Daithankar's solution by providing a prototype of script that would automate the vacuuming as proposed by Daithankar. Taylor did not provide a full solution. Instead he indicated that Weilguni's objections were not well founded and could be easily solved.

Tom Lane, a core member of the PG project from Pennsylvania, responded to Weilguni's initial e-mail by indicating that nothing would happen in the current (7.3) release cycle. However he thought that the problem identified by Weilguni was worth resolving and proposed an outline for how the feature might work:

> Mario Weilguni <mweilguni@sime.com> writes:
> >[38] I know everyone is busy with the 7.3beta, but maybe this is something
> > to think of before releasing the beta.
>
> We are already in feature freeze.
>
> In terms of what might happen for 7.4 or beyond, what I'd personally
> like to see is some "auto vacuum" facility that would launch background
> vacuums automatically every so often. This could (eventually) be made
> self-tuning so that it would vacuum heavily-updated tables more often
> than seldom-updated ones --- while not forgetting the
> every-billion-transactions rule...

Lane's message encapsulated the problems and possible solutions from both Weilguni and Daithankar as well as creating a name for the feature "auto vacuum." Daithankar responded to Lane's message by indicating that he would work on implementing auto

[38] In email message protocol – when you quote someone else you typicaly have their name at the top of the message and have the quoted text preceded by a ">" symbol on every quoted line.

vacuum and asked for feedback on a detailed design of his proposed solution. His proposal contained six major elements and he invited feedback on his section. Lane responded to him with a review of his design.

> "Shridhar Daithankar" <shridhar_daithankar@persistent.co.in> writes:
> > 1)Is this sounds like a workable solution?
>
> Adding a trigger to every tuple update won't do at all.
> <snip>
>
> > 4)Is use of threads sounds portable enough?
> Threads are completely out of the question, at least if you have any
> hope of seeing this code get accepted into the core distro.
> <snip>
>
> What I had in the back of my mind was: each backend counts attempted
> insertions and deletions in its relcache entries (an update adds to both
> counts).
> <snip>

Lane's review was highly detailed and specific. He indicated that Daithankar's plan to use "triggers" would not be a good solution and his plans to implement it via "threads" would not be acceptable inside the core system. Lane then proceeded to provide an alternative detailed technical design. His solution was reflective of his deep knowledge of PG internals with significant interactions across critical components of the software code. In contrast, Daithankar's design avoided the core software system.

Weilguni responded to Lane's critique of Daithankar's design by proposing yet another design to accomplish auto vacuuming. His solution was triggered by Lane's suggestion to allow for selective vacuuming:

> >I do not think we need or want a control table for this; certainly I see
> >no need for per-table manual control over this process. There should
> >probably be a few knobs in the form of GUC parameters so that the admin
> >can control how much overall work the auto-vacuumer does. For instance
> >you'd probably like to turn it off when under peak interactive load.

> If (auto)vacuum is clever to check that some tables do not need vacuum there's really no need for that. That brings me to another point, can't the statistics collector used for that?
>
> For my database I wrote a statistic display program for web-access, and all the info autovacuum would need is here.
> http://mw.sime.com/pgsql.htm

Weilguni's proposal used an existing feature of PG called "statistics collector" and he demonstrated the feasibility of his design by providing a URL to his own website that showed the core of his design idea in operation. Lane indicated that this solution was also feasible and posed additional technical issues:

> "Mario Weilguni" <mario.weilguni@icomedias.com> writes:
> > That brings me to another point, can't the statistics collector used
> >for that?
> Hmm, that would be a different way of attacking the problem. Not sure offhand which is better, but it'd surely be worth considering both.
> <snip>

Matthew O'Connor, another peripheral participant from the US, responded to Lane's message by indicating that he also planned on implementing Weilguni's design.

There was no further public activity on the discussion list on this matter. The initial phase of AV development did not come to a certain and absolute conclusion regarding the final design, expected delivery date, and eventual author of the feature. The problem definition and solution exploration phase illustrated three important traits of the community problem solving process. First, we observed that no one individual was directing the dialog over this issue. Instead individuals participated as they saw fit and responded to issues that they were interested in. Second, solutions and designs were continually being modified, challenged and replaced as additional individuals joined the discussion. In all five alternative solutions were proposed to solve the same problem and core members did not necessarily present solutions that were eventually adopted. Third, the person initiating the problem, Weilguni, did not publicly signal that he would work on solving his own problem, instead two "new" people indicated that they would attempt to

create this functionality. But it was not clear if and when they would succeed as the solution of the first volunteer, Daithankar, was ruled to be impractical by a core member and the second person, O'Connor, did not provide significant solution information. Interestingly, the potential solution providers did not have any previous ties or connections with Weilguni nor did they indicate that they had experienced a similar problem themselves.

While the development email list was quiet on this issue, activity was ocurring in the background by the various individuals who had expressed some degree of interest in the problem posted by Weilguni. As discussed below, there were three independent attempts to solve the initial problem, with one individual reporting on his initial success.

Solution Development (September 23, 2002 – March 20, 2003)

Twenty days later, on September 23, 2002, Daithankar announced to the development e-mail list that he had created a program that implemented auto vacuuming and offered it to others for use and comments. The program he wrote integrated elements of his initial design proposal and various elements of the cumulative design discussion, from other individuals. In particular, Daithankar abandoned the use of threads and triggers in is original proposal and instead embraced the use of the statistics collector as proposed by Weilguni. However, he made his solution a separate client program (daemon) instead of a full integration into the PG system, as recommended by Lane:

Subject: Postgresql Automatic vacuum
Date:9/23/2002 9:43 AM
Hello All,
I have written a small daemon that can automatically vacuum PostgreSQL database, depending upon activity per table. It sits on top of postgres statistics collector.
<snip>
The project location is
http://gborg.postgresql.org/project/pgavd/projdisplay.php
Let me know for bugs/improvements and comments..
<snip>

John Buckman, a commercial user of PG software from Washington D.C., indicated that this solution was needed and would be good for attracting new users:

> Just an FYI - this kind of thing would be a *great* feature addition to the generic PostgresSQL release. **<snip>** Automatic maintenance of database tables is a Good Thing (tm) and would make more people we introduce to pgsql favorably disposed toward it. -john

Daithankar then posted to the list a response to a private e-mail from Matthew O'Connor regarding his implementation:

> **Date: 9/24/2002 2:16 AM**
> On 23 Sep 2002 at 13:28, Matthew T. O'Connor wrote:
> > Hello Shridhar, sorry I didn't respond to the email you sent me a while
> >back. Anyway, I saw this post, and just started taking a look a it. I
> >wasn't thinking of doing this as a totally separate executable / code
> >base, but perhaps that has advantages I need to think more.
> > A couple of quick questions, you are using C++, but all postgres source code is in C, do you want this to eventually be included as part of the postges distribution? If so, I think that C might be a better choice.
>
> Well, I wrote it in C++ because I like it. I have lost habit of writing pure C code. Nothing else.
>
> As far as getting into base postgresql distro. I don't mind it rewriting but I have some reservations.
>
> 1) As it is postgresql source code is huge. Adding functions to it which directly taps into it's nervous system e.g. cache, would take far more time to perfect in all conditions.
>
> My application as it is is an external client app. It enjoys all the isolation provided by postgresql. Besides this is a low priority functionality at runtime, unlike real time replication. It would rarely matter it vacuum is triggered after 6 seconds instead of configuerd 5 seconds, for example.
>
> Less code, less bugs is my thinking.

> I wanted this functionality out fast. I didn't want to invest in learning postgresql source code because I didn't have time. So I wrote a separate app. Besides it would run on all previous postgresql versions which supports statistics collection. That's a huge plus if you ask me.
> <snip>
> I am Cc'ing this to Hackers because I am sure some people might have same doubts.

His answer to O'Connor and to the rest of the discussion list indicated that he made two unique design choices to match his abilities and interest. He programmed the system in C++ instead of the C because he had better knowledge of that programming language even though the base PG system was written in C. The program itself was designed to be completely separated from the base PG system because Daithankar did not have the time or the inclination to learn all the internal operations of the software. Both of these choices and Daithankar's explanation for them showed that he preferred a quick working solution to the initial problem over meeting the programming conventions of the community and/or following strictly the design outline made by Lane which needed significant understanding of PG internals.

Weilguni, the initiator of this need and the person who presented the idea to use the statistics collector as the basis for doing auto vacuum, responded to Daithankar's email by raising technical issues with his design. He also told the community that he had made an attempt, but had run out of time, to write something similar as well:

> Two weeks ago I began writing a similar daemon, but had no time yet to finish it.

No further public activity was observed on the email list until 11/26/2002, when O' Connor announced that he had taken Daithankar's code and implemented it in the C programming language. He indicated that he had it working on his system and also provided evidence for its performance via a graph (see figure 3.1):

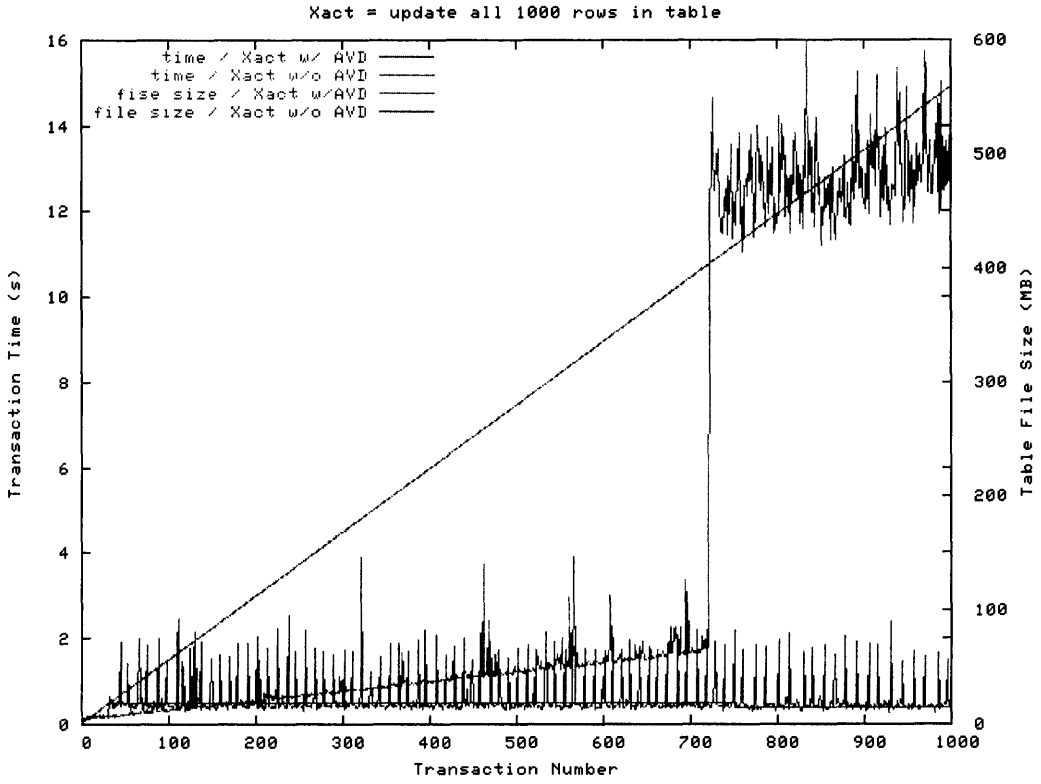
> Several months ago [I] tried to implement a special postgres backend as an Auto Vacuum Daemon (AVD), somewhat like the stats collector. I failed due to my lack of experience with the postgres source.

On Sep 23, Shridhar Daithankar released an AVD written in C++ that acted as a client program rather than part of the backend. I rewrote it in C, and have been playing with it ever since. At this point I need feedback and direction from the hacker group.

Anyway for you reading pleasure, I have attached a plot of results from a simple test program I wrote. As you can see from the plot, AVD keeps the file size under control. Also, the first few Xacts are faster in the non AVD case, but after that AVD keeps the average Xact time down. The periodic spikes in the AVD run correspond to when the AVD has fired off a vacuum.

<snip>

Figure 3.2 – Graphic of Performance Improvement Submitted by Matthew O'Connor



In response to O'Connor's posting Daithankar explained the logic of his original program and also thanked him for his work.

Between 11/28/2002 and 12/10/2002 five other people gave feedback to O'Connor on possible new functionality and improvements to the features. O'Connor continued to work on the program and then sent in an updated version of the software to the community on 2/3/2003. Bruce Momjian, one of the core developers, indicated that this was a nice feature to have and asked the community as to "where" in the source code this program belonged. Between 2/17/2003 – 2/19/2003 discussion ensued on the patches list, based on Momjian's trigger, as to the best place to put this code. Note that the modular nature of this code, it only ran as a client, gave the community flexibility in terms of how to absorb the code. Lane's comments exemplified this:

> I agree, it seems like a server-side implementation would be the only credible way to go for a production-grade version of this feature. But I don't see anything wrong with building a client-side prototype, which is what pg_avd looks like from here. (Unless the client is contorted by not being able to get at things it needs.)

A peripheral user, Magnus Neusland from Germany, entered the discussion by stating that he was very satisfied with the feature as implemented originally by Daithankar and that he had actually been using it in production systems (i.e., systems that are running databases performing real tasks) for months:

> Maybe i can provide a datapoint on this matter.
> I've been using pgavd from over at gborg (Shridhar Daithankar's (i think?) code) in production for some months now. I love it.
> I was some hazzle to get it up, but once installed everything is smooth and nice.
>
> The vacuum + analyze seems to affect my system in an only postive way:
>
> 1) Everything seems snappier.
> 2) Weekly vacuum full is faster now.
> 3) The updates on a table with counters is updated faster nowadays.
> 4) pgavd is totally automatic, so just install it and run it.
>
> Well i just wanted to say that i don't care if it's an external/client process, it makes a huge differance to us lazy sob's that always forgets to vacuum all the time.

> It's even better that it's using pgstats to determine when to run vacuum, so after big inserts it gets vacuumed promptly.

Neil Conway, a peripheral developer from Kingston, Ontario, provided a very detailed review of the code and suggested specific changes in the coding style and logic. Interestingly the feedback provided by Conway was corrected by Lane on the same day with issues of coding style and logic.

On 3/3/2002, O'Connor posted an updated version of the code based on the earlier comments from the community:

> I have updated my pg_autovacuum program (formerly pg_avd, the name changed as per discussion on the patches list).
>
> This version should be a good bit better. It addresses all the issues pointed out by Neil Conway. <snip> I have moved it from bin to contrib. More detail on the changes are in the TODO file.
> <snip>
> As always, any and all feedback is appreciated.

Members of the community revived the discussion relating to the proper location of the program within the overall PG code structure. Many in the list wanted to have a core/server-side implementation of the auto vacuuming functionality. This was not a trivial exercise and would entail a significant amount of change to the original submission. O'Connor was reluctant to commit to this course of action:

> **Date: 3/18/2003**
> Right. I am trying to work on server-side solution but can't promise to complete it by June 1.[39] If I (or someone else) gets a server-side solution done before the 7.4 feature freeze, then I think we should pull this out of /contrib. Otherwise, I find it a useful tool, I have it running in production and have received some positive feedback from others who are using it, so /contrib seems safe enough.

and Lane, as one of the core developers, was satisfied with the current situation:

[39] June 1, 2003 had been announced as the feature freeze date for the 7.4 release cycle.

```
Date: 3/18/2003
I think a server-side solution is the way to go in the long run, but
since one probably won't be available for 7.4, a contrib module seems
like a reasonable stopgap offering.  Contrib is our traditional refuge
for "not ready for prime time" code, no?  The fact that it's a client
and doesn't touch server-side code actually works in its favor here
--- there's nothing to rip out after we have a better answer
```

On 3/20/2003, Bruce Momjian, applied the changes to the source code repository so that O'Connor's program became part of the official PostgreSQL database software.

This phase of the development process shows how the community worked with donations of time, effort and code from distributed volunteer contributors. Daithankar's code relied on ideas from others on how to implement the feature and his own requirements to have the code completed in a short period of time. O'Connor's transformation of Daithankar's original submission (without asking his permission) made it more appealing and useful to the wider PG community and he bolstered his case by showing compelling graphical evidence of performance differences in production systems made using his code. Code review from Conway and further discussion on its precise implementation allowed O'Connor to modify the code and to demonstrate his flexibility in responding to feedback and being an active community member. Finally community leaders and members demonstrated a pragmatic sense of inclusion by not insisting that the code submission be modified so that it reside on the server-side instead of the current client-side.

Use and Ongoing Refinement (3/21/2003 – 9/13/2003)

The inclusion of AV functionality into the official source repository exposed it to many more users and initiated further refinement of the implementation. In particular, some users ran the new functionality on actual production systems and then reported back issues and concerns with the software. In particular two peripheral community members, Christopher Browne and Adam Kavan, initiated changes to the AV functionality based on problems they encountered while running the system. Browne

actually made changes to the software to suit his use environment and then generalized the solution to other users. He then submitted those changes privately to O'Connor, who acknowledged the contribution on the public e-mail list. Kavan reported problems on his system which upon further investigation by five other members of the community showed bugs and design flaws in O'Connor's implementation as well as previously undiscovered problems in other unrelated subsystems. Kavan's report triggered more investigation by Browne and he initiated further changes to the AV code, which were then incorporated by Momjian and O'Connor into the CVS repository.

Use and ongoing refinement became critical components of the ongoing problem solving process as it exposed the software to heterogeneous production environments that were not available to any one contributor working in isolation. In addition, the early deployment of the software on actual production systems by knowledgeable participants helped to expose bugs and problems and created new solutions that may not been detected by the developers in their own use environment.

3.7.2: Vignette 2 – Core member improves existing feature (dimension of merit change)

The development of the "Regular Expressions" update[40] involved the participation of eight community members (2 core and 6 periphery) over a one week period of time. The need for the update was identified by a peripheral community participant by reporting and providing evidence of a significant performance slowdown in the latest version of PG. One core member of the community modified software code from another F/OSS project to provide the updated requirements for the new release (7.4) of PG. Another core member solved the problem for the existing version of PG. Another five periphery community members participated in giving ideas and suggestions and also discussing the user-facing functionality for this feature.

"Regular expressions" (RegEx) are part of the essential tool kit for any advanced programmer and database analyst. Regular expressions enable the search and

[40] This update involved changes to 29 files and included adding 4448 lines of code and removing 2402 lines of code.

replacement of complex patterns in text and enable programmatic functions based on the detection of specified patterns in text. "Search and replace" functionality available in most modern word processing and spreadsheet applications is a rudimentary example of using regular expressions in everyday computing. I first discuss the origins of the RegEx package and the community's need for an updated version. I then show how the development process unfolded.

History of RegEx Module

The PG RegEx module was derived from Henry Spencer's[41] (not a member of the PG community) work on the BSD operating system. The original Berkeley release of PG, in 1996, included Spencer's RegEx module, suitably modified to fit PG requirements. Interestingly many of the current members of the PG core team did not know that the RegEx module in PG was written by Spencer. Bruce Momjian, a core developer, first discovered its origins in an e-mail exchange regarding the possibility of an alternative text searching package in early 1998:

> Subject: Re: Appended a string of text to each line in a file
> Date: 2/23/1998 2:44 PM
> <snip>
> Henry, I am CC'ing this to the PostgreSQL group. (See www.postgresql.org for more info.) Hey folks, guess who wrote our regex stuff.
>
> Copyright 1992, 1993, 1994 Henry Spencer. All rights reserved.
>
> Henry, will the new code you write be in the public domain, or only part of BSDI? Would you recommend we replace our regex stuff with something else? Do you have any patches you would like us to test?

Momjian, by copying Spencer on the message, informed him of the PG community's use of his software module and also inquired about the possibility of new faster code from him. Spencer responded to Momjian and the rest of the community, within an hour,

[41] Henry Spencer is recognized in the open source community as the key implementer of Regular Expressions in various programming applications. Further details on him are available in wikipedia: http://en.wikipedia.org/wiki/Henry_Spencer.

indicating that the new software should be available shortly and would be under a compatible intellectual property licensing arrangement so that the PG community could use it as well. He also noted that his package was based on contributions from other people:

> **Date: 2/23/1998 3:42 PM**
> > Henry, will the new code you write be in the public domain, or only
> >part of BSDI?
>
> The new regex code will be under essentially the same redistribution terms as the old stuff (in fact, slightly more generous). BSDI didn't end up contributing to this particular project, and the folks who did were all happy with open redistribution.
>
> I should clarify that this code isn't "to be written" -- it already exists, although I'm not entirely happy with it yet and want to limit distribution until it's tidied up somewhat.
> <snip>

Momjian then added the following item on the PG project's TODO list: "Get faster regex() code from Henry Spencer." There was one PG-specific modification[42] to Henry's original code in early 2002 but nobody specifically worked on the TODO item from 1998 to 2003. The lack of work on this item in the five years it was on the project's TODO list may be because none of the PG users or developers encountered significant performance problems with the current RegEx module and/or the members of the community did not have a suitable version from Henry to import into PG. The second reason is confirmed by Momjian's message to the developer list when a peripheral member requested an update on the TODO item in 2000:

> Date: 3/8/2000
> Henry has new code that is faster, and he has put it only in TCL so far. I am waiting for a library version of it that we can include.

[42] Adding locale support (local language support) to character sets.

Momjian indicated that although Spencer's new code was available, it was embedded in another program (TCL is a programming language) and thus he was waiting for a more modular version (in the form of a library) for import into PG.

Problem Definition and Solution Exploration (1/31/2003 – 2/4/2003)

Active work on updating the Regular Expressions package began when a user encountered significant slowdown in performance while using the very latest development version of PG's code. In early 2003, Wade Klaver from Kelowna, British Columbia, a relatively new peripheral community member (he had posted some bug reports earlier), posted a message to the development list:

> Subject: POSIX regex performance bug in 7.3 Vs. 7.2
> Date: 1/31/2003 5:37 PM
> Hello,
> We recently upgraded a project from 7.2 to 7.3.1 to make use of some of the cool new features in 7.3. The installed version is CVS stable from yesterday. However, we noticed a major performance hit in POSIX regular expression matches against columns using the ~* operator. http://arch.wavefire.com/badregex73.txt has explain analyze output from 7.2 and 7.3.1 respectively. Both of these tables have only 101 rows. The 7.3.1 install is using the default settings from postgresql.conf. Any ideas as to why this should be happening? Should anyone require additional information, please let me know.

Klaver reported the exact issue with PG and provided a web link which contained evidence of the performance decrease. A drop in performance in the new code over the old code is something that is not desirable in software development and will raise flags in the minds of the developer community. Five hours later, Tom Lane, a core member of PG, provided a possible explanation for the problem:

> The only thought that comes to mind is that multibyte character sets are supported in 7.3 whereas they were optional (and not default) in 7.2. I'd not have expected a factor-of-150 performance hit from that, though. Could you rebuild your backend with profiling enabled and get a gprof summary of where the time is going?

Lane acknowledged the severe drop in performance ("factor-of-150") encountered by Klaver, by examining the web link information, and he requested him to participate in the problem solving process by starting from scratch ("rebuild your backend") and enabling options in the system that would allow for proper error detection.

Three days later Klaver asked Lane how to enable "profiling" and then provided the e-mail discussion list with a web link to the system error information:

> Here is the profile information. I included a log of the session that generated it at the top of the gprof output. If there is any other info I can help you with, please let me know.
> http://arch.wavefire.com/pgregexgmon.txt

Less than 20 minutes later, Lane examined Klaver's web link and then asked him to re-run the test because the information he provided was not sufficient and gave him details on how to run the test:

> A four-second test isn't long enough to gather any statistically meaningful profile info. On most machines, gprof samples 100 times per second, so realistically you need a minute or two of runtime to have trustworthy numbers.
>
> Please replicate the rows in the table by a factor of ten or twenty or so and try again.

Klaver responded back in about two hours with further evidence which indicated that performance problem was quite severe. This allowed Lane to make an initial diagnosis of the problem at hand. Lane reported the precise change in the source code, from before, that may have caused the problem to arise in the first place. Lane used the public archives of the source code repository to identify the change made by another developer that could be the source of the problem. He also raised the issue that since there was such a significant drop in performance in the code that a impending incremental release of the PG 7.3 code may have to be re-done to solve this problem.

Although it appeared that Lane had located the source of the problem, he continued to ask Klaver questions about his particular use environment and also his set up of PG. Over the course of the evening Lane asked two more questions of Klaver's and requested him to run a few more tests. The questioning indicated that Lane was no longer sure that his initial problem identification was correct. The next day, Klaver replied back to Lane's queries with more specific data from his use environment. Klaver's further reporting caused Lane to come up with a different reason for the problems experienced by him:

> **Date: 2/4/2003 11:46 AM**
> Right, so the caching of compiled regexps that regexp.c does is of no help, and any change in its behavior in 7.3 wouldn't have made any difference anyway. I leapt to a conclusion after reviewing the CVS logs for pertinent changes, but it was the wrong conclusion. The true problem is that MULTIBYTE[43] is now forced on, and that causes some loops in the regexp compiler to change from 256 to 65536 iterations.
> <snip>
> Rather than trying to band-aid a solution like this in the main sources, I think I shall go investigate Spencer's new regexp code in Tcl, which reputedly is designed for wider-than-8-bit chars from the get-go. We've had it on the TODO list for a long time to assimilate that code; it's probably time to make it happen.

Lane indicated that the problem resided in the current implementation of RegEx in PG not in his initial diagnoses of blaming another developer's specific changes. He noted that a quick fix solution did exist to the problem but the real issue was the integration of Spencer's new RegEx code into PG. He also stated that this had been on the TODO list for sometime; and that the current situation, a significant drop in performance in a new software release, compelled him to volunteer to integrate this code into PG. Lane further noted in another e-mail that the problem lay with the PG community's modification of Spencer's original code:

[43] MULTIBYTE support means that non-Latin languages like Japanese, Chinese, Korean, Hebrew, Arabic etc can be natively supported within PG

> The real problem is simply that we're up against design limitations of the existing regex package, which was never designed for wider-than-8-bit character sets. It's been rather crudely hacked while it was in our hands (Henry Spencer would probably disown the code if he saw it now ;-)) so that it sorta kinda does MULTIBYTE, but it's slow and I don't think it's complete either.
>
> I'm about to go off and look at whether we can absorb the Tcl regex package, which is Spencer's new baby. That will not be a solution for 7.3.anything, but it could be an answer for 7.4.

Lane's raised the issue that the modifications done to RegEx by the PG community to support "MULTIBYTE" functionality caused the software to be extremely slow in certain circumstances. He also noted that the changes he was going to make would only impact the future 7.4 release and would not be suitable for 7.3.x releases where Klaver's problem originally came from.

Neil Conway, a peripheral member, responded to Lane's post by indicating that he had the same idea as Lane regarding Henry Spencer's new code and also provided an alternative solution that would not be based on Spencer's work.

> **Date: 2/4/2003 12:15 PM**
> On Tue, 2003-02-04 at 11:59, Tom Lane wrote:
> > I'm about to go off and look at whether we can absorb the Tcl regex
> > package, which is Spencer's new baby. That will not be a solution for
> > 7.3.anything, but it could be an answer for 7.4.
>
> Sounds like we had about the same idea at about the same time -- I emailed Henry Spencer inquiring about the new RE engine last night. I came across a post this post that indicates he was planning to package the new RE engine separately:
>
> http://infosoc.uni-koeln.de/pipermail/php/1999-February/000019.html
>
> but I wasn't able to find a release of it anywhere -- I'll let the list know if/when he gets back to me.
>
> Another option is to consider a different regular expression engine. At

least according to the benchmarks here,

http://ourworld.compuserve.com/homepages/john_maddock/proposals/exregex.htm

Spencer's implementation is outperformed by some other RE engines, notably PCRE (www.pcre.org). But switching to another engine might impose backward-compatibility problems, in terms of the details of the RE syntax.

Conway's post showed that the main thing that held him back, and possibly other individuals before, from utilizing Spencer's code was the availability of a RegEx library that was separate from the TCL programming language. Conceivably a separate engine would make the integration task into PG a lot easier instead of an extraction from another programming language. Thus Neil's proposal to use an alternative pre-packaged engine, PCRE, was one solution that the community could adopt.

Lane responded to Neil's e-mail by indicating that he too had contacted Henry but more as a courtesy than a requirement. Lane thought that the task of importing the new code from TCL was not going to be too difficult and he corrected Neil's contention that the newer PCRE code might be faster than Spencer's code by referring to a recent book on regular expressions:

Date: 2/4/2003 12:36 PM

Neil Conway <neilc@samurai.com> writes:

> Sounds like we had about the same idea at about the same time -- I
> emailed Henry Spencer inquiring about the new RE engine last night.

I just did that this morning ;-) ... but more as politeness than anything else. AFAICT[44] from searching the net, packaging his new code as a separate library is something that's been on Spencer's TODO list for several years now. We've been waiting for him to do it, but I'm now thinking that it's time to quit waiting. We can lift the code from Tcl with probably not all that much more work than if it were an official separate package.

<snip>

[44] AFAICT = As Far As I Can Tell.

> Spencer's implementation is outperformed by some other RE engines,
> notably PCRE (www.pcre.org).

AFAICT, that page is benchmarking Spencer's old code (the same library we started from). His new code is state-of-the-art according to Friedl in _Mastering Regular Expressions_, 2nd ed 2002 (O'Reilly).

There was further discussion amongst Lane, Neil and three other peripheral participants about other alternative regular expression packages that could be utilized in PG. However the onus was on the particular person to do the work to make such a package available for PG. Lane expressed support for their efforts but indicated that he would spend his time on Spencer's code, however, he was willing to share his know-how so that another RegEx package could be utilized in PG.

Solution Development (2/4/2003 – 2/5/2003)

Later that day, Lane, posted to the e-mail discussion list the evidence of his work on extracting Henry's code from TCL and importing it into PG. He provided time data indicating how fast the system responded with the old code and new code:

Proof of concept:
PG 7.3 using regression database[45]:
Regression 1 <snip>
Time: 676.14 ms
Regression 2 <snip>
Time: 3426.96 ms
Regression 3 <snip>
Time: 466344.48 ms

CVS tip[46] plus code extracted from Tcl[47]:
Regression 1 <snip>
Time: 472.48 ms
Regression 2 <snip>
Time: 4414.91 ms

[45] This is the test with the original code.
[46] "CVS tip" refers to the latest version of the software that is currently under development. CVS stands for concurrent versioning system (the source code repository) and tip implies the most recent version.
[47] This is the test of the new code changes made by Lane.

```
regression  3<snip>
Time: 4608.49 ms
<snip>

This is nowhere near ready to commit, but it compiles cleanly and passes
regression tests ...
```

Note that the above changes were applicable to the upcoming 7.4 code release and did not solve Klaver's problem with the 7.3.2 release. That problem was solved later in the day by Tatsuo Ishii, a core developer from Japan:

```
Ok. The original complain can be sasily[sic] solved at least for single
byte encoding databases. With the small patches(against 7.3.1)
included, I got following result.
test1:<snip>
Time: 113.81 ms
test2:<snip>
Time: 419.36 ms
test3: <snip>
Time: 1633.21 ms

The ratio for test3/test1 is now 14.35. Although not great as the
Spencer's new code according to Tom (with the code test3/test1 =
9.75), it seems much better than the original 7.3 code (test3/test1 =
689.71).
<snip>
```

Ishii also provided evidence of an approximate 14 fold improvement in performance based on his modifications and included the source code changes which could be implemented applied to an existing PG system to get similar results. There was no explicit coordination between Ishii and Lane regarding his participation in the coding process. However, Ishii's contribution was not a surprise to the community because he had previously made changes in the same area of code. The next day, Lane was appreciative of Ishii's contribution and he requested that Ishii apply the changes to the official repository. He also requested Klaver to verify that Ishii's changes worked for his use environment.

```
Date: 2/5/2003 9:59 AM
Nice work, Tatsuo!  Wade, can you confirm that this patch solves your
problem?

<snip>
```

Two hours later, Klaver confirmed the operation of Ishii's code with evidence of success

```
Date: 2/5/2003 11:50 AM
Confirmed.  Looks like a 100-fold increase. Thanx guys.
Explain output can be seen here:
http://arch.wavefire.com/pgregex.txt
```

Use and Refinement (2/5/2003)

Tom Lane updated the source code repository for the 7.4 release with the changes that incorporated the new code from Spencer. The following message was generated in the automated source code repository notification system to the e-mail list:

```
Date: 2/5/2003 12:41 PM
CVSROOT:  /cvsroot
Module name:       pgsql-server
Changes by:        tgl@postgresql.org       03/02/05 12:41:33

Modified files:
  doc/src/sgml    : func.sgml release.sgml
  src/backend/utils/adt: regexp.c
  <snip>
Added files:
  src/include/regex: regcustom.h regerrs.h regguts.h
  <snip>
Removed files:
  src/include/regex: cclass.h cname.h regex2.h utils.h
  <snip>

Log message:
  Replace regular expression package with Henry Spencer's latest version
  (extracted from Tcl 8.4.1 release, as Henry still hasn't got round to
  making it a separate library).  <snip>
```

Note how the email included information on all the files modified, added and removed due to this particular change. The log message indicated the problem Lane was trying to solve and how it was accomplished. Lane then informed the community of his change and then request their input to test the code and to help him think through various user parameters:

```
Subject: Status report: regex replacement
Date: 2/5/2003 1:10 PM
I have just committed the latest version of Henry Spencer's regex
package (lifted from Tcl 8.4.1) into CVS HEAD[48].  This code is natively
able to handle wide characters efficiently, and so it avoids the
multibyte performance problems recently exhibited by Wade Klaver.

I have not done extensive performance testing, but the new code seems
at least as fast as the old, and much faster in some cases.
<snip>
There's some stuff still to do:
<snip>
Any suggestions about the name of the parameter?
<snip>
Should it be split out as an appendix, or is it okay where it is?
<snip>
Anyone want to try some more extensive benchmarking?

<snip>
```

The next day Ishii confirmed that the new code worked on his system and that he had tested it on various language settings. Three other participants discussed with Lane the proper naming convention for the control parameter. Lane used the discussion on the naming convention to guide his changes to the RegEx's user-available settings in PG by making one more change to the source code repository the next day. There was no further public discussion about the other elements raised by Lane.

[48] "CVS HEAD" is a different way of saying"CVS tip" and refers to the latest version of the software that is currently under development. CVS stands for concurrent versioning system (the source code repository) and HEAD implies the most recent version.

3.7.3: Vignette 3 – Joint problem solving and new feature creation by core and peripheral members

The development of a new command "`--describe-config`" to support a graphical configurator (GC) illustrates novel feature creation and joint problem solving by core and periphery members. The basic PG software installation is geared towards advanced database administrators who are comfortable working in a text-based command line interface instead of the more user friendly graphical user interface (GUI). However, as PG has gained more adoption and users there have been attempts to create various GUIs for the database. These GUIs typically require the database to make available certain parameters that can be adjusted by external applications. The "`--describe-config`" feature arose out of a need by one corporate redistributor of PG, Red Hat, in its attempts to create a more friendly interface for customers. Specifically, Red Hat engineers wanted the ability to configure the database using friendly GUI menus instead of command line functionality, ultimately creating a GUI configurator or GC.

The feature was developed by Red Hat engineers in private consultations with Lane, who is also employed by Red Hat. The feature was then released to the community along with a request for inclusion into the global repository. While the feature was designed to pass parameter information between PG and an external GUI, it made several additions to the core software that had not been discussed. Over the course of four months much of the initial functionality of this feature, as envisioned by Red Hat was removed. This removal occurred because there was not sufficient agreement in the community about the validity of the changes being made by the Red Hat team and the concern that future flexibility would be compromised. In addition there was significant concern amongst core and peripheral participants that the "proper" community process had not been followed. The feature was initially developed by one peripheral member and then one core member modified it to meet community expectations. Two additional core members and five peripheral participants were involved in the various discussions pertaining to the changes.

Problem Definition and Solution Exploration (6/26/2003 – 6/27/2003)

Fernando Nasser, a peripheral member and a manager with Red Hat in Toronto, Canada initiated the feature development process by posting information on a new GUI utility that his team within Red Hat is developing. Internal development at Red Hat had made changes to the PG software that enabled the proper functioning of this utility. Nasser indicated that one of his engineers Aizaz Ahmed had worked closely (but privately) with Tom Lane on developing the appropriate server-side changes. His hope was that the community would accept these changes and make them part of the next PG release:

> Subject: pg_guc
> Date: 6/26/2003 1:28 PM
> Hi Peter[49],
> We have a server side GUI utility that among other things let us configure GUC variables. We badly need to know what variables exist in the specific backend version, which are the min and max values and if possible a description. <snip>
> Aizaz have, with hints from Tom Lane, implemented a basic version of such utility. We thought that this can be used by other tools as well, so it would be nice to have it added to the 7.4 release.
> <snip>
> Anyway, I hope you find this useful and people find the motivation to enhance it. Aizaz is already working on the internationalization.

It is interesting to observe that Nasser broke with convention by addressing the email directly to "Peter" Eisentraut who was a core developer from Germany and had worked on other related backend related activities. Eisentraut responded the same day by giving Nasser suggestion on how to implement the feature in the software along with recommendations on user-specific options:

> In that case I think it's best to put it directly into the server executable and add an option like --help-long or possibly some variations if you need specific program-parsable formats. This would certainly

[49] This refers to Peter Eisentraut, a steering committee member. Its highly unusual for emails within the community to be addressed to a particular person.

> solve a few of the implementation concerns I've heard about, and it's also a fairly logical place to look for it.
> <snip>

Lane joined the conversation by noting that Eisentraut's suggestions had been considered before and rejected. However, he notes that on second thoughts, they may be more appropriate and gives instructions to Ahmed on how to implement Peter's recommendations:

> Hm. We had toyed with that idea for a bit but rejected it on the grounds that it would add bloat to the postgres executable. On the other hand, two sets of message catalogs would bloat an installation even more. Maybe that's the best plan after all.
>
> Aizaz, if you look at backend/main/main.c it should be pretty obvious how to handle this --- it's just like bootstrap mode. main.c kicks off control to GucInfoMain or whatever we call it, and then that routine can act pretty much the same as if it were the actual main() of a standalone pg_guc. <snip> in fact, I think quite a large percentage of the patch disappears ...

Ahmed and Lane then had three public exchanges on the technical issues related to the implementation of those issues. Note that these discussion were taking place between two employees of Red Hat in a public forum. There was no attempt to take the discussion private and exclude the rest of the community in the solution exploration phase.

This phase of the vignette is interesting because the feature under consideration was already developed, in the sense that this was not a proposal or a design conversation, rather, the work on it had been substantially completed and Red Hat was asking for inclusion of the code. However Nasser had not submitted any code yet, instead, he broadly outlined the new functionality. The timing of this feature is also salient because it was announced to the community three days before a self-imposed feature freeze deadline of July 1, 2003. During a feature freeze, all new functionality that is developed by the community is put on hold, that is, it is not committed to the source code repository,

and instead the community development effort shifts to beta testing of the existing code base so that a new official version of the software can be released. This announcement without the code and the lack of significant discussion about the functionality posed significant problems during the beta testing phase.

Solution Development (6/30/2003)

On 6/30/2003 Ahmed sent in the computer code to the email list and asked for its inclusion into the base software:

> The attached patch adds the --long-help option[50] to the server executable. This option displays all the available runtime options for that
> particular server version, along with Max, Min and Reset values if applicable and a description. It also groups the runtime options together in accordance with the documentation.

The code for this feature arrived on the day of the deadline and Eisentraut within four hours had more comments and concerns. The feature itself looked rushed because community members did not have a significant amount of time to discuss its contents. Eisentraut raised a few issues that seemed to question the detailed design of the implementation and the changes it was making to the source code. He indicated significant reservations with what the code was trying to implement and how it was being done:

> Conceptual comments:
>
> If the option is named --long-help, I'd expect a longer version of --help, which this is not. The name should probably involve "help" and "config" to make it clearer what you get. (Personally, I think "help" should go before the qualifying word, but there may be other opinions.)
> Do we really want to encode the notion of option categories into the source code? This looks like a pretty large burden to me.

[50] For the GC feature.

```
<snip>

Code comments:
<snip>
There is already a file guc.c, why should there be a file pg_guc.c
now?  That doesn't make sense; the names should be differentiated
better.

Why have various things been moved from guc.h to guc_vars.h, which
seems to just split things up uselessly?
<snip>

Should options not for general use (e.g., session_auth_is_superuser)
be hidden from this tool?  Are they?  What other provisions of this
kind does this tool make?
```

Lane responded to Eisentraut's critique with specific answers to each of his questions and also invited Ahmed to provide some comments. While Lane's answers provided a justification for each of Eisentraut's concerns, Ahmed provided more background information on the code writing assumptions he made. There was no further public discussion on the email list about it and the changes were committed to the source code repository a few days later. On 7/4/2003 Lane committed the patch to the source code repository which automatically generated the following message to the rest of the community:

```
CVSROOT:  /cvsroot
Module name:       pgsql-server
Changes by:        tgl@svr1.postgresql.org   03/07/04 13:41:22
Modified files:
  doc/src/sgml    : runtime.sgml
  src/backend/main: main.c
  src/backend/tcop: postgres.c
  src/backend/utils/misc: Makefile guc.c postgresql.conf.sample
  src/bin/initdb : initdb.sh
Added files:
  src/backend/utils/misc: help_config.c
  src/include/utils: guc_tables.h help_config.h
Log message:
  Add --help-config facility to dump information about GUC parameters
  without needing a running backend.  Reorder postgresql.conf.sample
```

```
to match new layout of runtime.sgml.  This commit re-adds work lost
in Wednesday's crash.
```

Use and Refinement (09/28/2003 – 10/18/2003)

There was no subsequent discussion on the hackers or patches list until 9/28/2003. The PG 7.4 release was then in active beta testing and work was being completed so that multiple world languages would have native support. Alvarao Herrera, a peripheral community developer from Chile, was working on language internationalization for PG, when he noted some inconsistent results from his tests of the software:

```
Date: 9/28/2003 3:36 PM
Subject: more i18n/l10n issues
<snip>
Now for something completely different:
The postmaster executable shows --help display perfectly localized.
However I just noted that postgres --help output (the standalone
backend) does not; is it not i18n'ed, or is some sort of missetup?

However, if I try postgres --help-config the localized display is shown,
but postmaster --help-config says
<snip>
So, for some things postmaster is localized, but others are not
supported; and for some things postgres is localized, but for others
it's not.  Is this some sort of planned behavior, inconsistency, or
plain oversight?
```

He then conducted some additional test on this odd behavior and within an hour wrote some additional software code to provide a temporary stop gap measure:

```
>Now for something completely different:

Oh, there's another thing about the --help-config option.  This option
includes an, er, option to display the items that belong to a given
group.
<snip>
This little patch allows both versions to match, translated and
untranslated.
```

Herrera's identification of this problem and his subsequent code caused Eisentraut to express significant reservations about the original feature developed by Ahmed and committed by Lane:

> I'm quite unhappy about the --help-config option. It was developed without discussion, it was installed hastily, we don't have any information about that interactive configuration application it's supposed to target, it's not documented, it's full of unfinished business, it certainly doesn't make the code easier to maintain because all the documentation is duplicated, but not one-to-one. At this point, I wouldn't spend a lot of time trying to make sense of it. We can revisit it again in the next release and investigate how we can eliminate the duplication of effort between the documentation and the code.
> <snip>
> Again, I don't think we should worry about that now or we'll end up spending too much time on this.

Herrera responded back to Eisentraut saying that he would leave it alone and also wondered about the usefulness of that feature. Lane then joined the conversation two hours later by providing some context regarding why the feature was created in the first place:

> Alvaro Herrera <alvherre@dcc.uchile.cl> writes:
> > If you put it that way :-) I'll leave it alone. I hope it can be
> > enhanced in the next release. I'm not sure of it usefulness anyway;
> > the documentation seems good enough.
>
> Some guys at Red Hat wanted it to support an admin tool that should see the light of day Real Soon Now. Peter's right that it could be improved though; in particular I would not care to defend its i18n behavior. I've left it undocumented partly because I figure we'll be changing it.

A few hours later, Dave Page, a peripheral community member from the UK and developer of another GUI tool for PG complained about the special treatment given to Red Hat and the lack of proper process. His complaint was directly aimed at Lane:

> Hi Guys,
>
> I find this a little worrying because if we want a feature or tweak for pgAdmin we usually have to fight tooth & nail to justify getting it committed (which is not a bad thing), however 'some guys at Red Hat' are getting switches added to the postmaster without any discussion? I realise they pay the wages of at least one of the developers many of us depend on, but surely they should have to justify their modifications as the rest of us do?

Eisentraut seconded Page's concern and asked, in a general sense, for the Red Hat employees, to specify what was required for their GC tool because the current implementation of the feature was not appropriate or "future-proof":

> It was not a nice thing to do.
>
> Could whoever is responsible for this admin tool at Red Hat please specify exactly what data they need out of this interface, so we have a chance to make the interface a little more future-proof now and possibly remove some of the unneeded clutter that is giving translators problems? Surely that would be in everyone's interest, because if we're already set on changing the feature again pretty soon, it won't do that admin tool much good.

Lane responded to Eisentraut's (and Page's) concern about not following the community process by giving evidence of prior discussion on the feature and admitting that perhaps he had been too hasty in getting the feature included in the source code repository while also justifying his choices:

> Peter Eisentraut <peter_e@gmx.net> writes:
> > It was not a nice thing to do.
>
> Gimme a break, guys. There *was* discussion, e.g. here,
> http://archives.postgresql.org/pgsql-hackers/2003-06/msg01092.php
> and the patch was posted for review, see this thread:
> http://archives.postgresql.org/pgsql-patches/2003-06/msg00420.php
>
> I'll admit that I applied the patch with more than usual speed, but that was because we were right up against our self-imposed feature freeze

> deadline for 7.4, and I didn't see any big objections. The biggest gripe left over at the end of the above-mentioned patches thread was that the message texts were unpolished, but as even Peter agreed, that could be fixed later. So MHO is let's fix them now.
> <snip>
> I would like to think that the patch would eventually allow us to generate postgresql.conf.sample automatically from the guc.c tables, and thereby reduce the number of files to maintain, but that didn't get done yet.
> <snip>

Lane's response triggered Page to apologize for relying on Eisentraut's assertions on the lack of community process and it caused Eisentraut to apologize as well but maintain his stance that the feature was not appropriate and needed to be corrected:

> I confused this with the private mails that we exchanged. Sorry.
> <snip>
> OK, but does this tool actually need all of the following features:
> <snip>
> Were some of these just added for "completeness"? With what rationale?
> <snip>
> Also, --help-config 'foo' outputs all parameters matching 'foo' somewhere in the string, not only 'foo'. I think that is a misdesign.

Lane responded immediately by noting that he has asked his fellow Red Hat employees to provide clarification and explanation as asked by Eisentraut:

> I've asked the Red Hat folks who did the detail design to respond to this. I'm not sure if they had specific use-cases in mind for those behaviors, or were just trying to make the feature useful for manual invocation. I would think the GUI tool doesn't need most of those behaviors, but can't swear to it.

On 10/1/2003, Lane posted the reply he received from Nasser regarding all the questions that Eisentraut had. The reply gave details on the questions raised by Eisentraut with an explanation for the logic of all the functionality for the GC tool and changes in the PG source code to accommodate those requirements.

On 10/9/2003, Momjian, wrote an email to the developer list expressing his serious reservations with this particular feature. He outlined that the feature had options that might curtail future development, lacked proper discussion and documentation and implied a violation of the norms of the community:

> 10/9/2003 7:15 PM
> Subject: postgres --help-config
> Where should I start on this? :-)
>
> The implementation of postgres --help-config has several issues:
>
> - it has options added "just in case someone ever need them"
> - it has capital letters to negate, which we have never used before
> - uses GNU message reporting style
> - there was little discussion about how this should work
>
> Its biggest "plus" is that it isn't documented :-) --- because the API[51] might change in the next release, and no one wants to stand up for this.
>
> If Red Hat didn't do so much for PostgreSQL by hiring Tom and others, I would be more upset, but putting something in for the convenience of some Red Hat tool isn't something I would like to see happen regularly, and I think we can all agree on that.
>
> I can't even put a mention on the TODO that this has to be cleaned up because I would then advertise it. I will put it on my personal list and we can discuss how this is supposed to for 7.5.

Lane fired back at Momjian for missing the prior work in June and not providing specific issues that needed correcting:

> Where should I start on all the people who are complaining now, but said not a word when the patch was put up for review?
>
> I'm quite annoyed at these claims that procedure wasn't followed. It's either selective memory or historical revisionism, and either way I feel it's unfair to me and to Red Hat.

[51] Application Programming Interface.

> Let's see some specific suggestions for improvement, rather than bootless complaining. I'm quite prepared to agree that the patch could use improvement. If we can fix it before 7.4 release, let's do so.

Momjian then responded to Lane by giving him detailed comments on specific areas that needed to be fixed. On 10/13/2003 Lane responded k to Momjian by agreeing with his requirements for a fix and outlined a general principle regarding how new functionality is added to the software:

> Bruce Momjian <pgman@candle.pha.pa.us> writes:
> > Is that enough feedback? :-)
> Yup, thanks for the comments. I am not sure how much of the existing --help-config functionality is actually needed by Red Hat's tool, so I've asked those guys to respond.
>
> Personally I agree with the idea of stripping out whatever functionality isn't immediately necessary. It seems to me we've generally been more successful by adding features in response to specific demand rather than inventing things we think might be needed.

The next day Nasser responded specifically to Momjian and provided detailed answers to his concerns he also raised the point that Momjian should have provided comments on the submission back in June:

> Before I comment on your suggestions, I would like to mention that many of the things below were added on request by the few people who cared to comment on it. Aizaz spent most of his time changing here and there to accommodate these requests. Anyway, we know we can't satisfy all, but I wish people would be more timely when criticizing things. Aizaz is away until May and left thinking everything was under control. And we developed a very nice tool that depends on this feature confident that we could count on it.
> <snip>

Momjian responded to Nasser comments by complaining about process and fairness to other firms that may be participating with the community. A key concern he noted was

that there was significant discrepancy between what were the explicitly discussed changes and options and the actual code. He also recommended that a stripped down version of the feature may be most appropriate for the present circumstances:

> I understand this is not ideal timing. However, open source is certainly very fluid and it is hard when someone adds something and then isn't available for later adjustments.
>
> I knew you were adding --help-config, but I didn't realize the extent of the "features". The commit message is:
>
> revision 1.1
> date: 2003/07/04 16:41:21; author: tgl; state: Exp;
> Add --help-config facility to dump information about GUC parameters without needing a running backend. Reorder postgresql.conf.sample to match new layout of runtime.sgml. This commit re-adds work lost in Wednesday's crash.
>
> which I thought was a single option, which we all knew was needed, not six additional options for output format. Also, with no documentation, or
> a posted list of the flags you wanted to add (at least I never saw it), it was easy to miss.
> <snip>
>
> Let me be clear on this --- your tools is not part of the PostgreSQL community. We are not required to allow any of this functionality unless the community decides they want it. The major argument for keeping it, in my mind, is to be helpful to Red Hat.
>
> My current idea is to keep --help-config as readable output, add --help-config-raw as machine-readable output, document those, and remove all the additional flags.

Eisentraut joined the conversation by recommending that the entire new feature be removed in the upcoming release and to devote more time to it in the future. Momjian replied to him by stating that Red Hat would be left in a lurch by these late changes and that he would prefer that they find a way to meet the minimal requirements. Lane joined

the conversation by reiterating his stance that the original problem was a lack of timely participation by Momjian in the original June review process:

> >>Let me be clear on this --- your tools is not part of the PostgreSQL
> > > community. We are not required to allow any of this functionality
> > > unless the community decides they want it.
>
> I'm really having a hard time responding to this line of argument politely. Where were all these complaints when the patch was proposed and accepted? If there's not time now to redesign the feature to your liking, it is *NOT* Red Hat's fault, it is *YOURS*. Yanking the rug out from under someone else's project just because you didn't review the patch adequately at the time is not my idea of how a community should act.

Momjian in reply noted that he did not actively participate in the review process because he trusted Lane's original judgments. He further reiterated his point that the community process was not followed by either Red Hat or Lane:

> I thought you might have an emotional reaction to this issue.
> I did not review these changes thoroughly because:
> - o There was no proposal on the switches and their usage.
> - o The commit message didn't mention any switches other than --help-config.
> - o There are no docs to show the new flags.
> - o You were handling it, and I trusted your style, so I didn't see a reason to study it more thoroughly.
>
> Let's imagine how this would have worked for an outside project/company:
> - o Project leader comes to us and says they want to make a PostgreSQL admin tool.
> - o They explain their needs and we agree on how to implement it.
> - o We implement the feature as discussed.
>
> Would we have agreed to adding all those flags? I don't think so. We would have given them a clean output, and asked them to handle the functionality in their code, which is probably the correct approach.
>
> This procedure is in our developer's FAQ:
>
> *The usual process for source additions is:*

o Review the TODO list.
o Discuss hackers the desirability of the fix/feature.
o How should it behave in complex circumstances?
o How should it be implemented?
o Submit the patch to the patches list.
o Answer email questions.
o Wait for the patch to be applied.

Now, we have Red Hat having you add a patch on July 4 (posted for review June 30), very near feature freeze, but it meets a discussed need (--help-config), so it goes in. I only learned about it when Peter saw the C code handling the new flags and asked questions about it. I do see the patch submitted, with clear illustration of the flags:

http://archives.postgresql.org/pgsql-patches/2003-06/msg00420.php

I guess I thought those flags were for Red Hat's tool or a separate utility, but it clearly states it is part of the postgres binary, so that was my fault.

<snip>

Lane responded to Momjian's message by asking for group consensus on how to work with the existing feature and still meet the needs of Red Hat:

It'd be better if we could get it right the first time, with the understanding that the output format is not very negotiable at this late hour. But as best I can tell, most of the unhappiness is with the design of the switch set, which is not something I want to defend in detail. There's a lot there that isn't needed for the RHDB tool as I understand it, and I think that altering the switches used to get the output that the tool does need would still be a feasible change from the tool's point of view.

I would be in favor of simplifying the supported switch set to the minimum needed by Red Hat's tool (the equivalent of -G -M if I understood Fernando correctly), and re-adding complexity in future when and if it's shown to be needed. But we need to make a decision about this now. Preferably yesterday.

Between 10/15/2003 and 10/18/2003 there was further technical discussion on the list regarding what would be the minimally acceptable set of option in the new feature that would satisfy both the community and Red Hat. In the end on 10/18/2003, Eisentraut makes changes to the source code and committed the revised code to the CVS repository. His log message on the source code archives stated what was done:

```
Cleanup on --help-config: Now called --describe-config, no further
options, machine readable, without headers, not sorted.  Parameter
descriptions adjusted to fit first sentence + rest convention.
```

This phase of the vignette showed the operation of a distributed problem solving system and the dynamics of a community development process. There was an approximately two and half month gap between the inclusion of the feature into the global repository and the chance discovery of the associated issues by a periphery member. It is interesting to note that even though Eisentraut, as a core member, had initial issues with code as submitted, he never publicly expressed his serious concerns during the first two phases of the development. Only after Herrera had inadvertently found concrete evidence of some issues, did he raise strenuous objections. Herrera's discovery of the offending code highlights the importance of a distributed problem solving process where many diverse participants "test" the entire software system for their own particular purpose and report back anomalous findings.

The severe public disagreements between Lane, Eisentraut and Momjian exposed the expectations of a community development process. Eisentraut and Momjian's main concern was that the community process had been both short-circuited and violated. Lane's self-admission of a hasty commit of the code to the CVS repository indicated that he himself realized that he had perhaps moved too fast in the face of an impeding deadline. Along with the speed of the commit, there was also concern that the Red Hat triggered changes had not been fully disclosed. The discussion on the email list along with the commit message from Lane did not fully reveal the extent of changes within the base PG system, thus violating the expectation of full disclosure in the community.

Lane's specific reaction to this episode also highlighted conflict management within the community. Without doubt, Lane was remains one of the most important core contributors to the PG community. However this status was based on current and specific performance and did not make him immune from being severely questioned and tested by the community. His own defense regarding this episode was that Momjian and Eisentraut should have spoken up during the initial phases of the development cycle. From his perspective, lack of participation implied agreement. However, both core and periphery members noted that the spirit of the community process had not been followed. Momjian even went further when he stated that his own lack of participation was due to the fact that he had trusted Lane's judgment and felt that he could rely on him to make the correct choices. Lane's response was to both embrace the criticism but also request specific concrete changes to the code base. Publicly admitting that he was "wrong" and getting the community to agree on a compromise between what the community needed and what Red Hat needed was accomplished via his insistence on specific changes. He did not use his status to push through the changes that were made, even though, his employer Red Hat, had sponsored those changes and had made adjustments in their own product road map expecting those changes to be permanent.

3.8: Distributed Problem Solving Practices in the PostgreSQL Community

Through my analysis of the innovation process histories and the 32 vignettes I identified an array of activities that enable the PG community to solve technical problems. These activities form a repertoire of practices that are recurrently enacted by the distributed PG membership and enable the community to continuously innovate and keep producing and reproducing their collective output. Table 3.10 outlines the practices and their constituent activities for distributed problem solving. I have grouped the practices as collective and individual to highlight the sociological construct that "work" is a duality of independent and interactive tasks (Hughes 1971: 304). Hughes argued that the division of labor implicitly implied interaction because ultimately the divided labor

and its constituent actions had to be integrated to form a complete "whole." The act of integrating the "whole" requires contributing individually and then interacting over those contributions.

The collective practices are done with and in response to actions from others. The practice of *work broadcasting* serves to mobilize the community around problems; to jointly create, transfer and transform knowledge across the community; and to informally coordinate action. *Building and using community memory* is a practice that creates shared understanding between the various community members. This practice enables current and future community members to access technological objects and the related social interactions in order to develop a common sensibility around past and current activities in the community. The collective practice of *distributed decision making* helps the community make choices and too keep options open so that possibilities for innovation and breakthroughs by members are preserved.

The individual practices are what community members do in response to their own requirements and the what they observe happening in the community. Ultimately action at the individual needs to be taken in order for the software to be developed, thus, *choosing type and level of participation* is a practice observed in the community. Individuals need to figure out what to do and then to do it. Since the community is focused on creating a technological artifact that serves a material purpose, *using the community's output,* that is, its software, is a practice that connects individuals to the community and shapes the future actions and interactions of the membership. It also serves to "objectively" test the status of the collective output of the community. Finally in a setting of relative strangers, accomplishing technical tasks, there is a need to provide evidence of accomplishment and to confirm claims of efficacy. *Evidence provisioning* is a practice that enables community members to concretely demonstrate their technological accomplishments and to confirm claims of efficacy. In a setting of relative strangers, evidence serves as a social and technical lubricant for continual interaction and the ability to jointly share and assess knowledge and objects. It is important to note that these practices should not be seen as completely exclusive or exhaustive. They are not to be

viewed as "instrumentable" constructs to be used in an analytical regression. Rather, "in practice," they are highly intertwined and interdependent. The distinctions are for analytic purposes and allow us to create a general theory of distributed problem solving.

Table 3.10 – Practices for Community-Based Distributed Problem Solving

Practices	Activities	How they enable distributed problem solving
	Collective Practices	
Work Broadcasting	Posing issues and problems Providing ideas and solution Contributing code Providing feedback	Mobilize community Create, transfer and transform knowledge
Building and using community memory	Global and local archiving of community activity Retrieval and use of community activity	Build mutual understanding over time and space
Distributed decision making	Incremental and local planning Provisional settlements Lazy consensus	Preserving choices and making choices in the community
	Individual Practices	
Choosing type and level of participation	Identifying tasks Doing tasks	Doing the actual work of the community
Using the community's output	Using latest code Integrating latest code	Connects individuals to community Testing community output
Coordinating action and building trust through evidence	Providing evidence of claims Seeking evidence of claims Testing evidence of claims	Enable self and others to assess claims and outputs Share and assess knowledge

3.8.1: Collective Practice 1 - Work Broadcasting

At the heart of collective problem solving in a F/OSS community is the practice of *work broadcasting* through participant interactions on public e-mail lists. The e-mail list is ***the*** "place" where discussion of the software development activity for the community occurs. E-mail messages become the medium (work happens inside of an e-mail message) and/or carriers (actual code or signals of code completion) of all the community work. E-mail becomes the medium of work when community members use email to respond to a request for help or ask questions. Some of the work of the community then happens in email messages. Email is a carrier of the community's work when individuals do activity outside of their community participation and then present the work through e-mail, for example writing software code on a computer and then sending it to the community. In either case, the email list then broadcasts all the work to interested subscribers. Community members can choose their level of immersion in the stream of work broadcast. They can subscribe to all the e-mail lists of the community and be continually updated about all the activity at all times, or they can be selective about which lists they subscribe to and the methods of subscription (individual emails, daily digests or archived web pages)[52].

The practice of work broadcasting provides an ongoing signal[53] to the entire distributed membership as to current state of affairs in the community. Work and the communication about the work are not separated. The act of working and participating in the community becomes the act of communicating as well. The availability of the public broadcast mechanism allows the peripheral members to participate with the core members in identifying issues and problems.

I identified four related activities that constitute the practice of work broadcasting; 1) pose issues and problems; 2) propose ideas and solutions; 3) contribute code; and 4) provide public feedback. Problem solving in the community occurs via members

[52] Individuals can be subscribed to receive every email generated on the list, or an autmated digest of all the email traffic on the list for that day. Others can participate by visiting the email list's archives where all of the conversations are presereved on a web page.
[53] Each community generates approximately 50-75 e-mail messages/day.

enacting these activities and responding to the activities of others. Contributors can enact multiple activities at the same time as part of their interaction with the community.

Pose issues and problem

The e-mail list is the place to raise problems and issues and get community engagement. The main purpose of this activity is to mobilize the wider developer community to pay attention to the particular problem faced by the member and to initiate problem solving within the community. By publicly broadcasting issues, the member attempts to transfer the ownership of the problem from the individual to the community and to trigger others to take interest in the problem. Problems or issues can be immediate, as in "there is a bug in the system," or can reflect a desire for new or improved functionality, as in "can PG do the following task?"

In the case of a software bug, contributors provide detailed information about the problem at hand combined with computer generated output regarding the bug and instructions on how to create it again. As illustrated in the Regular Expressions vignette, a relatively new community member, Wade Klaver, provided very specific information about the degrading performance issues he was experiencing along with a URL to computer generated output that would enable other community members to diagnose the problem further and provide a solution. Klaver mobilized the attention of the community by reporting his problem and providing concrete evidence about its impact on his system. The problem Klaver reported was not of a catastrophic nature, but it was of concern to the developer community because a new version of the software was performing poorly as compared to an older version in the same use environment. The indication of a severe drop in performance created the impetus for other community members to take interest in the bug and contribute their time and effort in resolving the bug.

Participants also engaged the community by reporting issues that they had with the software program. Issues were typically related to improvements in the software that are needed by the participants due to their particular use environment. As observed in the Auto Vacuum vignette, Mario Weilguni, a peripheral developer, raised the need for better

database vacuuming functionality by posting it on the e-mail list. Weilguni explicitly identified how the current vacuuming feature was not suitable for the large databases that he was encountering. Community attention was secured by generalizing the issue to other participants and getting them interested in resolving the issue.

Propose ideas and solution

Solution proposals can be in response to specific issues raised by another participant or can be combined with an issue statement. The form of solutions and ideas proposed can vary from general thoughts on how a problem can be solved to very specific implementation details and a potential plan of attack. The Auto Vacuum vignette shows that there were five distinct proposals to solving the vacuuming problem. Weilguni, the person who raised the original issue, embedded a potential solution in the original e-mail. Daithankar in response to Weilguni's problem statement proposed two solutions. The first implied creating a computer script[54] to automate the existing vacuuming functionality. This was followed by Rod Taylor's outline of how the script should be written. Daithankar's second proposal was significantly more detailed and included his design for the code he planned to write and incorporated suggestions from Weilguni and Taylor. The fourth proposal was from Tom Lane and it was in response to and a significant deviation from Daithankar's second proposal. Finally, Weilguni responded to Lane's critique of Daithankar's design by proposing yet another design to accomplish auto vacuuming. Weilguni's proposal used an existing feature of PG and he demonstrated its feasibility by providing a link to his own website that showed the core of his design in operation.

In the case of the Regular Expressions vignette, Lane's requests for further testing and evidence was integral to his proposing solutions to Klaver's problem. Throughout the problem diagnosis phase, Lane continuously proposed new solutions as new evidence was distributed by Klaver. Once he identified the exact cause of the problem (the old RegEx package) he then proposed a final solution which involved

[54] A script is a very simple computer program that automatically runs various tasks in a pre-specified manner.

replacing the existing module with a newer version. The proposal of this solution generated further alternative solutions from other peripheral community members. In all, three different types of RegEx packages were discussed as possible alternatives to Spencer's module. The open solution proposal process generated alternatives for the community. Merits of the other systems were discussed and Lane indicated a willingness to provide an interface to other developers based on his work with Spencer's code if they so desired.

The broadcasting of proposed ideas and solutions triggered other community members to review the proposed ideas and if they were not satisfied, to propose their own alternative or modified solutions. The community members engage in collective problem solving by building and extending on each other's contribution. The public broadcast of specific and concrete ideas and solutions coordinated the activity of other participants by signaling potential areas where work could be done. As before, the expression of the work (i.e. proposing an idea or solution) became the coordination signal for others participants. However the signal contained content information only and did not explicitly specify who might be an appropriate respondent.

Contribute software code

Ultimately, community members' contribution of software code is what creates and sustains F/OSS communities. Code contribution can occur in indirect and direct ways. In the Regular Expression vignette we observed Spencer contributing code to the community in an indirect manner. Spencer was not officially part of the PG community, however his efforts in another F/OSS project were directly applicable to the needs of the PG community. He passively contributed code by making it available with an intellectual property license that enabled appropriation by others. Similarly, Taylor's contribution of a small script demonstrating how Daithankar's initial ideas on how vacuuming could be implemented was another indirect way of contributing code to the community.

Software code by peripheral community members is typically sent on the public e-mail lists. Peripheral members create a "patch[55]" file and then submit that file in an e-mail to the appropriate list. The arrival of a patch from peripheral members signals to the community that someone has done programming work that should be reviewed and perhaps integrated in the official source code repository. The patch file contains information so that anyone, who has the appropriate version of the software source code, can apply the patch to their local machines and assess the veracity of the claims being made by the potential contributor. In the PG community, patches from the periphery are usually sent to a dedicated "patches" e-mail list where other interested developers can provide feedback (see below for more on feedback) on the specific software change being made. In the AV vignette, we observed that Daithankar placed his code on a website and provided a link to it for others to try out. O'Connor's subsequent modifications of the AV code were then posted on the development list until a consensus emerged as to the appropriate location for it within the existing software code structure.

Core members of the community apply their own software patches directly to the source code repository and do not necessarily have to go through a community review process. However, as soon as a patch is applied, the source code repository automatically generates an e-mail to the development lists indicating the nature of the change being made, the location of the change, the author of the change and any relevant author generated message. This automatic generation of the e-mail and its subsequent broadcast allows other community members to be up to date on the ongoing changes being made to the source code and to react to it if appropriate. Typically, as a community norm, core members will inform the development e-mail list that they are going to make a change or have made already made a change to the source code repository. Tom Lane's participation in the Regular Expression vignette illustrates how a core member kept the development community informed of his activity. The accompanying automatic email message from the source code repository included all the relevant information for a knowledgeable participant to understand and follow the exact changes that were made.

[55] A patch file is a small computer script that contains only the differences needed original source code files and the modifications desired. A patch file is inserted into the source code of an existing system to create a new version.

Provide public feedback

When participants report issues, propose solutions and/or contribute code they implicitly or explicitly ask for feedback from the community. Feedback serves as a coordination and problem-solving function in the community. Coordination occurs because the request for feedback, the actual feedback and the response to the feedback facilitates (micro) alignment of effort between different individuals in the community. The request for feedback signals areas in which an individual needs help and guides potential help providers to furnish related information. Problem-solving occurs because the request for feedback allows community members to present alternative solutions for the problem at hand and/or redefine both problems and proposed solutions.

Asking for feedback can be in the form of a simple courtesy at the end of an email message or can be related to very specific issues and items. The RegEx vignette illustrates the typical form, where community members, both core and periphery, closed their dialog with an invitation for feedback from the community. Depending on the situation and the individuals, the request for feedback can be very detailed, as observed in Daithankar's second proposal for Auto Vacuuming where he outlined six areas where he needed help. Feedback from the community was in the form of questioning some assumptions in the design of the proposed solution, review of submitted source code, providing confirmatory or contrary information about issues raised, or acknowledging the utility of a solution. Lane strongly critiqued Daithankar's second proposal and indicated that the solution as proposed was unacceptable to him. This feedback provided Daithankar, a relatively new peripheral participant, with an understanding of the challenges he faced in contributing to the community, and encouraged him to consider information about other potential solutions.

Matthew O'Connor's submission of the updated AV source code provides an illustration of the feedback process in the community. Momjian, a core developer, and Magnus Neusland, a peripheral user on the development list acknowledged the general usefulness of the solution. Bruce Momjian stated that the code submission was good for

the community and that it was relevant for the upcoming release. Momjian's approval provided legitimating for O'Connor's efforts and increased the likelihood that his changes would be incorporated into the system. Neusland gave direct user feedback indicating that he had already incorporated the changes into a live running system with positive results. His feedback was important because it verified the general occurrence of this problem and the direct utility of the solution being proposed.

Even after all the glowing reviews, O'Connor's code submission also generated critique of his programming syntax along with specific suggestions for change in the underlying logical structure and design of the program. Neil Conway, a peripheral developer, provided feedback on both programming style and logic issues. Conway specifically identified the areas that needed changes and also suggested how those changes might be undertaken. Feedback does not occur in isolation and other members can also further participate by modifying or correcting the previous feedback. Thus we observed Lane correcting Conway's suggestions to O'Connor. All of the feedback that O'Connor received results in a further modified code submission and acknowledgement of the impact feedback.

A concern with this practice may be that the sheer volume of broadcasts may overwhelm participants. The large volume may cause "weak" but important signals to be missed by the community. Thus a report of a problem or suggestion for a new feature may get completely swamped out if both core and periphery community members are not diligent with all the communications that occur. The large volume may also pose too high a cost for participation for new members as they may not have the ability or the skill to be able to navigate a community email list with over 75 technical messages per day.

3.8.2: Collective Practice 2 – Building and Using Community Memory

While a vast majority of community members do not work together at the same time or in the same place, they still need to engage in collective problem solving and action coordination in order to jointly produce a highly complex software product. The

asynchronous, distributed and virtual nature of community participation becomes an advantage by utilizing a simple and lean electronic infrastructure that is based on written text. Written text is both the means of communication about work (emails) and the ultimate product of the work (software code). This reliance on text in both the work process and the end-product enable the community to develop a collective memory that is both local and global, that is, the textual artifacts of communication and software code are stored on an individual's local machines and community owned websites. Individual community members can easily store and refer to the community's textual process and output. At a global level, the community also represents itself via publicly storing the textual process and output.

The practice of building and using community memory does not entail dedicated memory creation and management tasks. Rather, the tools that the community uses for communication (e-mail lists) and software development (software repository), have built in memory archiving. Thus, community memory is created as part of doing the work itself and is available for search, reflection and debate by current and future community members. Two activities constitute this practice: 1) global and local archiving of current community activity; and 2) retrieval and use of previous community activity.

Archiving community activity

The electronic infrastructure used by the community members enables automatic global and local archiving of community activity. Community activity centers around email discussions and changes and updates to the community's software repository. At the global level, all public email discussions are immediately archived and made available on the community's website. The email archive enables the creation of a memory system that allows community members to go back in time and trace the evolution of decisions and issues inside the community.[56] Similarly all changes in the software code are publicly archived and available for analysis. One of the key features of the software repository is to maintain version control of all the official changes. This enables the community to reverse any changes that may cause issues or bugs in the

[56] It is also the availability of these archives that enables me to do my research!

software system. Both the email list archives and the software repository act as virtual rendezvous points for community activity. Members from around the world "congregate" at these points and are able to observe the activities of others as they interact with the community. In addition, members can utilize the latest technologies to search (e.g. using Google) the archives and access and read previous conversations and technical changes in the community. Each community member can also maintain a local archive of the community's activity. All members act in the community via email lists and can retain community conversations across space and time by storing related email on their local computers. Similarly, members who develop software in the community can mirror the global software repository on their own local machines. This enables local members to be in sync with community activity.

The PG community does not explicitly view its electronic infrastructure as a source of community memory. However the Frequently Asked Questions (FAQ) of the PG website[57] provides guidance to potential new contributors by highlighting the importance of the community's memory systems:

> 1.1) How go I get involved in PostgreSQL development?
> Download the code and have a look around. See 1.7.
> Subscribe to and read the pgsql-hackers mailing list (often termed 'hackers'). This is where the major contributors and core members of the project discuss development.
> <snip>
> 1.3) What areas need work?
> Outstanding features are detailed in the TODO list. This is located in doc/TODO in the source distribution or at http://developer.postgresql.org/todo.php.
> You can learn more about these features by consulting the archives, the SQL standards and the recommend texts (see 1.10).
> 1.4) What do I do after choosing an item to work on?
> Send an email to pgsql-hackers with a proposal for what you want to do (assuming your contribution is not trivial). Working in isolation is not advisable: others may be working on the same TODO item; you may have misunderstood the TODO item; your approach may benefit from the review of others.
> <snip>

[57] Available at http://developer.postgresql.org/readtext.php?src/FAQ/FAQ_DEV.html+Developers-FAQ

1.7) How do I download/update the current source tree[58]?
There are several ways to obtain the source tree. Occasional developers can just get the most recent source tree snapshot from ftp://ftp.postgresql.org.
Regular developers may want to take advantage of anonymous access to our source code management system. The source tree is currently hosted in CVS. For details of how to obtain the source from CVS see http://developer.postgresql.org/docs/postgres/cvs.html.

The FAQ recommends that potential code contributors immerse themselves in the community's memory by downloading the code reading the email list. It also recommends that they read community email archives and use the code from the CVS repository (source tree) to participate in the community.

Retrieval and use of community activity

Retrieval and use of previous community activity is an important central component of working in these settings. Community members refer to the previous work of others by including snippets of text from the emails of other individuals in their own communication with the community. This activity has become a norm of participation in most online communities (Orlikowski and Yates 1994) and has evolved into a standard way of interacting online. Indeed in most email discussion, there will be at least one instance where participants will invoke the words of previous individuals, either in the same discussion, or from another discussion, as way to provide a pointer to all participants about the community's memory. The use of this activity is further enabled by most modern email systems which enable "reply to" functionality. Thus participants, using their local email archives, can reply to others by using extracts (often called quoting) from the email of others to make their point. This activity can be observed throughout the vignettes where participants continually quoted others in their responses during an email conversation. For example in the AV vignette, Lane reviewed Conway's coding style suggestions to O'Connor by first quoting the precise passages from Conway's email to O'Connor and then below it providing his critique. The discussion between Conway and O'Connor became part of the community memory on Lane's local e-mail archive and he was then able to, seven hours later, use it to contribute back to the

[58] Source tree refers to the CVS repository.

community and indicated O'Connor his concerns about Conway's suggestions. In the GC vignette, Lane defended his stewarding of the Red Hat requested feature by giving explicit url links to prior community discussions about the need and suitability of that feature.

Community members also retrieve and use the community's memory by accessing the software repository. The software repository contains information on every change made in the program by the community. Each software change is archived with information on who made the change, the date and time of that change, comments by the author regarding the change and the actual technical change itself. The RegEx vignette illustrates the enactment of this activity, Momjian, a core developer, initially did not know the author of the RegEx module. By searching the software repository he was able to identify Henry Spencer as the author. The repository also contained contact information about Spencer and thus Momjian was able to get in touch with him with his questions. Similarly, Lane used the software repository to help him in identify the author and the logic of the change responsible for his initial diagnosis of the reason for the RegEx performance degradation. Lane in communicating with the community, used the repository comment statement to show the link between his diagnosis and the change from six months before.

The strength of this practice, for retrieval of community discussions, is also its weakness as discussions that occur in the heat of the moment, may not be worth archiving or individuals may not be completely forthright in their discussions knowing that it is publicly archived. Participants lose a sense of privacy because all of their comments are archived for posterity and accessible to any one.

3.8.3: Collective Practice3 – Distributed Decision Making

There are two types of decisions made by community members. First, at the individual level, community members have to decide which tasks they are going to perform and what level of participation they want to have. Individuals make local decisions about task selection and that is discussed below. Second, the community has to

decide which of the code submissions are accepted and which ones are rejected. On the surface, the core community members have the ultimate decision rights to make these choices. However, these choices are not made in a vacuum or behind closed doors but occur via email broadcasts and are visible to the entire community. This means that the core group is continually justifying to each other and to the community their decisions. In addition, unlike the benevolent dictatorship model of Linux (Lee and Cole 2003), where one person has ultimate decisions rights on the code, in the PG case, every member of the core group of 11 individuals has commit authority and in theory can veto someone else's choices. I identified three activities that constitute the practice of distributed decision making in the PG community;1) incremental and local planning; 2) provisional settlements and 3) lazy consensus.

Incremental and local planning

The traditional approach to software development emphasizes significant upfront planning by product managers before any coding starts. There is a strong emphasis on requirements engineering, product and feature specification and task assignment to ensure that software engineering projects achieve their goals (Crowston 1997; Cusumano and Selby 1997; Cusumano and Selby 1996; Weber 2004) . In the corporate setting the best practice in software development emphasizes the significant role of upfront planning and clear and unambiguous decision making by the managers of the software team(Cusumano 2004; Cusumano and Selby 1995b). In the open source community setting, I observed more emphasis on incremental, decentralized and local planning. The particular dynamics of working in an all-volunteer[59] setting where there is no ex-ante information on the community's access to resources, commitment and effort from the participants means that the PG community did not and could not plan ahead of time. The only formal coordination device in the community that could serve a planning function is the TODO list. However, as discussed above, the TODO list was primarily a record keeping device for assigning ex-post credit to work completed. In an interview, Momjian noted:

[59] From the community's point of view – no one is paid by the community. Individuals are often paid to participate as seen in Lakhani and Wolf 2005.

> "At the start of a release cycle we have no clue as to what will be developed by the community. Everybody has their own personal wish list as to what needs to be done. But I have no information on what is on those wish lists. Sometimes we do have discussions about those wish list items but there is no expectation that those elements will be completed."

Lane expressed a similar view during the Red Hat GC module controversy in an email message to the entire community:

> "Personally I agree with the idea of stripping out whatever functionality isn't immediately necessary. **It seems to me we've generally been more successful by adding features in response to specific demand rather than inventing things we think might be needed."**

Instead of a significant investment in upfront planning, the core team and the community is relatively open to peripheral users and developers, gains access to information about technical needs as they arise in the actual use environment. Core members may have their own agendas that they need to fulfill due to their various sponsorship requirements or personal interests, but that does not mean that they attempt an ex-ante community consensus on what should be done or who should do it. As the AV and RegEx vignettes illustrate important features get developed or updated via a close connection with the use environment. Avoidance of global planning does not mean that local planning by individuals does not take place. We saw evidence of such planning in Daithankar's proposals for implementing AV and Lane's proposals for fixing the RegEx module. The local plans of one individuals as they are enacted in software code then trigger other individuals to create their own micro-plans for action. The AV vignette provides a good example by illustrating how iterated incremental and local planning by Daithankar, O'Connor and others created new functionality for PG.

Too much incremental and local planning may cause the community to drift from issue to issue without making significant progress in creating new functionality and innovation. The lack of global public plans may also raise the barrier for entry for new comers who may not be easily able to discern the priorities of the community and where

their contributions would be most welcome. It may also cause duplication in effort as participants may not have a full view of all the planned activities in the community.

Provisional Settlements

Complementary to incremental and local planning is the observed activity of core group members making provisional settlements (Girard and Stark 2002) of issues by deferring decisions until they are necessary. Girard and Stark (2002: 1947) in their study of internet design firms identified a collective decision making activity that "instead of *reaching an agreement*, they *reach a settlement*." In their empirical context, a project team achieved settlement of an issue by involving a superior or relevant outsider (e.g., client). The settlements allowed the relevant parties to continue with their work until they reached another moment of collective decision making. The settlements were also provisional in the sense that they were open to future reinterpretation.

In the case of PG, provisional settlements were achieved by deferring decisions on issues of design and timing of features. Decision to defer some aspects of a technological implementation are driven by an understanding that multiple ways of accomplishing the same task are possible. Core members also realize that they are under no obligation to accept code that does not perform as advertised via public testing. Another benefit of deferring decisions is that it enables the distributed community members to pursue multiple design options with the core group having the ultimate ability to chose the one that performs the best amongst all the trials.

The AV vignette shows how provisional settlements by deferring decision making work in the PG context. Recall that there were five design proposals that were floated by various individuals. These proposals matched the local knowledge, ability and interest of the contributors. Notably, Lane's proposal to create an AV system based on modifications to the core PG server module was not followed-up by any of the peripheral contributors. Lane also did not insist on any one particular approach and his view regarding the contribution from O'Connor confirms a deferring decisions perspective:

> I think a server-side solution is the way to go in the long run, but since one probably won't be available for 7.4, a contrib module seems like a reasonable stopgap offering. Contrib is our traditional refuge for "not ready for prime time" code, no? The fact that it's a client and doesn't touch server-side code actually works in its favor here --- there's nothing to rip out after we have a better answer

It is also interesting to note that besides Daithankar, Weilguni and O'Connor had also attempted to create solutions for the AV issue. Core team members, by not forcing designs or picking favorites upfront, allowed the natural experimentation within the community to take hold and the eventual accepted design to emerge. O'Connor in an email interview reflected that he had been stuck in his attempts to create a solution to the vacuuming problem several times, but that Daithankar's code gave him the necessary ideas to get it going:

> My interest in creating an autovacuum program for postgresql predated the C++ based pgavd work that Shridhar [Daithankar] had done. I had made a few attempts at starting pg_autovacuum by hacking backend code, each time I failed due to the complexity of the postgresql internals and my limited C programming skills. The one thing that I did take away from the work Shridhar [Daithankar] did was that the autovacuum process didn't need to be a backend project, it could (at least initially) be a libpq based client app which I was familiar with. That said, there is no common code from Shridhar's project and mine. The main reason for that is that if I was going to fold pg_autovacuum into the backend at some point, then it needed to be C based, not C++ as all backend postgresql code is in C.

Thus deferring decisions allowed peripheral members to experiment and enabled a local learning from the work of one member by another.

Another type of deferring decisions relates to initiating work only when it is absolutely necessary. In the RegEx vignette, the original need to get Spencer's new RegEx code was added to the TODO list by Momjian in 1998 based on one user complaint. However nobody took on the task of importing the code until early 2003 when Klaver demonstrated a use situation that was causing a significant performance drop in new PG code. In the approximate five years in between there was no observable

activity by any core or periphery developer to work on this item. Since there were not any public complaints about this issue within the wider developer and user community and no one had initiated any work on it there was no reason to pursue completing the item. The core team let the item stand on the TODO list but did not try to encourage others to complete it. Instead the decision to import the code was deferred until it was demonstrably proven, via Klaver, that it should occur.

Ward et al. (1995) have observed a similar delaying decisions tactic at Toyota. They called it the "Second Toyota Paradox" and defined it as follows: "The second paradox, in brief, delaying decisions, communicating "ambiguously," and pursuing excessive numbers of prototypes, enables Toyota to design cars faster and cheaper [pg 44]." They show that mangers in the Toyota Production System encourage the generation of multiple design sets for each critical component. Toyota issues very general specifications for the components and then allows the various internal engineering teams and external suppliers to pursue their own problem solving strategies until a "winning" design emerges. The winning design is not based on an ex-ante selection by elite automotive designers. Rather there is a process of constant comparison, based on actual "field" performance data, that allows the Toyota managers to make a choice. The winning choice emerges from a process of trial and error and continuous improvement instead of an upfront bet on the right design.

Deferred decision making may also be confused for or be given as an excuse for inaction or task laziness. There may be wasted effort in the various parallel trials in the community and ultimately peripheral contributors may get upset if their contributions are not accepted. Deferring action on an item because there is no vocal demand for it may cause the project to ignore important pain-points in the software. It may be the case that the user base may feel too intimidated to ask for a fix to features that are broken given that open source communities have a strong "do-it-yourself" attitude towards work.

Lazy Consensus

Community-based product development may imply an egalitarian type of decision making amongst both core and periphery members. The idealized view of such a community would mean that members, especially core members, would reach consensus on all decisions that confront the community. In practical terms I observed a leaning towards much more lazy consensus. By lazy consensus I mean that typically only one or two core members participated in the decision making process of committing the source code from peripheral developers to the global repository. Silence by core members implied tacit agreement and most of the time there was no explicit attempt to gain agreement from other core members unless the work impinged directly on an area of known interest of that developer. Lazy consensus thus lowers the threshold of participation and acceptance of code in the community. Instead of attempting full consensus on each and every issue, the core members in the PG community express voice as needed or else refrain from participating in the decision making process.

However, lazy consensus may not be ideal under circumstances when clear and direct leadership and accountability is needed. In contentious issues, developers may chose the exit option, that is, stop participating because they are uncertain about the outcome or the process, instead of voice, that is, persistently staying to participate and enact changes, resulting in an overall decrease in community effectiveness. The GC vignette showed that another downside of lazy consensus is that some times even core developers do not raise objections at the right time and place leading to significant community friction.

3.8.4: Individual Practice 1 – Choosing type and level of participation

Ultimately to get work done, community members have to undertake the tasks required to create a complex software product. Formal tasks in developing software include determining user needs, creating functional specifications and writing, integrating and testing software code. In formal organizations, these tasks are often pre-determined and assigned to employees via their unit and functional managers (Cusumano 2004). However, in the absence of formal authority structures and resource ownership,

community members have to determine for themselves the areas that they will work on, the specific tasks that will engage in, and their level of participation in those tasks. The two constituent activities of this practice are: 1) identifying tasks; and 2) doing the tasks. As the analysis below shows, task identification, engagement and participation are emergent and are dependent on community members selecting activities that match their own interests and needs with their skills and abilities.

Identifying Tasks

The first activity in task accomplishment is figuring out what things need to be done. There are four ways in which participants identify tasks that need to be accomplished; 1) meeting a use requirement; 2) meeting a community need; 3) conforming to technical architecture and standards and 4) following the project TODO list.

Tasks get identified when users report to the community their experience with the software. Thus Weilguni's complaints about poor performance with vacuuming and Klaver's concerns with decreased system functioning with RegEx informed the rest of the community of a potential area of work and tasks. Their complaints did not spell out the tasks that needed to be done, rather it directed the attention of community members to examine their claims and to determine potential tasks that needed to be accomplished. Tasks also get accomplished for specific corporate needs that may not be related to direct user experience. Nasser, as a representative of Red Hat, identified the need for GC functionality for their own version of the PG database and requested that rest of the community agree with both the need and their proposed solution. Nasser did not directly use the database for his own work but was representing a potential use case for Red Hat customers.

Reports of first and second hand use experience reports other community members to respond by giving feedback on the legitimacy of the issues being presented and also to determine if there is a community need that needs to be met. It is easy to understand why Lane took a deep interest in resolving Klaver's RegEx issues. Lane as

one of the core developers noticed that Klaver's identification and proof of a severe performance penalty in an important use domain had to be resolved in order for stable future releases. Thus he took it upon himself to diagnose the problem in collaboration with Klaver and to create a fix for the new version. Similarly, Ishii as a core member working in the same area as Klaver's problem wanted to ensure that the problems could also be resolved in the older versions of the software. Besides, core members, who have a direct interest in ensuring that the software performs as expected, periphery members also participate based on community need. Daithankar, in an email interview, reported that his motivation to respond to a community need reflected his view that it was easy:

> I joined postgresql project lists due to a project requirement on my day job. I stayed with it on/off for quite some while. I used to read most of the mailing list threads there. I still do though my direct contribution is hardly any.
>
> Oer [sic] the time, one thing became obvious to me that people didn't get how to run postgresql effectively. There are simple things that needs to be done with discipline but people just couldn't get the concepts.
> Vacuum was one problem that is not a real problem as much it was made out of it. It prevented postgresql from being 24x7. And large part of blame was with admins/app. developer.(or So I think)
> So I tossed idea of autovacuum on mailing list. Quite a few suggestions poured in, some of which were really helpful. Looking at the whole issue, I thought I could do it without much mucking with the core postgresql. So I said, why not try it?..:-) And it started..

A similar sentiment was expressed by O'Connor regarding his contribution to the AV module:

> [It was a]Nice to have [feature]. While do use pg_autovacuum on my servers, I don't think I befifit [sic] much from it, nightly vacuums were fine for my needs.
> More specifically, I wrote pg_autovacuum because I had been using PostgreSQL for a few years, and I wanted to give back to the project. I identfied autovac as something that PostgreSQL lacked, the community wanted, and was simple enough for me to work on.

While Daithankar and O'Connor did not have a direct need for the AV functionality, however, they both made the assessment that doing so would be a contribution to the community and also that it was something that was within their capabilities to execute.

Another source of task identification is conforming to technical and architecture standards. As contributors broadcast source code to the rest of the community, others examine it based on their own expertise and experience. This examination allows them to identify tasks that need to be done based on the degree of alignment of the technological architecture of the base software and the new code. In all three vignettes, the new software code submitted needed additional work so that it would fit with the existing code. Daithankar's original submission was written in C++ and it was used as a model by O'Connor to create the same functionality in C, the base language for PG. O'Connor's AV submission was critiqued by Conway for compatibility in coding and logic conventions within PG. Spencer's new RegEx code had to be adjusted to fit within the new constraints of PG and Red Hat's GC code had major modifications made to it in order to satisfy the technological architecture of the system.

Finally, the TODO list by the project serves as another source of task identification for the community. The TODO list is maintained by Momjian and he has the decision rights to add items to it. A naïve view of the TODO list would be to consider it as a primary coordination and task identification device for the community. However this normative explanation does not reflect the actual workings of the community. Instead the TODO list is viewed as one potential source of task identification within the community. In interviews core members reported that the TODO list was simply Momjian's view of what needed to be done in the project and did not reflect consensus within the community or even a commitment to get things done.

Doing Tasks

Once a task has been identified it also needs to be accomplished. On one level task identification is also a task within the community. A primary task of the community is to determine what needs to be done and often identification and accomplishment go

hand-in-hand. Thus, Conway's critique of O'Connor's code was both a task identification and task accomplishment – done in email. He identified the changes that needed to be done and also how they should be completed.

The vignettes show that contributions from both core and peripheral members range in intensity and effort. Sproull, Conley, & Moon (2005) have postulated that micro-contributions, contribution of small textual units (via email) requiring limited time and attention, are the basic building block of online communities. In the vignettes, micro-contributions, such as, reporting a bug, sharing an idea, requesting a feature, participating in a discussion, doing translations, testing software changes, and contributing code were observed. Micro-contributions allow peripheral members to problem solve with the core and other community members without the need for significant upfront investment in time and effort. Micro-contribution does not imply micro-impact. Weilguni's idea of using the "stats-collector" as the foundation for the AV module arrived in the form of a micro-contribution. However, it was a critical component of all the work done in the future versions of the AV module. The small increments of information also create a significantly smaller cognitive burden on the participants, allowing them to make an assessment of the shared information and its implication for the particular technology.

Furthermore, since the micro contributions are relatively small and public, mid-stream course corrections and changes in directions are more easily accomplished as compared to making changes to much larger contributions. Micro-contributions from the range of participants involved in a particular feature accumulate towards the collective output of the community. Micro-contributions, however, do not occur in a vacuum, but are instead directly tied to an evolving major code contribution. The significant code writing effort of Daithankar, O'Connor, Browne, Spencer, Lane Ahmed and Eisentraut in the three vignettes, made it possible for community members to engage in other tasks as code review, algorithm design, code use and bug finding and fixing.

Community members' ability to choose the type and level of participation does away with a lot of the bureaucratic means of administration observed in software development projects within firms (Cusumano 2004; Cusumano and Selby 1995a; Cusumano et al. 2003). Contributors are exposed to an ongoing stream of activity by other community members and decide if, when, where and how to participate. The risk of this practice, however, is that some tasks may not get done. For example, software from open source communities has often been criticized for not being overly user friendly, thus tasks that relate to making the software accessible to less sophisticated users are often neglected or never undertaken by the community members. These tasks are often viewed as being boring and not sufficiently challenging and will only get accomplished if a community member has either a commercial or personal need for them. The RegEx vignette did illustrate that an item can languish for quite some time on the TODO list. Thus there is a possibility that important tasks may be neglected by over reliance on this community participants choosing their type and level of participation.

3.8.5: Individual Practice 2 - Using The Community's Output

The collective practice of building and using the community's memory is directly related to individuals using the community's output. Community members take the collective outputs, that is, the software, and then integrate it within their own local system and deploy the software in the use-environment. From a software development point of view, integration and use are means of finding problems within the design and construction of the software.

Software integration is defined as an assembly of parts both old and new creating an updated version of the software. Systems integration, before actual product release, is a fundamental engineering design activity as it initiates "interference finding" between various parts and helps to determine unforeseen interactions (Marples 1961). In the case of software, integration is important because as the various modules developed by various programmers are integrated, bugs and problems that were not visible in individual components may become evident as latent interdependencies (Jorgensen 2005).

The use of software in actual production environments may also reveal issues with it. Users stress the software by using it unanticipated ways and in unanticipated environments revealing new sources of dependencies and lack of functionality within the software. Most often software developers will have a phase of "beta" testing where external users will put the software through simulated or near production environments.

In the PG community software integration and use are continuous and decentralized. Integration is not centralized and there is no one person in charge of the software build. Similarly although there is a formal "beta" testing phase within the software release process for PG, many participants were deploying pre-beta, that is using, software in actual use environments. This individual use of the community's output helps the participants to identify and rectify issues and interdependencies earlier in the process and to resolve problems as they arise instead of waiting till much later stage. I describe two activities: 1) integrating latest code and 2) using latest code that constitute this practice.

Integrating latest code

The community software resides in both global (community-wide public servers) and local (participant's own private computers) repositories. As new software gets developed by peripheral community members, it is first sent via a broadcast email to the entire community. Other core and peripheral members then examine the software for obvious flaws and then integrate the code with their own local repositories. If the local integration is passed then a core member will commit the code to the global repository and make the changes permanent. The global integration then allows for a much wider set of individuals to try out the code and observe its effects. This early code examination and integration helps to find errors in the new code that may have been missed by the original author and also exposes the code to a diverse technological and intellectual base for evaluation and testing. In the RegEx vignette, Klaver integrated both Lane's new code and Ishii's changes immediately with his system to confirm their utility. Similarly O'Connor's code was first locally integrated by Conway resulting in a request for

changes and then the changes were committed to the repository by Momjian. The issues with Red Hat's GC contribution were discovered when Herrera did an integration of new code in a previously unrelated section, internationalizations, on his local machine. This occurred three months after the global integration into the source code repository by Lane.

Using latest code

After local and/or global integration, many development community participants start to use the code directly in production environments, before the beta test release or the final release. This activity provides both need information about the new features being developed and an indication of potential problems as a variety of users test out the new features under heterogeneous contexts. As table 3.9 shows, 39% of the code submitted by the periphery is actually in their local production environment to start with. Some peripheral developers, as they encounter problems will first try to resolve them in their local environment before passing on the solutions to the rest of the community. In other cases, peripheral participants will use the latest code from others in their own production environment to gain access to critical features that they need prior to a general stable release. In the AV vignette, O'Connor indicated that his code submission was already working in his production environment at the time of his code submission. During the discussion of the AV feature, another peripheral user also endorsed the general usefulness of the feature and informed the community that he had been running Daithankar's old AV code for sometime in his production environment. Improvements to O'Connor's code from Brown also arose because of his usage of that feature in an actual use setting which led him to discover the abnormal behavior. Similarly Klaver's identification of the RegEx bug occurred because he was using the latest globally integrated source code from PG repositories in an actual production setting.

There are three potential risks to the community with this practice. First, the lack of a centralized integration role may mean that some changes and additions that are not appropriate may slip by, simply because somebody else did not seek to systematically test the changes that were committed. This was certainly the case in the GC module where

the community trusted Lane's judgment and let him make the changes as needed without extensive evaluation and testing by others. Second, there is a risk that incremental integration and continuous use may place the community in a situation where only incremental improvements are being made in the system and radical changes are being eschewed. Third, the use of pre-beta code in production systems may cause unexpected behavior which may result in an unanticipated loss of data or systems failure. Thus those trying out brand new changes run some risk in using pre-release software in a production setting.

3.8.6: Individual Practice 3 - Coordinating Action and Building Trust Through Evidence

The core-periphery interaction in the PG community is essentially one of strangers interacting with one another over a relatively short period of time using very lean media. In the absence of pre-existing ties, managerial authority, and a history of joint experience, the PG community relies on individuals to provide concrete evidence of their claims and requests. Evidence can be in the form of software code, output of an error log, a detailed description of the situation that caused an unanticipated breakdown in the software, the logical reasoning behind a particular request or simply reporting the presence of similar issues in another use environment. Providing evidence serves both social, coordination and technical problem solving requirements of the community. From a social perspective, peripheral participants can gain quick legitimacy within the community and especially with the core group by giving evidence of the veracity of their problems and solutions. Core group members provide evidence to demonstrate that they are acting in the best interests of the community by being inclusive and rational about the choices they are making. The evidence provided also helps to coordinate the response of the distributed community members by letting them assess if they need to respond and how. In addition, it gives guidance as to the location of the appropriate area of technical interest in a complex technological artifact. The evidence also helps the community to assess the technical merits of a problem and/or solution by giving them a concrete and "objective" means of assessing the claims. This practice consist of two complementary activities; providing evidence of claims and seeking and testing evidence of claims.

Evidence Provisioning

Community members who initiate work by posing problems or issues or those who respond with potential solutions also typically provide evidence of their claims to the entire community. Problems can be transferred from the individual to the collective by giving compelling evidence of their impact. In the RegEx vignette, Klaver's posting the output of his database query provided concrete evidence of a significant drop in performance (a factor of 150) for a soon-to-be-released PG software version. This was enough to convince two core group members and six other peripheral members to participate in problem solving. Similarly, in the AV vignette, Weilguni's articulation of his unhappiness with the current vacuuming functionality along with a description of its negative impact on his usage of the software mobilized the community. Weilguni's evidence was further bolstered by other community members further claiming that vacuuming was a problem for them as well. Nasser's request to have the new GC interface code be part of the PG system was based on an articulation of a general need that would be of use to many others in the community.

Solution providers also need to give evidence of their claims about the usefulness of their submissions. The most dramatic example of giving such evidence was O'Connor broadcast of a graph demonstrating how the inclusion of his AV code resulted in a significant increase in software performance (transactions per second supported) as compared to similar software without his AV code. O'Connor's comparison of the old with the new along with the supporting graphs and source code built confidence in the community that he had a viable solution to the original problem. Similarly Ishii and Lane, both core members, showed the effectiveness of their solutions to the RegEx problem by comparing performance of their code on PG-specific regression tests for the software. They computed time in milliseconds and showed how fixes to the code improved performance. In this case, stating that the software worked and "compiled cleanly" was not enough. Rather, since Klaver's original complaint was a drop in performance, giving evidence of performance results was deemed important by the core developers.

Seeking and testing evidence of claims

In addition to providing evidence, community members also actively seek evidence of claims to aid in problem diagnosis, issue identification and clarification, and for specific help with unique circumstances. The request for more evidence can arise from members who have problems and proposed solutions. In the RegEx vignette, Klaver and Lane executed four cycles of seeking evidence and providing evidence. Each iteration helped Lane get closer to the proper problem diagnosis. Since Klaver was in a unique user setting and experiencing an unexpected failure it was necessary for Lane to involve him in the problem diagnosis process. Seeking evidence then becomes a way for the distributed members to participate in the problem solving process. The back and forth between Klaver and Lane demonstrated to the entire community that the problem was on its way to being characterized and the first diagnosis was not necessarily correct.

On the other hand in the GC vignette, Momjian complained to Nasser that he had not provided sufficient evidence for his particular need for the GC software addition. Momjian noted that Nasser was absent during the second review process and had to be prodded by Lane to participate. Momjian viewed this as an affront to the community process because the person behind the solution being implemented was not available to respond to community questions. Nasser felt that the community process was over when the code was committed to the repository in July 2003. Instead, Momjian argued that, at least before a final version is released, all members who have submitted code need to be able to provide evidence for their claims.

Those creating solutions to problems can also request evidence in the form of specific testing regarding their software code. Both Lane and Ishii in creating their changes to the RegEx module, asked Klaver to test their solutions to give evidence to the community (and themselves) that their solutions performed as expected. Ishii also tested Lane's code and gave performance evidence that it worked as advertised. In other cases, claims can be tested directly in code by others. In the AV vignette, Daithankar's initial proposal for a script-based solution was deemed infeasible by Weilguni. However, Taylor responded back to Weilguni by linking to an actual script that did what Weilguni

had initially thought to be not acceptable. Thus Taylor created new evidence that debunked Weilguni's claims.

Evidence provisioning forms an important individual practice of participation in the community. However a concern with this practice may be that the standards of evidence might be too high for peripheral members who are not experts with the code. They may not have the experience with the software to generate the correct types of evidence for their problems. Also there may concerns with disputes over interpretation of evidence. A severe breakdown for one person may be a trivial situation for another and "objective" numbers may not be enough to garner mobilization of others to one's problems.

3.9: Discussion

The motivation for my study has been to explain a central empirical observation in F/OSS communities that shows the inclusion of many thousands of individuals in highly complex product development tasks. Most casual observers of such a phenomenon would predict a breakdown in the social order and not expect much in terms of community accomplishment. Similarly the observations, recommendations and theories of scholars of innovation and product development activity inside of formal organizations have viewed with skepticism the ability of F/OSS communities to innovate, solve tough technical problems and create, competitive products.

In my study I wanted to understand the value of the periphery in the community problem solving effort and to develop a grounded theory of how the core and peripheral members in the community work together to solve technical problems. Andrew Morton's quotation about code writing participation in the Linux kernel community indicated that very few individuals contributed actual code to the development effort, in effect, begging the question why those few individuals allow such an open and transparent development process along with the participation of thousands of individuals. Yet the core participants persist in keeping their activities open so that others may participate with them.

Momjian, an original and founding core member of the PG community shared this view with me on an internet relay chat:

> "A few core folks might to [do] the majority of the coding, but we are pretty lost without the support group around us. They give us ideas for features, report bugs, test stuff, and act as a sounding board for ideas, plus 1-2 folks usually show up to give major ideas on each major feature, it is usually a different 1-2 guys, but they are deep thinkers in that area and provide invaluable input. Doing the coding is only 1/2 the job, in fact perhaps less than half. The design of how to do something, and how the user should interact with it, and the testing are keys to our group dynamics."

My quantitative analysis has shown the primacy of the periphery in the PG community. I find that the periphery is essential for the core in four ways: they provide use-information about the product, they provide unique pre-made solutions; they provide solution-related information; and they themselves are a resource for the community. Practically, peripheral members initiate a majority of code writing episodes; they write about half the code; they participate in collective problem solving with other code writers; they test and evaluate newly written code; and they are a source of new core members. More importantly they are the primary source of *new needs and solutions* adopted by the community. The ongoing interactions among core and periphery interactions are critical to the community's ability to produce working software. Organizational research on the nature of knowledge has shown that it is both sticky and leaky (Brown and Duguid 2001). Existing research has shown that when knowledge is totally sticky and distributed, functionally novel innovation – involving novel need information - will occur primarily in the periphery (Riggs and von Hippel 1994; Tyre and von Hippel 1997; von Hippel 2005). When knowledge is totally non-sticky, problem-solving may occur only in the core. However another perspective on knowledge is to emphasize that it is not an object but rather "knowing in practice" (Orlikowski 2002). Thus core and periphery interactions are not about just about transferring knowledge or the degree of stickiness of knowledge, but instead they represent the practice of sharing "know how" amongst the participants.

My findings are consistent with extant research which highlights the importance of external information for core team problem solving (e.g.: Allen 1977), the advantages of "intellectually heterogeneous groups" (Leonard and Sensiper 1998) for innovation, and the importance of the technical core for task accomplishment (Thompson 1967). However, I show that the relatively open interaction architecture changes the organizing logic such that peripheral community members can play an important role in complex technology development.

In these core-periphery communities, instead of core gatekeepers (Allen 1977) seeking relevant information from the outside, peripheral members, on their own accord, bring in both problem need and solution information. Furthermore, peripheral members work hand-in-hand with core members to jointly solve problems and contribute pre-made technical solutions. In keeping with Granovetter's (1973) strength of weak ties theory, I show that the periphery provides unique knowledge not available to the core. However, tie activation is conducted by the periphery instead of the core. My findings also demonstrate the importance of the technical core to the collective problem solving effort. However, unlike Thompson's (1967) recommendations of buffering and sealing the technical core from the environment, I show that the core's embrace of the use environment via its interactions with the periphery is critical to community's success.

Along with my quantitative analysis I have attempted to develop a grounded theory of the social practices that enable distributed problem solving in a community. The collective practice of work broadcasting, building and using community memory, and distributed decision making enable public participation and transparency in the community and are critical for work accomplishment. It is hard to imagine organizations where a significant majority of work activity occurs in public settings and all work-related activity is broadcast to any *interested* party. The cognitive burden of absorbing information from over 100 community-related e-mails per day and also participating in

relevant discussions can appear to be overwhelming. However, this is a sine qua non[60] of F/OSS communities. Public participation enables interested members to be updated about relevant activities and issues across the distributed community. Members are immersed in a continuous stream of technical, task-related discussion and can choose to selectively participate or initiate new dialog. Public participation allows for the possibility of peripheral and hidden (lurking) members to contribute to relevant discussions and opens up the potential that diverse sources of information may be accessible to the project. Public participation also ensures that community members who have alternative perspectives on issues can at least voice their concerns to all other members.

An important derivative of public participation is the transparency provided of the work process of the community. Lave and Wenger have noted that transparency of process and technology-use-in-practice are important considerations towards successful legitimate peripheral participation. New apprentices need to be able to "see," observe and participate in all aspects of a community's practice as part of their journey towards full membership. When transparency is blocked or is difficult, learning is inhibited. Similarly the technology in use by the community needs to be transparent so that newcomers can learn to adapt and use the technology for their practices and to enhance learning, instead of treating them as black boxes. Transparency is manifest in both community discussions (on e-mail lists) and in the technological objects created. Members can observe (and participate if needed) in all of the design, problem-solving, code creation, and decision making processes of the community. Transparency in process and code builds confidence amongst participants and enables them to collaborate. Concerns for opportunism and/or subversion from others can be tempered by transparency enabling a community monitoring system, where the ability to call into question decisions and choices is available to both core and peripheral members.

[60] This does not preclude private discussions amongst members. However my interviews and data analysis show that this is not a norm for the community and many private discussions do spill over onto the public side.

Another outcome of the collective practices is the sense of *collective ownership* of the developed software code and the ideas around the code. Once code is submitted to the community or is imported from another community, the ownership of the code shifts from the individual to the collective. This allows other contributors to propose and/or create further modifications to the donated software code without explicitly asking for permission from the originator. It also ensures that the responsibility for the viability of the software code – its quality, maintainability and updates – becomes the responsibility of the community instead of the individual. Additionally even after a particular contributor has lost interest in the community's effort – others can take over and evolve the code submitted by the absent members[61]. Collective ownership also allows community members to leverage each other's contributions and build on top of each other's work instead of doing a de-novo start.

The collective ownership of ideas means that individuals are free to elaborate and further expand ideas proposed in the community. "Co-construction" has been proposed by cognitive and learning scholars as mechanism for effective group problem solving (Hausmann 2003). Hausman's (2003: 10) review of the extant literature on co-construction allowed him to proposed the following synthesized definition of co-construction: "(*a*) the ideas are not attributable to any one person, but are distributed over two (or more) speakers, (*b*) ideas are built up over multiple turns, (*c*) new information is generated that neither knew before the interaction, and (*d*) the co-constructed ideas are unlikely to have been generated by either individual alone." In the distributed problem solving community, the collective practices of work broadcasting, building and using community memory, and distributed decision making enable co-construction to take place.

At the individual practice level, members also experience relatively *low participation costs* for contributing to F/OSS communities. On the social side, the practice of task self selection and accomplishment means that members do not have to

[61] This does not mean that there is no "moral" ownership of the code, that is, if one contributor is public about working in a particular area of the source code, then making changes in the same area, without publicly alerting that this is being done – would be considered a breach of community ethics.

ask permission to work on any particular part of the software code. Often, members attempt code changes privately and then reveal their results to the development e-mail list only if they were successful. Failure[62] does not have to be publicly revealed thus lowering the social costs of community participation. If members ***had*** to make an ex-ante revelation of intent or ask for permission before hand – then a lot fewer attempts would be expected as potential contributors would have their public reputations at stake and would need to be confident in their ability to deliver on their claims. Even when contributors publicly announce their intentions to work on something, there are very low community expectations about timing or even actual completion of effort. Community members are aware of the voluntary nature of their participation and are very forgiving with regards to delays and partial or non-completion of work. Some of the negative consequence of this include: lost lessons about failures, if people are not discussion failed attempts in public; inefficiency in many people trying the same thing; and the possibility that work initially done in private may not get accepted by the core team.

The cost of participation from a task point of view can also be low for a contributor. The threshold of any quanta of contribution can be quite low. In F/OSS communities, micro-contributions (Sproull, Conley and Moon 2005: 144), in the forms of reporting a bug, sharing an idea, requesting a feature, participating in a discussion, doing translations, testing software changes, contributing code are the essential currency of generalized exchange (Ekeh 1974) and participation. Membership and entry in the community can take many forms and does not require a significant up-front investment for initial participation. As Sproull et al. note (2005: 144): "Not only is it easy to make a helpful contribution, it is also easy to control the extent of further involvement. In the offline world, a person may hesitate to offer help for fear that helpful response will lead to further demands on one's time or emotional energy. In the online world, a person offering help may feel in complete control of how much further involvement will ensue; he or she can simply ignore further requests." In contrast, formal organizations are not structured to accept micro-contributions. Individuals are formally assigned to teams and projects and the possibility of unsolicited participation in the affairs of another team

[62] There are many sources of failure: lack of time, lack of interest, lack of knowledge, and limited skill.

would is quite remote and may be viewed with hostility. The receiving party may consider it an intrusion of their workspace and the managers of the contributor may be concerned that his/her direct reports are contributing to other (extraneous) projects.

Another dimension of an individual choosing the type and level of participation is that the coordination happens informally. Communities do not attempt to (re)create bureaucratic modes of organizing as typified in software engineering projects inside of firms and other formal organizations (Adler 2003). Synchronicity of action in the community is not primarily determined by plans, schedules and technological road maps or individuals undertaking these activities for ***other*** community members. Instead, individual members choose the task that they will undertake, how it will be initially accomplished, and when it will get done. The essential work and task structures are limited to proposing ideas, writing, testing, and using code, and commenting in discussions. Coordination occurs primarily via mutual adjustment and by "organizing work by adaptation" (Hutchins 1991). Adaptation and adjustment are facilitated by the collective practices discussed and in particular by work broadcasting. Similar to findings on communities of practice, where learning is not separated from working (Brown and Duguid 1991), coordination occurs while work is completed and revealed to others and is not a separate form of activity or specialization within the communities.

The development of trust has been found to be important for success in distributed and virtual teams (Jarvenpaa and Leidner 1999; Powell, Piccoli and Ives 2004; Sarker, Lau and Sahay 2001). Nearly all definitions of trust imply that one party, the truster, must willingly place themselves in a position of vulnerability to or risk from another party, the trustee (Gallivan 2001: 280). Traditional theories of trust development note that trust is built through shared social norms, repeated (face-to-face) interactions, shared experiences, anticipation of future interactions, interpersonal relationship development, and emotional support (Jarvenpaa and Leidner 1999). F/OSS communities with their ever changing and continuously flowing peripheral membership could be considered highly vulnerable to trust issues and an unlikely fertile ground for its development. Some scholars have eschewed the notion that trust, as traditionally defined and developed, is

necessary for effective collective action. Thus, Das & Teng (1998) have argued that control is an alternative mechanism to ensure cooperative behavior, and Meyerson, Weick, & Kramer (1996) have suggested that in temporary teams, instead of trust being developed, it is imported from other contexts and then sustained via visible action – resulting in what they term "swift trust."

I propose that another alternative to trust development for distributed core and periphery problem solving is the individual practice of coordinating action and building trust by providing evidence. For F/OSS communities, the continual public submission of software code helps to build and maintain trust amongst all parties. Software development provides for the verification of completed work, that is, "Does it do what the contributor claims?" via "objective" measures. *"Truth in code"* is achieved when others apply submitted changes to their personal copies of the software repository and then verify if the software works. Verification works at both the basic level, that is, does the changed system actually work as advertised, and at an advanced level, that is, is there any degradation in performance (speed, load handling etc) due to the changes. Software development kits provide standard tools that produce objective measures, about software changes, which can be shared with others and serve as a basis for further development. Those submitting problems are also able to utilize "objective measures" to make their claims. Contributors raising issues provide (or are asked to provide) detailed information on how specific problems were created along with associated computer-generated log files and use scenarios which allow others to recreate the exact issues on their own local machines.[63] The provision of evidence along with a capacity to share and assess (Carlile 2004) the evidence serves as substitute for traditional forms of trust development.

Finally I note that the practices for distributed problem solving are completely intertwined with the technology of production, that is, email lists, repositories and program compilers, and the technological artifacts created by community members, that is, a working software system is produced by people in the community. Knorr-Cetina

[63] In the PostgreSQL project – core developers have been observed asking for remote access to problem systems so that a better diagnosis can be made or a fix tested.

(2001) has critiqued currently defined views of practice as being rule-based routines or embodied skills with an emphasis on human dispositions and habits and connotation of procedural routines. Lave and Wenger (Lave and Wenger 1991) stress the reproductive and historical dimensions of practice and ignore the improvisational and unfolding and productive aspects of practice in their description of communities of practice (Osterlund and Carlile 2003). Knorr-Cetina offers an "objectual" orientation towards practice in which work around and with epistemic or knowledge objects, characterized by their lack of completeness (for example software is an epistemic object because it is always under construction with regular new releases), cause practices to continuously unfold and co-evolve with the epistemic objects. Thus epistemic practices imply a relational view of actors with themselves and with their use and construction of technology. Practice is not just about routines and skillful performance of historic activity but when employed around "incomplete" epistemic objects are consciously being formed and updated. It is this understanding that undergrids my articulation of the practices for distributed problem solving. Thus the current articulation of these practices should be not be viewed as complete or finished. Rather, I leave open the possibility that changes in technology, changes in the types of people participating, and changes in the use of the technology in may cause the practices to evolve. Thus in my empirical context, the dynamic nature of the software industry and the high degree of corporate interest in open source software communities may lead to modification of work practice[64].

3.10: Implications and Conclusion

For the last nine years, the PG community has been producing, supporting and updating a free, robust and competitive database management system. Over three thousand individuals have participated in its development by interacting virtually on email lists with very few occasions of face-to-face contact. This type of large scale collaboration and creation is now becoming a routine way of developing complex software (Weber 2004) with thousands of projects and hundreds of thousands of

[64] As I was writing this section of the dissertation, Sun Microsystems announced that they will be including the PG database as a standard component in all the future releases of the Sun Solaris operating system. This has brought in a new influx of contributors and I anticipate that the practices should evolve to support this change.

individuals "volunteering" their time and effort in creating public goods. My study has tried to build towards an initial understanding of how these communities "work," how they solve problems and achieve coherence and social order without relying on the scaffolds present in formal organizations. I now turn to a discussion of the implications of my findings for innovation and product development and organizations.

3.10.1: Implications for Innovation and Product Development

Innovation, is a process characterized by its requirement for new knowledge, new combinations of existing knowledge, and non-standard procedures. Problem solving for innovation thus requires that solutions to novel problems be found (Clark 1985). External sources of knowledge have been found to be critical to the innovation process (Cohen and Levinthal 1989; Cohen and Levinthal 1990). March and Simon (1958: 188) first posited that, at the organizational level, most innovations result from "borrowing rather than invention." They contended that borrowing was either a function of imitation or by importing new personnel into the organization. Regardless of the mechanism for borrowing, they postulated that the rate and type of innovation was based on the communication structures of the organization with a special emphasis on external connections.

I have shown that the distinction between external and internal is very blurry in F/OSS communities and possibly not even appropriate. The core – periphery concept provides a decision rights perspective on who has the final say on the ultimate shape of the joint output. However, these decision rights are kept in check by a public and software development process. In a public setting the external-internal distinction collapses with the boundaries of the community made relatively permeable via the practices of work broadcasting and building and using community memory. I observed core developers reaching "outside" the community to appropriate software for use inside the community, and I observed peripheral participants impinging on the ongoing work of the core by raising issues, offering solutions and developing technology for their own particular use.

Research inspired by the social networks tradition has shown that weak ties are critical for obtaining novel innovation (i.e. search), and the literature on innovation and product development has concluded that tight coupling and extensive communication between teams, departments and business units is needed for effective product development (i.e. transfer). Hansen (1999) has shown that, within a multi-unit organization, weak ties help to identify novel sources of knowledge but strong ties are required for effective transfer of non-codified knowledge. In either case, search and transfer are being conducted by the focal innovating team. He has identified this as the "search-transfer" paradox in innovation and new product development.

I contend that distributed problem solving communities, as exemplified by F/OSS communities, invert the received logic of organizing for innovation and product development. The core group in the F/OSS community does not undertake search and transfer effort for product development. Instead through open and willing engagement with the peripheral members and joint enactment of distributed problem solving practices, "search and transfer" gets transformed into interactive and distributed problem solving. The responsibility for search and transfer are no longer with a small group of experts, divining knowledge about technology, customer and market requirements. The core group does not forage outside for ideas and solutions in the external environment. Instead through their interactions with the peripheral members, the distinction of an external environment becomes less appropriate. The classic solutions to bringing external knowledge inside via gatekeepers (Allen 1977), boundary spanners (Tushman 1977), scouts (Ancona and Caldwell 1992) and building an absorptive capacity capability (Cohen and Levinthal 1990) are not relevant in this setting. Effective collaboration with the periphery appears to be more important for innovation.

3.10.2: Implications for Organization Theory

Thompson's (1967) classical work "Organizations in Action," has become the canonical reference for scholars studying organizations and their attempts at coordination and work accomplishment (Kamps and Polos 1999; Scott 1998). In the book, Thompson argues that the environment is a key source of uncertainty for organizations and that

much of organizational action can be explained by the need to reduce this uncertainty (1967: 10). Thompson proposes that an organization is differentiated into three components; production, managerial and institutional. He specifically argues that the production component – what he calls "the technical core" – needs to be sealed from the environment. If sealing is not possible he recommends buffering the core with mediating specialized units that transmit the signal from the environment into the technical core. The technical core in Thompson's view was where the organization produced its main outputs. He used examples like the factory production floor, a bank's primary role of matching deposits to loans and an emergency room in an hospital as illustrative of a complex organization's technical core. These examples illustrate that Thompson's view of the technical core assumes a stability in technology and knowledge use and does not anticipate an innovation function for the core. The technical core's mission is to faithfully reproduce the organization's outputs, and thus it makes sense to keep it isolated from the environment once the organization has evolved its routines.

However, if the goal is to innovate, create new knowledge and new objects then it may not make sense to follow Thompson's recommendations. The PG example shows that exposing the technical core to the environment can also lead to productive outcomes. Buffering may not be necessary or possible in this form of complex organization. More generally, the open source model of organizing shows that an explicit mission to create new products and technology inside of the organization, a task imbued with uncertainty, means that organizations and their actions should not be conceived in terms of uncertainty reduction. Rather, knowledge generation as the fundamental organizing principle of organizations may imply embracing uncertainty and developing practices that enable the organization to perform given high degrees of uncertainty.

The relative success of distributed problem solving communities raises interesting questions for our understandings about the operations of formal organizations. F/OSS communities, as currently structured, are free from the formal constraints faced by firms and formal organizations. These communities do not need to sell anything, make a profit, and exclude competitors. While many communities have a strong orientation towards

supporting their users, they are under no obligation to follow through on any promises of support or development (e.g., Lakhani and von Hippel 2003). The common answer for problem statements is an invitation to "do it your self." This freedom allows the community to only deliver in on a small set of technically focused tasks. However: firms and mangers inside of firms do have significant resource advantages; they can hire and fire people, set concrete goals, offer significant material rewards for goal accomplishment and plan and achieve objectives. Direct technical evaluations of similar products produced by firms and F/OSS communities have shown no significant difference in performance, maintainability and quality (Samolados et al. 2004). The ability of two very different forms of organizing with two very different resource models to produce similar complex outcomes raises has interesting implications for our theories about organizing.

Foss'(2003) study of Oticon's "spaghetti organization" structure provides us with insights into the obstacles that may be present in formal organizations adopting or attempting to transform into more informal and fluid ways of working and organizing. Oticon, a Danish maker of hearing-aid devices with revenues of 500M DKR, in 1991, restructured itself from a functional department based organization to a project-based organization with a two layer hierarchy, consisting of employees in self-organizing project teams and a 10 member management team. Employees were free to join any project that they wanted and project managers could recruit any employee to participate with them in the project. Wage negotiations and bonuses were also decentralized and were determined between employees and project leaders. Employees could also join as many projects that they wanted. The only formal requirement was that projects be initially approved by the management team and were subject to a progress review every three months by the management team. The results from this shift were dramatic, product development time was cut by 50%, 15 new products were introduced in a space of three years when the previous years had not yielded one new introduction; and the return on equity for the firm jumped from -1.5% to 30% level. However, even with such success the Oticon organization in late 1990's shifted to a more traditional matrix form of organization.

Foss attributes this shift back to a more traditional organizational form as a problem of incentives. In particular, managerial meddling into the delegated decision rights of the project teams caused a strong loss of motivation in the employees and pushed the organization back towards more structure. As Foss notes (2003: 337):

> The fact that the Projects and Products Committee could veto a project ex ante suggests that it was the real holder of power in Oticon. Frequent intervention on the part of the Committee ex post project approval confirms this. Thus, it became increasingly clear that the Committee could at any time halt, change, or even close projects.....However, the way in which the Projects and Products Committee exercised their ultimate decision rights is more akin to reneging on a contract, perhaps even to performing a "hold-up" (Williamson 1996). Thus, the Committee effectively reneged in implicit contracts with the projects as the efforts of project became, in the eyes of the Committee, superfluous (e.g., because of new technological developments), moved in unforeseen directions, or were revealed to have been founded on ill-conceived ideas.

This type of selective intervention by the management team was not welcomed by the employees and interestingly, it was the employees who requested a more traditional approach where lines of authority were clearly delineated and understood. In contrast, the PG community thrives in a spaghetti- like organizational structure. Selective intervention by management is not possible because there is no "management", that is a separate authority structure that approves work of others, and community participants are volunteering their time and effort. The core team does have the right to reject any submission that they do not like and could be conceived of having the same rights as the management in the Oticon case. However this right, to reject a submission is tempered by two factors; 1) the rejection occurs in a public setting and it requires a rational technically-based explanation and 2) even if rejected the contributor can still, although with more difficulty, implement the changes on his/her own private version of the system and thus not face a complete loss of effort.

Foss' work raises the question if self-organizing behavior as observed in the spaghetti organization and in open source communities are compatible with formal,

managerially driven organizations. While many scholars have shown significant emergent activity and behavior inside of firms (Burgelman 1983; Burgelman 1994; Mintzberg 1978; Mintzberg and McHugh 1985; Noda and Bower 1996), the role of the visible hand of management and hierarchy (Chandler 1977) cannot be minimized. It is an open question whether the distributed problem solving practices in communities could be adapted by firms.

Even more interesting are attempts by firms to work with communities e.g., IBM and Sun working with the Linux, Apache and PG communities. And communities establishing firms e.g., Mozilla Foundation setting up Mozilla Corporation. In these situations, as members of firms and communities traverse organizational boundaries, the potential for innovation and change in the distributed problem solving practices are high along with the commensurate risk of failure if there is no adaptation.

References

Adamson, R. E. 1952. "Functional fixedness as related to problem solving: a repetition of three experiments." *Journal of Experimental Psychology* 44:288-291.

Adler, Paul S. 2003. "Practice and process: The socialization of software development." in *Academy of Management Best Papers*. Seattle, WA.

Allen, Thomas, J. 1977. *Managing the flow of technology*. Cambridge, MA: MIT Press.

Ancona, Deborah G., and David F. Caldwell. 1992. "Bridging the Boundary: External Activity and Performance in Organizational Teams." *Administrative Science Quarterly* 37:634-665.

Barley, Stephen, and Gideon Kunda. 2001. "Bringing Work Back In." *Organization Science* 12:76-95.

Barnes, Barry. 2001. "Practices as collective action." Pp. 17-28 in *The practice turn in contemporary theory*, edited by Theodore R Schatzki, Karin Knorr Cetina, and Eike Von Savigny. New York, NY: Routledge.

Baron, Jonathan. 1988. *Thinking and deciding*. New York: Cambridge University Press.

Birch, H. G., and H. S. Rabinowitz. 1951. "The negative effect of previous experience on productive thinking." *Journal of Experimental Psychology* 41:121-126.

Borgatti, Stephen P., and Martin G. Everett. 1999. "Models of core/periphery structures." *Social Networks* 21:375-395.

Bourdieu, Pierre. [1972] 1977. *Outline of a theory of practice*. Cambridge, UK: Cambridge University Press

Brint, Steven. 2001. "Gemeinschaft revisited: A Critique and Reconstruction of the Community Concept." *Sociological Inquiry* 19:1-23.

Brown, John Seely, and Paul Duguid. 1991. "Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation." *Organization Science* 2:40-57.

—. 2001. "Knowledge and Organization: A Social-Practice Perspective." *Organization Science* 12:198-213.

Brown, Shona L, and Kathleen M Eisenhardt. 1997. "The art of continuous change: Linking complexity theory and time-paced evolution in relentlessly shifting organizations." *Administrative Science Quarterly* 42:1-34.

Brown, Shona L., and Kathleen M. Eisenhardt. 1995. "Product Development: Past Research, Present Findings, and Future Directions." *Academy of Management Review* 20:343-378.

Burgelman, Robert A. 1983. "A Model of the Interaction of Strategic Behavior, Corporate Context, and the Concept of Strategy." *Academy of Management Review* 8:61-70.

—. 1994. "Fading Memories: A Process Theory of Strategic Business Exit in Dynamic Environments." *Administrative Science Quarterly* 39:24-56.

Calhoun, C. J. 1980. "Community: toward a variable conceptualization for comparative research." *Social History* 5.

Carlile, Paul. 2004. "Transferring, translating, and transforming: An integrative framework for managing knowledge across boundaries." *Organization Science* 15:555-568.

Chandler, A.D. 1977. *The Visible Hand*. Cambridge, MA: Harvard University Press.

Chubin, Daryl E. 1976. "The Conceptualization of Scientific Specialties." *The Sociological Quarterly* 17:448-476.
Clark, K. B., and T Fujimoto. 1991. *Product development performance*. Boston: Harvard Business School.
Clark, Kim B. 1985. "The interaction of design hierarchies and market concepts in technological evolution." *Research Policy* 14:235-251.
Cohen, Wesley M., and Daniel A. Levinthal. 1989. "Innovation and Learning: The Two Faces of R & D." *The Economic Journal* 99:569-596.
—. 1990. "Absorptive Capacity: A New Perspective on Learning and Innovation." *Administrative Science Quarterly* 35:128-152.
Crane, Diana. 1969. "Social Structure in a Group of Scientists: A Test of the "Invisible College" Hypothesis." *American Sociological Review* 34:335-352.
Crowston, Kevin. 1997. "A Coordination Theory Approach to Organizational Process Design." *Organization Science* 8:157-175.
Cusumano, Michael A. 1992. "Shifting Economies: From Craft Production to Flexible Systems and Software Factories." *Research Policy* 21:453-480.
Cusumano, Michael A, and Richard W Selby. 1997. "How Microsoft builds software." *Communications of the ACM* 40:53-61.
Cusumano, Michael A. 2004. *The business of software : what every manager, programmer, and entrepreneur must know to thrive and survive in good times and bad*. New York: Free Press.
Cusumano, Michael A. , and Richard B. Selby. 1995a. *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets and Manages People*. New York: The Free Press.
Cusumano, Michael A., Alan MacCormack, Chris F. Kemerer, and William Crandall. 2003. "Software development worldwide: The state of the practice." *IEEE Software* 20:28-34.
Cusumano, Michael, and Richard W Selby. 1996. "How Microsoft competes." *Research Technology Management* 39:26-30.
Cusumano, Michael, and Robert Selby. 1995b. *Microsoft secrets: how the world's most powerful software company creates technology, shapes markets and manages people*. New York, NY: Free Press.
Das, T. K, and B. Teng. 1998. "Between trust and control: developing confidence in partner cooperation in alliances." *Academy of Management Review* 23:491-512.
DeMonaco, Harold J., Ayfer Ali, and Eric von Hippel. 2005. "The Major Role of Clinicians in the Discovery of Off-Label Drug Therapies." *MIT Sloan School of Management Working Paper Series*.
Dunbar, Kevin. 1998. "Problem Solving." Pp. 289-298 in *A companion to Cognitive Science*, edited by W Bechtel and G Graham. London, UK: Blackwell.
Duncker, K. 1945. "On problem solving." *Psychology Monographs* 58.
Durkheim, Emile. 1966. *Suicide: A study in sociology*. New York, NY: Free Press.
—. [1911] 1965. *The elementary forms of religious life*. New York, NY: Free Pass.
Edge, David O, and Michael J Mulkay. 1974. "Case studies of scientific specialties." University of Edinburgh, Science Studies Unit.
Eisenhardt, Kathleen M. 1989. "Building theories from case study research." *Academy of Management Review* 14:532-550.

Eisenhardt, Kathleen M. 1991. "Better Stories and Better Constructs: The Case for Rigor and Comparative Logic." *Academy of Management Review* 16:620-627.
Ekeh, Peter P. 1974. *Social Exchange Theory: The Two Traditions*. Cambridge, MA: Harvard University Press.
Faulkner, R. R. 1987. *Music on demand: Composers and Careers in the Hollywood Film Industry*. New Brunswick, NH: Transaction Books.
Feller, Joe, Brian Fitzgerald, Scott Hissam, and Karim R Lakhani (Eds.). 2005. *Perspectives on Free and Open Source Software*. Cambridge: MIT Press.
Foss, Nicolai J. 2003. "Selective Intervention and Internal Hybrids: Interpreting and Learning from the Rise and Decline of the Oticon Spaghetti Organization." *Organization Science* 14:331-349.
Galbraith, J. 1973. *Designing complex organizations*. Reading, MA: Addison-Wesley.
Galison, Peter. 1999. "Trading zone: Coordinating action and belief." Pp. 137-160 in *The Science Studies Reader*, edited by Mario Biagioli. New York, NY: Routledge.
Galison, Peter, and David J Stump (Eds.). 1996. *The Disunity of Science: Boundaries, Contexts and Power*. Stanford: CA: Stanford University Press.
Gallivan, Michael J. 2001. "Striking a balance between trust and control in virtual organization: a content analysis of open source software case studies." *Information Systems Journal* 11:277-304.
German, Daniel. 2005. "Software engineering practices in the GNOME project." Pp. 211-227 in *Perspectives on Free and Open Source Software*, edited by Joseph Feller, Brian Fitzgerald, Scott A Hissam, and Karim R Lakhani. Cambridge, MA: MIT Press.
Ghosh, Rishab Ayer, Ruediger Glott, Bernhard Krieger, and Gregorio Robles. 2002. "Free/Libre and Open Source Software: Part IV Survey of Developers." International Institute of Infonomics, University of Maastricht.
Girard, Monique, and David Stark. 2002. "Distributing intelligence and organizing diversity in new-media projects." *Environment and Planning A* 34:1927-1949.
Glaser, B, and A Strauss. 1967. *The discovery of grounded theory: Strategies for qualitative research*. New York, NY: Aldine de Gruyter.
Gläser, Jochen 2001. "'Producing Communities' as a Theoretical Challenge." Pp. 1-11 in *TASA 2001*. The University of Sydney.
Gottman, Jean (Ed.). 1980. *Centre and Periphery*. Beverly Hills, CA: Sage Publications.
Granovetter, M. 1973. "The strength of weak ties." *American Journal of Sociology* 78:1360-1380.
Hansen, Morten T. 1999. "The search-transfer problem: The role of weak ties in sharing knowledge across organization subunits." *Administrative Science Quarterly* 44:82-111.
Hausmann, Robert G. M. 2003. "Co-construction: A proposed mechanism for collaborative problem solving (dissertation prospectus)." Pp. 1-48: University of Pittsburg.
Hayek, F. A. 1945. "The use of knowledge in society." *American Economic Review* 35:519-530.
Hertel, Guido, Sven Niedner, and Stefanie Herrmann. 2003. "Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel." *Research Policy* 32:1159-1177.

Hillery, George A. 1955. "Definitions of Community: Areas of Agreement." *Rural Sociology* 20:111-123.
Hughes, Everett C. 1971. *The sociological eye*. Chicago: Aldine-Atherton.
Hutchins, Edwin. 1991. "Organizing work by adaptation." *Organization Science* 2:14-39.
Jarvenpaa, Sirkka L, and Dorothy E Leidner. 1999. "Communication and trust in global virtual teams." *Organization Science* 10:791-815.
Jorgensen, Niels. 2005. "Incremental and decentralized integration in FreeBSD." Pp. 227-245 in *Perspectives on Free and Open Source Software*, edited by Joseph Feller, Brian Fitzgerald, Scott A Hissam, and Karim R Lakhani. Cambridge, MA: MIT Press.
Kamps, Jaap, and Laszlo Polos. 1999. "Reducing uncertainty: A formal theory of organizations in action." *American Journal of Sociology* 104:1776-1812.
Knorr-Cetina, Karin D. 1982. "Scientific Communities or Transepistemic Arenas of Research? A Critique of Quasi-Economic Models of Science." *Social Studies of Science* 12:101-130.
Knorr Cetina, Karin. 1999. *Epistemic Cultures: how the sciences make knowledge*. Cambridge, MA: Harvard University Press.
—. 2001. "Objectual practice." Pp. 175-188 in *The contemporary turn in practice theory*, edited by Theodore R Schatzki, Karin Knorr Cetina, and Eike Von Savigny. London, UK: Routledge.
Koch, S, and G Schneider. 2002. "Effort, Cooperation and Coordination in an Open Source Software Project: GNOME." *Information Systems Journal* 12:27-42.
Kraut, Robert E., and Lynn A. Streeter. 1995. "Coordination in Software Development." *Communications of the ACM* 38:69-81.
Krugman, Paul. 1996. *The self-organizing economy*. Oxford, UK: Blackwell.
Kuhn, Thomas. 1970. *The structure of scientific revolutions*. Chicago, Ill: University of Chicago Press.
Lakhani, Karim R, and Robert Wolf. 2005. "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects." in *Perspectives on Free and Open Source Software*, edited by Joe Feller, Brian Fitzgerald, Scott Hissam, and Karim R Lakhani. Cambridge, MA: MIT Press.
Lakhani, Karim R., and Eric von Hippel. 2003. "How Open Source Software Works: Free User to User Assistance." *Research Policy* 32:923-943.
Langlois, Richard N. 1983. "Systems theory, knowledge, and the social sciences." Pp. 581-800 in *The Study of Information: Interdisciplinary Messages*, edited by Fritz Machulp and Una Mansfield. New York, NY: John Wiley.
Lave, Jean, and Etienne Wenger. 1991. *Situated Learning: Legitimate Peripheral Participation*. Cambridge, UK: Cambridge University Press.
Lawrence, P, and J Lorsch. 1967. "Differentiation and integration in complex organizations." *Administrative Science Quarterly* 12:1-47.
Lee, Gwendolyn, and Robert E Cole. 2003. "From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development." *Organization Science* 14:633-649.
Leonard, D, and S. Sensiper. 1998. "The role of tacit knowledge in group innovation." *California Management Review* 40:112-132.

Lerner, Josh, and Jean Tirole. 2002. "Some Simple Economics of Open Source." *Journal of Industrial Economics* 50:197-234.
Lientz, B. P., and E.B. Swanson. 1980. *Software Maintenance Management: A Study of the Maintenance of Computer Applications Software in 487 Data Processing Organizations*. Reading, MA: Addison-Wesley.
MacCormack, A, R. Verganti, and Iansiti. 2001. "Developing products on "Internet Time": The anataomy of flexible development process." *Management Science* 47:133-150.
March, James G, and Herbert Simon. 1958. *Organizations*: Wiley.
Markus, Lynne M, Brook Mannvile, and Carole E Agres. 2000. "What make a virtual organization work? Lessons from the open-source world." *Sloan Management Review* Fall 2000, 42:13-26.
Marples, D. L. 1961. "The Decisions of Engineering Design." *IRE Transactions on Engineering Management*:55-71.
McDaniel, S. , G. Olson, and J. Magee. 1996. "Identifying and Analyzing Multiple Threads in Computer-Mediated and Face-to-Face Conversations." *Proceedings of the Conference on Computer Supported Cooperative*:39-47.
Merton, Robert K. [1942] 1973. *The sociology of science: Theoretical and empirical investigation*. Chicago: University of Chicago Press.
Meyerson, D, K. E. Weick, and R. P. Kramer. 1996. "Swift trust and temporary groups." in *Trust in organizations: Frontiers of theory and research*, edited by Kramer R. M. and T. R. Tyler. Thousand Oaks, CA: Sage Press.
Mintzberg, Henry. 1978. "Patterns in Strategy Formation." *Management Science* 24:934-948.
Mintzberg, Henry, and Alexandra McHugh. 1985. "Strategy Formation in an Adhocracy." *Administrative Science Quarterly* 30:160-197.
Mockus, Audris, Roy Fielding, and James Herbsleb. 2002. "Two case studies of open source software development: Apache and Mozilla." *ACM Transactions on Software Engineering and Methodology* 11:1-38.
Newell, Allen, and H.A. Simon. 1972. *Human Problem Solving*. Engelwood Cliffs, New Jersey: Prentice-Hall INC. .
Noda, Tomo, and Joseph L. Bower. 1996. "Strategy Making as Iterated Processes of Resource Allocation." *Strategic Management Journal* 17:159-192.
Orlikowski, Wanda J. 2002. "Knowing in practice: Enacting a collective capability in distributed organizing." *Organization Science* forthcoming.
Orlikowski, Wanda J, and JoAnne Yates. 1994. "Genre repertoire: Examining the Structuring of Communicative Practices in Organizations." *Administrative Science Quarterly* 39:541-574.
Orr, Julian E. 1996. *Talking about machines: an ethnography of a modern job*. Ithaca, NY: ILR/Cornell University Press.
Osterlund, Carsten, and Paul Carlile. 2003. "How practice matters: A relational view of knowledge sharing." in *Communities and Technologies*, edited by Marleen Huysman, Etienne Wenger, and Volker Wulf. Boston, MA: Kluwer Academic Press.
Perrow, Charles. 1986. *Complex Organizations: A Critical Essay, Third Edition*. New York, NY: McGraw-Hill.

Powell, Anne, Gabriele Piccoli, and Blake Ives. 2004. "Virtual Teams: A Review of Current Literature and Directions for Future Research." *Database for Advanced in Information Systems* 35:6-36.
Raymond, Eric. 1999. *The Cathedral and the Bazaar: Musings on Linux and Open Source from an Accidental Revolutionary.* Sebastopol: CA: O'Reilly and Associates.
Riggs, William, and Eric von Hippel. 1994. "Incentives to innovate and the sources of innovation: The case of scientific instruments." *Research Policy* 23:459-469.
Samolados, Ioannis, Ioannis Stamelos, Lefteris Angelis, and Apostolos Oikonomou. 2004. "Open source software development should strive for even greater code maintainability." *Communications of the ACM* 47:83 - 87.
Sarker, S., F. Lau, and S. Sahay. 2001. "Using an Adapted Grounded Theory Approach for Inductive Theory Building about Virtual Team Development." *Database for Advanced in Information Systems* 32:38-56.
Scharpf, Fritz W. 1997. *Games Real Actors Play. Actor-Centered Institutionalism in Policy Research.* Boulder, CO: Westview Press.
Schatzki, Theodore R. 2001. "Introduction: practice theory." Pp. 1-14 in *The practice turn in contemporary theory*, edited by Theodore R Schatzki, Karin Knorr Cetina, and Eike Von Savigny. New York, NY: Routledge.
Scott, W. Richard. 1998. *Organizations: rational, natural and open systems (4th edition).* Upper Saddle River, NJ: Prentice-Hall.
Simon, H.A., and Allen Newell. 1962. "Computer Simulation of Human Thinking and Problem Solving." *Monographs of the Society for Research in Child Behavior* 27:137-150.
Simon, Herbert Alexander. 1976. *Administrative Behavior, Third Edition.* New York, NY: The Free Press.
Sproull, Lee, Caryn Conley, and Jae Yun Moon. 2005. "Prosocial behavior on the net." Pp. 139-162 in *The Social Net: Human Behavior in Cyberspace*, edited by Yair Amichai-Hamburger. Oxford, UK: Oxford University Press.
Stinchcombe, Arthur L. 1959. "Bureaucratic and Craft Administration of Production: A Comparative Study." *Administrative Science Quarterly* 4:168-187.
Stonebraker, Michael , Lawrence A. Rowe, and Michael Hirohama. 1990. "The Implementation of Postgres." *IEEE Transactions on Knowledge and Data Engineering* 2:125-142.
Strauss, Anslem, and Juliet Corbin. 1990. *Basics of qualitative research.* Thousand Oaks, CA: Sage.
Thompson, J. 1967. *Organizations in action.* New York: McGraw-Hill.
Tönnies, Ferdinand. [1887] 1957. *Community and Society.* New York, NY: Harper.
Tushman, Michael L. 1977. "Special boundary roles in the innovation process." *Administrative Science Quarterly* 22:587-605.
Tyre, Marcie J, and Eric von Hippel. 1997. "The situated nature of adaptive learning in organizations." *Organization Science* 8:71-83.
von Hippel, Eric. 1978. "Successful industrial products from customer ideas." *Journal of Marketing* 42:39-49.
—. 1982. "Get New Products from Customers." *Harvard Business Review* 60:117-122.
—. 1988. *The Sources of Innovation.* New York, NY: Oxford University Press.

—. 1989. "New Product Ideas from "Lead Users"." *Research Technology Management* 32:24-27.
—. 1994a. "'Sticky information' and the locus of problem solving: Implications for innovation." *Management Science* 40:429-439.
—. 1994b. "Sticky Information and the Locus of Problem Solving." *Management Science* 40:429-439.
—. 1999. "Economics of product development by users: Impact of "sticky" local information." *Management Science* 44:629-644.
—. 2001. "Innovation by User Communities: Learning from Open Source Software." *Sloan Management Review* 42:82-86.
—. 2005. *Democratizing Innovation.* Cambridge, MA: MIT Press.
von Krogh, Georg, Sebastian Spaeth, and Karim R Lakhani. 2003. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study." *Research Policy* 32:1217-1241.
Weber, Max. 1947. *The Theory of Social and Economic Organization.* New York: Oxford University Press.
Weber, Steve. 2004. *The success of open source.* Cambridge, MA: Harvard University Press.
Weiman, Gabriel. 1982. "On the Importance of Marginality: One More Step into the Two-Step Flow of Communication." *American Sociological Review* 47:764-773.
Wenger, Etienne. 1998. *Communities of Practice: Learning, Meaning and Identity.* Cambridge, UK: Cambridge University Press.
Woolgar, Steve W. 1976. "The Identification and Definition of Scientific Collectivities." Pp. 235-245 in *Perspectives on the Emergence of Scientific Disciplines*, edited by Gerard Lemaine, Roy MacLeod, Michael Mulkay, and Peter Weingart. The Hague: Mouton & Co.
Yates, JoAnne, Wanda J Orlikowski, and Stephanie L Woerner. 2003. "Virtual Organizing: Using Threads to Coordinate Distributed Work." *MIT Sloan School of Management Working Paper Series* 4320-03.

Appendix – Listing of Features Developed by PostgreSQL Community

#	Area	Change	Change Categorization
1	Contrib Changes	* New pg_autovacuum allows automatic "VACUUM"	Adaptive Periphery
2	Contrib Changes	* Prevent crash in xml	DOM Periphery
3	Contrib Changes	* Fix bug in metaphone[] in fuzzystrmatch	DOM Periphery
4	Contrib Changes	* Update spi/timetravel	DOM Periphery
5	Contrib Changes	* Update earthdistance to use cube	DOM Periphery
6	Contrib Changes	* Update cube	DOM Periphery
7	Contrib Changes	* Add hash-based crosstab function to tablefuncs	DOM Periphery
8	Contrib Changes	* Portability improvements to pgcrypto	DOM Periphery
9	Contrib Changes	* Add serial column to order connectby[] siblings in tablefuncs	DOM Periphery
10	Contrib Changes	* Update btree_gist	DOM Periphery
11	Contrib Changes	* New tsearch2 full-text search module	DOM Periphery
12	Contrib Changes	* Improve adddepend	DOM Periphery
13	Contrib Changes	* Improve pgstattuple	DOM Periphery
14	Contrib Changes	* Add named persistent connections to dblink	DOM Periphery
15	Contrib Changes	* Improve intarray	DOM Periphery
16	Contrib Changes	* Fix dbase "-s" option and improve non-ASCII handling	DOM Periphery
17	Contrib Changes	* Improve earthdistance	DOM Periphery
18	Contrib Changes	* Make pgbench honor environment variables PGHOST, PGPORT, PGUSER	DOM Core
19	Data Type and Function Changes	* Allow cidr data type to be cast to text	Adaptive Core
20	Data Type and Function Changes	* New function pg_get_triggerdef-prettyprint- and pg_constraint_is_visible	Adaptive Periphery
21	Data Type and Function Changes	* New hostmask[] function	Adaptive Periphery
22	Data Type and Function Changes	* Allow user defined aggregates to use polymorphic functions	Adaptive Periphery
23	Data Type and Function Changes	* New array functions array_append, array_cat, array_lower, array_prepend, array_to_string, array_upper, string_to_array	Adaptive Periphery
24	Data Type and Function Changes	* Allow WHERE qualification expr op ANY/SOME/ALL [array_expr]	Adaptive Periphery
25	Data Type and Function Changes	* Allow array concatenation with ||	Adaptive Periphery
26	Data Type and Function Changes	* Allow indexes on array columns	Adaptive Periphery
27	Data Type and Function Changes	* Allow proper comparisons for arrays, including ORDER BY and DISTINCT support	Adaptive Periphery

28	Data Type and Function Changes	* Arrays may now be specified as ARRAY[1,2,3], ARRAY[['a','b'],['c','d']], or ARRAY[ARRAY[ARRAY[2]]]	Adaptive Periphery
29	Data Type and Function Changes	* Allow functions that can take any argument data type and return any data type, using anyelement and anyarray	Adaptive Periphery
30	Data Type and Function Changes	* Add md5[] function to main server, already in "contrib/pgcrypto"	Adaptive Periphery
31	Data Type and Function Changes	* Add family[] function to report whether address is IPv4 or IPv6	Adaptive Periphery
32	Data Type and Function Changes	* Add IPv6 support to the inet and cidr data types	Adaptive Periphery
33	Data Type and Function Changes	* New server parameter extra_float_digits to control precision display of floating-point numbers	Adaptive Periphery
34	Data Type and Function Changes	* Input date order must now be YYYY-MM-DD - with 4-digit year - or match datestyle	DOM Core
35	Data Type and Function Changes	* Prevent interval from suppressing :00 seconds display	DOM Core
36	Data Type and Function Changes	* Treat NaN as larger than any other value in min[]/max[]	DOM Core
37	Data Type and Function Changes	* Make EXTRACT[TIMEZONE] and SET/SHOW TIME ZONE follow the SQL convention for the sign of time zone offsets, i.e., positive is east from UTC	DOM Core
38	Data Type and Function Changes	* Make float[p] measure the precision "p" in binary digits, not decimal digits	DOM Core
39	Data Type and Function Changes	* Trim trailing spaces when char is cast to varchar or text	DOM Core
40	Data Type and Function Changes	* Disallow invalid time zone names in SET TIMEZONE	DOM Core
41	Data Type and Function Changes	* Allow 60 in seconds fields of time, timestamp, and interval input values	DOM Core
42	Data Type and Function Changes	* Trap division by zero in case the operating system doesn't prevent it	DOM Core
43	Data Type and Function Changes	* Change EXTRACT[EPOCH FROM timestamp] so timestamp without time zone is assumed to be in local time, not GMT	DOM Core
44	Data Type and Function Changes	* Fix date_trunc-'quarter', ...-	DOM Periphery
45	Data Type and Function Changes	* Allow only datestyle field order for date values not in ISO-8601 format	DOM Periphery
46	Data Type and Function Changes	* Allow assignments to empty arrays	DOM Periphery
47	Data Type and Function Changes	* Make initcap[] more compatible with Oracle	DOM Periphery
48	Data Type and Function Changes	* Have SHOW datestyle generate output similar to that used by SET datestyle	DOM Core
49	Data Type and Function Changes	* Allow +1300 as a numeric time-zone specifier, for FJST	DOM Core

50	Data Type and Function Changes	* Fixes for to_char[] and to_timestamp[]	DOM Periphery
51	Data Type and Function Changes	* Allow time to be specified as 040506 or 0405	DOM Core
52	Data Type and Function Changes	* Add new datestyle values MDY, DMY, and YMD to set input field order; honor US and European for backward compatibility	DOM Core
53	Data Type and Function Changes	* Change the numeric data type internally to base 10000	DOM Core
54	Data Type and Function Changes	* Make pg_get_constraintdef to support unique, primary-key, and check constraints	DOM Periphery
55	Data Type and Function Changes	* Increase date range of timestamp	DOM Periphery
56	Data Type and Function Changes	* String literals like 'now' or 'today' will no longer work as a column default. Use functions such as now[], current_timestamp instead. change required for prepared statements	DOM Core
57	Data Type and Function Changes	* Remove rarely used functions oidrand, oidsrand, and userfntest functions	DOM Periphery
58	JDBC Changes	* Support SSL connections	Adaptive Core
59	JDBC Changes	* Allow executeBatch on a prepared statement	Adaptive Core
60	JDBC Changes	* Allow setNull on updateable result sets	Adaptive Core
61	JDBC Changes	* Add refcursor support	Adaptive Periphery
62	JDBC Changes	* Handle schema names in result sets	DOM Periphery
63	libpq Changes	* Add function PQexecPrepared and PQsendQueryPrepared functions which perform bind/execute of previously prepared statements	Adaptive Core
64	libpq Changes	* Add ability to pass binary data directly to the server	Adaptive Core
65	libpq Changes	* Allow access to the current transaction status	Adaptive Core
66	libpq Changes	* Allow thread-safe libpq with "configure" option "--enable-thread-safety"	Adaptive Periphery
67	libpq Changes	* Allow access to the underlying table and column of a query result	Adaptive Core
68	libpq Changes	* Allow new error codes and levels of text	Adaptive Core
69	libpq Changes	* Make PQsetdbLogin have the same defaults as PQconnectdb	DOM Core
70	libpq Changes	* Add function PQfreemem for freeing memory on Windows, suggested for "NOTIFY"	DOM Core
71	libpq Changes	* Allow libpq to cleanly fail when result sets are too large	DOM Core
72	libpq Changes	* Improve performance of function PGunescapeBytea	DOM Periphery
73	libpq Changes	* Control SSL negotiation with sslmode values disable, allow, prefer, and require	DOM Periphery
74	libpq Changes	* Allow function pqInternalNotice to accept a format string and arguments instead of just a preformatted message	DOM Periphery
75	libpq Changes	* Document service capability, and add sample file	DOM Core

76	Miscellaneous Interface Changes	* Allow thread-safe embedded SQL programs with "configure" option "--enable-thread-safety"	Adaptive Periphery
77	Miscellaneous Interface Changes	* Add Informix compatibility to ECPG	Adaptive Periphery
78	Miscellaneous Interface Changes	* Prevent possible memory leak or core dump during libpgtcl shutdown	DOM Core
79	Miscellaneous Interface Changes	* Add type decimal to ECPG that is fixed length, for Informix	DOM Periphery
80	Object Manipulation Changes	* Add ALTER TABLE ... CLUSTER ON	Adaptive Periphery
81	Object Manipulation Changes	* Add statement-level triggers	Adaptive Periphery
82	Object Manipulation Changes	* Add ALTER SEQUENCE to modify minimum, maximum, increment, cache, cycle values	Adaptive Periphery
83	Object Manipulation Changes	* Add ALTER TABLE ... WITHOUT OIDS	Adaptive Periphery
84	Object Manipulation Changes	* Add "ALTER DOMAIN"	Adaptive Periphery
85	Object Manipulation Changes	* Add check constraints for domains	Adaptive Periphery
86	Object Manipulation Changes	* Add WITH GRANT OPTION clause to "GRANT"	Adaptive Core
87	Object Manipulation Changes	* Disallow dollar signs in operator names, so x=$1 works	DOM Core
88	Object Manipulation Changes	* Fix several zero-column table bugs	DOM Core
89	Object Manipulation Changes	* Make "CREATE SEQUENCE" grammar more conforming to SQL 2003	DOM Periphery
90	Object Manipulation Changes	* Allow copying table schema using LIKE subtable, also SQL 2003 feature INCLUDING DEFAULTS	DOM Periphery
91	Object Manipulation Changes	* Have ALTER TABLE ... ADD PRIMARY KEY add not-null constraint	DOM Periphery
92	Object Manipulation Changes	* Improve automatic type casting for domains	DOM Periphery
93	Object Manipulation Changes	* Allow dollar signs in identifiers, except as first character	DOM Core
94	Performance Improvements	* Add parameter from_collapse_limit to control conversion of subqueries to joins	Adaptive Core
95	Performance Improvements	* Allow join optimization of explicit inner joins, disable with join_collapse_limit	Adaptive Core
96	Performance Improvements	* Fix hash indexes which were broken in rare cases	DOM Core
97	Performance Improvements	* Use faster and more powerful regular expression code from Tcl	DOM Core
98	Performance Improvements	* Improve hash index concurrency and speed	DOM Core
99	Performance Improvements	* Allow hash/merge joins on complex joins	DOM Core
100	Performance Improvements	* Improve optimizer cost computations, particularly for subqueries	DOM Core

101	Performance Improvements	* Add ability to inline simple SQL functions	DOM Core
102	Performance Improvements	* Improve constant folding	DOM Core
103	Performance Improvements	* Make nested-loop joins be smarter about multicolumn indexes	DOM Core
104	Performance Improvements	* Add hashing for GROUP BY aggregates	DOM Core
105	Performance Improvements	* Allow the postmaster to preload libraries using preload_libraries	DOM Periphery
106	Performance Improvements	* Align shared buffers on 32-byte boundary for copy speed improvement	DOM Periphery
107	Performance Improvements	* Improve trigger/constraint performance	DOM Periphery
108	Performance Improvements	* Data type numeric reimplemented for better performance	DOM Core
109	Performance Improvements	* Improve speed of col IN - const, const, const,	DOM Core
110	Performance Improvements	* Improve connection startup time	DOM Core
111	Performance Improvements	* Use bit-mapped relation sets in the optimizer	DOM Core
112	Performance Improvements	* Allow hash joins for more data types	DOM Core
113	Performance Improvements	* Deduce that WHERE a.x = b.y AND b.y = 42 also means a.x = 42	DOM Core
114	Performance Improvements	* Avoid sort when subquery ORDER BY matches upper query	DOM Core
115	Performance Improvements	* Allow most IN subqueries to be processed as joins	DOM Core
116	Performance Improvements	* Improve NOT IN - subquery performance	DOM Core
117	Performance Improvements	* Allow IN/NOT IN to be handled via hash tables	DOM Core
118	Performance Improvements	* Improve GEQO optimizer performance	DOM Core
119	Performance Improvements	* Reduce memory usage for queries using complex functions	DOM Core
120	Performance Improvements	* Allow multikey hash joins	DOM Core
121	Performance Improvements	* Pattern matching operations can use indexes regardless of locale	DOM Core
122	pg_dump Changes	* Make pg_dump preserve column storage characteristics	Adaptive Periphery
123	pg_dump Changes	* Allow pg_dump to dump specific schemas	Adaptive Periphery
124	pg_dump Changes	* Prevent pg_dump from lowercasing identifiers specified on the command line	DOM Core
125	pg_dump Changes	* Make pg_dump preserve "CLUSTER" characteristics	DOM Periphery
126	pg_dump Changes	* Multiple pg_dump fixes, including tar format and large objects	DOM Periphery

127	pg_dump Changes	* Long options for pg_dump are now available on all platforms PostgreSQL now includes its own long-option processing routines.	DOM Core
128	pg_dump Changes	* pg_dump options "--use-set-session-authorization" and "--no-reconnect" now do nothing, all dumps use "SET SESSION AUTHORIZATION" pg_dump no longer reconnects to switch users, but instead always uses "SET SESSION AUTHORIZATION". This will reduce password	DOM Core
129	pg_dump Changes	* Allow pg_dumpall to support the options "-a", "-s", "-x" of pg_dump	DOM Core
130	pg_dump Changes	* Have pg_dumpall use "GRANT"/"REVOKE" to dump database-level privleges	DOM Core
131	psql Changes	* Add backslash commands for listing schemas, casts, and conversions	Adaptive Periphery
132	psql Changes	* New prompt escape sequence %x to show transaction status	Adaptive Core
133	psql Changes	* New "\set VERBOSITY" to control error detail	Adaptive Core
134	psql Changes	* New "\set AUTOCOMMIT off" capability	Adaptive Core
135	psql Changes	* "\encoding" now changes based on the server parameter client_encoding server	DOM Core
136	psql Changes	* Improve tab completion	DOM Periphery
137	psql Changes	* Save editor buffer into readline history	DOM Periphery
138	psql Changes	* Long options for psql are now available on all platforms	DOM Core
139	psql Changes	* Improve "\d" display	DOM Periphery
140	psql Changes	* Enhance HTML mode to be more standards-conforming	DOM Periphery
141	psql Changes	* Add \pset pager always to always use pager	DOM Periphery
142	psql Changes	* Reorder \? help into groupings	DOM Periphery
143	Query Changes	* New SQL-standard information schema	Adaptive Core
144	Query Changes	* Add option to prevent auto-addition of tables referenced in query	Adaptive Periphery
145	Query Changes	* Implement CREATE TABLE AS EXECUTE	Adaptive Periphery
146	Query Changes	* Add read-only transactions	Adaptive Core
147	Query Changes	* Fix aggregates in subqueries to match SQL standard	DOM Core
148	Query Changes	* Print key name and value in foreign-key violation messages	DOM Periphery
149	Query Changes	* Allow users to see their own queries in pg_stat_activity	DOM Periphery
150	Query Changes	* Allow UPDATE ... SET col = DEFAULT	DOM Periphery
151	Query Changes	* Allow expressions to be used in LIMIT/OFFSET	DOM Core
152	Server Configuration Changes	* Add server parameter regex_flavor to control regular expression processing	Adaptive Core
153	Server Configuration Changes	* postgres --describe-config now dumps server config variables	Adaptive Periphery
154	Server Configuration Changes	* Add Mac OS X Rendezvous server support	Adaptive Periphery

155	Server Configuration Changes	* Add ability to print only slow statements using log_min_duration_statement	Adaptive Periphery
156	Server Configuration Changes	* Add checkpoint_warning to warn of excessive checkpointing	Adaptive Core
157	Server Configuration Changes	* New parameter log_error_verbosity to control error detail	Adaptive Core
158	Server Configuration Changes	* New read-only parameter is_superuser	Adaptive Core
159	Server Configuration Changes	* Make "pg_ctl" better handle nonstandard ports	DOM Periphery
160	Server Configuration Changes	* log_min_messages/client_min_messages now controls debug_* output	DOM Core
161	Server Configuration Changes	* Make default shared_buffers 1000 and max_connections 100, if possible	DOM Core
162	Server Configuration Changes	* Allow "pg_hba.conf" to accept netmasks in CIDR format	DOM Periphery
163	Server Configuration Changes	* Add new columns in pg_settings: context, type, source, min_val, max_val	DOM Periphery
164	Server Configuration Changes	* New "pg_hba.conf" record type hostnossl to prevent SSL connections	DOM Periphery
165	Server Configuration Changes	* Prevent server log variables from being turned off by non-superusers	DOM Core
166	Server Configuration Changes	* Change debug server log messages to output as DEBUG rather than LOG	DOM Core
167	Server Configuration Changes	* Rename hostname_lookup to log_hostname	DOM Core
168	Server Configuration Changes	* Rename show_source_port to log_source_port	DOM Core
169	Server Configuration Changes	* Rename show_*_stats to log_*_stats	DOM Core
170	Server Configuration Changes	* Rename server parameter server_min_messages to log_min_messages	DOM Core
171	Server Configuration Changes	* Remove parameter geqo_random_seed	DOM Core
172	Server Configuration Changes	* New read-only server parameters for localization	DOM Core
173	Server Operation Changes	* New code to detect corrupt disk pages; erase with zero_damaged_pages	Adaptive Core
174	Server Operation Changes	*Allow IPv6 server connections	Adaptive Periphery
175	Server Operation Changes	* Add binary I/O to client/server protocol	Adaptive Core
176	Server Operation Changes	* Add transaction status, table ID, column ID to client/server protocol	Adaptive Core
177	Server Operation Changes	* New error message wording, error codes, and three levels of error detail	Adaptive Core
178	Server Operation Changes	Fix inconsistent index lookups during split of first root page	DOM Core
179	Server Operation Changes	Preserve free space information between server restarts	DOM Core

180	Server Operation Changes	* Improve free space map allocation logic	DOM Core
181	Server Operation Changes	*Update "/tmp" socket modification times regularly to avoid their removal	DOM Core
182	Server Operation Changes	*Print lock information when a deadlock is detected	DOM Core
183	Server Operation Changes	*Enable PAM for Mac OS X	DOM Periphery
184	Server Operation Changes	*Fix SSL to handle errors cleanly	DOM Periphery
185	Server Operation Changes	* Add start time to pg_stat_activity	DOM Periphery
186	Server Operation Changes	*SSL protocol security and performance improvements	DOM Periphery
187	Server Operation Changes	* New client/server protocol: faster, no username length limit, allow clean exit from "COPY"	DOM Core
188	Server Operation Changes	* Allow B-tree index compaction and empty page reuse	DOM Core
189	Server Operation Changes	*Make B-tree indexes fully WAL-safe	DOM Core
190	Server-Side Language Changes	* Add new parameter $0 in PL/pgSQL representing the function's actual return type	Adaptive Periphery
191	Server-Side Language Changes	* Allow polymorphic SQL functions	Adaptive Periphery
192	Server-Side Language Changes	* Allow polymorphic PL/pgSQL functions	Adaptive Periphery
193	Server-Side Language Changes	* Prevent PL/pgSQL crash when RETURN NEXT is used on a zero-row record variable	DOM Core
194	Server-Side Language Changes	* Make PL/Python's spi_execute interface handle null values properly	DOM Periphery
195	Server-Side Language Changes	* Fix PL/Python's _quote[] function to handle big integers	DOM Periphery
196	Server-Side Language Changes	* Allow PL/Tcl and PL/Python to use the same trigger on multiple tables	DOM Core
197	Server-Side Language Changes	* Allow PL/pgSQL to declare variables of composite types without %ROWTYPE	DOM Core
198	Server-Side Language Changes	* Improved compiled function caching mechanism in PL/pgSQL with full support for polymorphism	DOM Periphery
199	Server-Side Language Changes	* Make PL/Python an untrusted language, now called plpythonu	DOM Periphery
200	Server-Side Language Changes	* Fixed PL/Tcl's spi_prepare to accept fully qualified type names in the parameter type list	DOM Core
201	Source Code Changes	* Add Darwin startup scripts	Adaptive Periphery
202	Source Code Changes	* Allow libpq to compile with Borland C++ compiler	Adaptive Periphery
203	Source Code Changes	* Allow OpenBSD to use local ident credentials	Adaptive Periphery
204	Source Code Changes	* Fix locking code for s390x CPU [64-bit]	DOM Core
205	Source Code Changes	* Generate a compile error if spinlock code is not found	DOM Core
206	Source Code Changes	* New function palloc0 to allocate and clear memory	DOM Core
207	Source Code Changes	* Support Intel compiler on Linux	DOM Core
208	Source Code Changes	* Make query plan trees read-only to executor	DOM Core

209	Source Code Changes	* Prevent need for separate platform geometry regression result files	DOM Core
210	Source Code Changes	* Add support for AMD Opteron and Itanium	DOM Periphery
211	Source Code Changes	* Improved PPC locking primitive	DOM Periphery
212	Source Code Changes	* Improve Linux startup scripts	DOM Periphery
213	Source Code Changes	* Add Windows compatibility functions	DOM Core
214	Source Code Changes	* Allow client interfaces to compile under MinGW	DOM Core
215	Source Code Changes	* Convert administration scripts to C	DOM Core
216	Source Code Changes	* Use our own version of getopt_long[] if needed	DOM Core
217	Source Code Changes	* New ereport[] function for error reporting	DOM Core
218	Utility Command Changes	* Functional indexes have been generalized into indexes on expressions	Adaptive Core
219	Utility Command Changes	* Allow "CLUSTER" to cluster all tables	Adaptive Periphery
220	Utility Command Changes	* Add ON COMMIT clause to "CREATE TABLE" for temporary tables	Adaptive Periphery
221	Utility Command Changes	* Allow cursors outside transactions using WITH HOLD	Adaptive Periphery
222	Utility Command Changes	* Implement SQL-compatible options FIRST, LAST, ABSOLUTE n, RELATIVE n for "FETCH" and "MOVE"	Adaptive Core
223	Utility Command Changes	* Add "EXPLAIN EXECUTE"	Adaptive Periphery
224	Utility Command Changes	* Have "COMMENT ON DATABASE" on nonlocal database generate a warning	DOM Core
225	Utility Command Changes	* Have "SHOW TRANSACTION ISOLATION" match input to "SET TRANSACTION ISOLATION"	DOM Core
226	Utility Command Changes	* Prevent possible memory leaks in "COPY"	DOM Core
227	Utility Command Changes	* Prevent "CLUSTER" on partial indexes	DOM Core
228	Utility Command Changes	* Disallow literal carriage return as a data value, backslash-carriage-return and \r are still allowed	DOM Core
229	Utility Command Changes	* Cause "FETCH" and "MOVE" to return the number of rows fetched/moved, or zero if at the beginning/end of cursor, per SQL standard	DOM Core
230	Utility Command Changes	* FETCH 0 and MOVE 0 now do nothing	DOM Core
231	Utility Command Changes	* Recover from "COPY" failure cleanly	DOM Core
232	Utility Command Changes	* "COPY" changes binary, \.	DOM Core
233	Utility Command Changes	* Improve reliability of "LISTEN"/"NOTIFY"	DOM Core
234	Utility Command Changes	* Allow "EXPLAIN" on "DECLARE CURSOR"	DOM Core
235	Utility Command Changes	* Allow "CLUSTER" to use index marked as pre-clustered by default	DOM Periphery
236	Utility Command Changes	* Properly handle SCROLL with cursors, or report an error	DOM Periphery

237	Utility Command Changes	* Make "TRUNCATE" transaction-safe	DOM Periphery
238	Utility Command Changes	* Allow DOS and Mac line-endings in "COPY" files	DOM Core
239	Utility Command Changes	* Allow "REINDEX" to reliably reindex nonshared system catalog indexes	DOM Core
240	Utility Command Changes	* Improve "VACUUM" performance on indexes by reducing WAL traffic	DOM Core
241	Utility Command Changes	* Allow prepare/bind of utility commands like "FETCH" and "EXPLAIN"	DOM Core

Chapter 4: Motivation and Effort of the Core Developers in Open Source Communities [65]

4.1: Introduction

Free/Open Source software (F/OSS) communities represent one of the most unique examples of innovation outside the traditional boundaries of the producing firms. F/OSS in general and the Linux operating system in particular have been called the "impossible public good" (Kollock 1999: 230). How could complex software systems, running mission critical applications, be designed, developed, maintained and improved by a virtual "collective" of mostly volunteer software hackers (Markus, Mannvile and Agres 2000)? Even more confounding, the resultant software products are free to use by anyone and no one organization or person has exclusive legal rights to determine the future direction of a particular project (McGowan 2001). "What drives Free/Open Source software (F/OSS) developers to contribute their time and effort to the creation of free software products?" is a question often posed by software industry executives, managers, and academics when they are trying to understand the relative success of the F/OSS movement. Many are puzzled by what appears to be irrational and altruistic behavior by movement participants: giving code away, revealing proprietary information, and helping strangers solve their technical problems. Understanding the motivations of F/OSS developers is an important first step in determining what is behind the success of the F/OSS development model and other forms of distributed technological innovation and development.

In this chapter, I report on the results of a of the effort and motivations of core developers contributing to the creation of Free/Open Source software. I define core developers as those individuals who have formal decision rights in the project and are identified as being on the core team by the F/OSS project. I used a Web-based survey, administered to 684 software developers in 287 F/OSS projects, to learn what lies behind the effort put into such projects. Academic theorizing on individual motivations for

[65] Research done in collaboration with Bob Wolf and colleagues at the Boston Consulting Group. A shorter version of this paper is in Feller, J., Fitzgerald, B., Hissam, S., & Lakhani, K. R. (Eds.). 2005. Perspectives on Free and Open Source Software. Cambridge: MIT Press,

participating in F/OSS projects has posited that external motivational factors in the form of extrinsic benefits (e.g., better jobs, career advancement) are the main drivers of effort. I find, in contrast, that enjoyment-based intrinsic motivation-namely, how creative a person feels when working on the project is the strongest and most pervasive driver. I also find that user need, intellectual stimulation derived from writing code, and improving programming skills are top motivators for project participation. A majority of the respondents are skilled and experienced professionals working in information technology-related jobs, with approximately 40 percent being paid to participate in the F/OSS project.

The paper is organized as follows. I review the relevant literature on motivations (Section 2) and then describe research methods and sample characteristics (Section 3). I then report on the findings on payment status and effort in projects (Section 4), creativity and motivations in projects (Section 5), and the determinants of effort (Section 6) in projects. I conclude with a discussion of our findings (Section 7).

4.2: Literature Review - Motivations to Participate

What motivates human behavior and action has been a fundamental concern of all branches of social sciences. The literature on human motivations differentiates between those that are intrinsic (the activity is valued for its own sake) and those that are extrinsic (providing indirect rewards for doing the task at hand) (Amabile 1996; Deci and Ryan 1985; Frey 1997). In this section I review the two different types of motivations and their application to developers in F/OSS projects. I also review current empirical evidence on motivations in F/OSS projects.

4.2.1 Intrinsic Motivation

The subject of intrinsic motivation has been well studied in psychology (for comprehensive reviews see Lindenberg 2001). Following Ryan and Deci (Ryan and Deci 2000: 56) "intrinsic motivation is defined as the doing of an activity for its inherent satisfactions rather than for some separable consequence. When intrinsically motivated a person is moved to act for the fun or challenge entailed rather than because of external prods, pressures, or rewards." Core to their theory of intrinsic motivation is a human's

need for competence and self-determination. They link the experience of being competent and self-determined to the emotions of interest and enjoyment (Deci and Ryan 1985: 35). A strong claim, backed by extensive experimental evidence, is that under certain circumstances extrinsic rewards, especially monetary, have a negative impact on intrinsic motivations and task performance that are initially intrinsically based(Deci, Koestner and Ryan 1999). Thus there is a "hidden cost of reward". Frey (1997) in extending psychological findings towards a formal economic theory of intrinsic motivation has called this the "crowding out" effect, where extrinsic rewards crowd out intrinsic rewards. Frey conceptualized intrinsic motivation as doing a task or activity without being induced by monetary payment and/or command.

However it may not be analytically easy to distinguish between extrinsic and intrinsic motivations and their effects on task performance (Frey 1997). Thus for instance an artist may be handsomely paid for creating works that they enjoy doing themselves. Lindenberg (2001) has proposed a reconceptualization of the relationship between intrinsic and extrinsic motivations and further explicated intrinsic motivation. As a starting point, he notes that recently Deci and Ryan (2000) have stated that: "most of the activities people do are not, strictly speaking, intrinsically motivated. This is especially the case after early childhood, as the freedom to be intrinsically motivated becomes increasingly curtailed by social demands and roles that require the individual to assume responsibility for nonintriniscially interesting task." Lindenberg proposes that people have multiple goals while in the midst of completing activities. For example a person may have a goal to make as much money as possible and a goal to have lots of fun (Lindenberg 2001: 322). These goals compete for scarce cognitive resources within a person with one goal eventually taking center stage. The winning goal creates a frame within which a person chooses actions that are most compatible with that goal. However, this does not imply that the other goals have disappeared. Rather they exist in the background and moderate the action alternatives that a person considers within the dominant frame. Thus a person who values making money and having fun may choose opportunities that balance economic reward (i.e. less pay) with a sense of having fun (i.e. more fun). Consistent with extant evidence and theory of intrinsic motivation Lindenberg

(2001) separates intrinsic motivation into two components: 1) enjoyment and 2)obligation/community. I consider each of them below.

Enjoyment-based Intrinsic Motivation

Having fun or enjoying oneself when taking part in an activity is at the core of the idea of intrinsic motivation (Deci and Ryan 1985). Csikszentmihalyi (1975) was one of the first psychologists to study the enjoyment dimension. He emphasized that some activities were pursued for the sake of the enjoyment derived from doing them. He proposed a state of "flow," in which enjoyment is maximized, characterized by intense and focused concentration; a merging of action and awareness; confidence in one's ability; and the enjoyment of the activity itself regardless of the outcome (Nakamura and Csikszentmihalyi 2003). Flow states occur when a person's skill matches the challenge of a task. There is an optimal zone of activity in which flow is maximized. A task that is beyond the skill of an individual provokes anxiety, and a task that is below the person's skill level induces boredom. Enjoyable activities are found to provide feelings of "creative discovery, a challenge overcome and a difficulty resolved" (Csikszentmihalyi 1975: 181). Popular accounts of programming in general and participation in F/OSS projects (Himanen 2001; Torvalds and Diamond 2001) in particular attest to the flow state achieved by people engaged in writing software. Thus F/OSS participants may be seeking flow states by selecting projects that match their skill levels with task difficulty, a choice that might not be available in their regular jobs.

Closely related to enjoyment-based intrinsic motivation is a sense of creativity in task accomplishment. Amabile (1996) has proposed that intrinsic motivation is a key determining factor in creativity. Amabile's definition of creativity consists of: (1) a task that is heuristic (no identifiable path to a solution) instead of algorithmic (exact solutions are known), and (2) a novel and appropriate (useful) response to the task at hand (Amabile 1996: 36). Creativity research has typically relied on normative or objective assessments of creativity with a product or process output judged creative by expert observers. Amabile (Amabile 1996: 40), however, also allows for subjective, personal interpretations of creative acts. In particular, she proposes a continuum of creative acts,

from low-level to high-level, where individual self-assessment can contribute to an understanding of the social factors responsible for creative output. Thus in our case, a F/OSS project dedicated to the development of a device driver for a computer operating system may not be considered terribly creative by outside observers, but may be rated as a highly creative problem-solving process by some individuals engaged in the project.

Obligation/Community-based Intrinsic Motivations

Lindenberg (2001) makes the case that acting on the basis of principle is also a form of intrinsic motivation. He argues that individuals may be socialized into acting appropriately and in a manner consistent with the norms of a group. Thus the goal to act consistently within the norms of a group can trigger a normative frame of action. The obligation/community goal is strongest when private gain-seeking (gaining personal advantage at the expense of other group members) by individuals within the reference community is minimized. He also suggests that multiple motivations, both extrinsic and intrinsic, can be present at the same time. Thus a person who values making money and. having fun may choose opportunities that balance economic reward (i.e., less pay) with a sense of having fun (i.e., more fun).

In F/OSS projects, there is a strong sense of community identification and adherence to norms of behavior. Canonical texts like The New Hacker's Dictionary (Raymond 1996), The Cathedral and the Bazaar (Raymond 1999), and the GNU General Public License (GPL) (Stallman 1999) have created shared meaning about the individual and collective identities of the hacker' culture and the responsibilities of membership within it. Indeed, the term hacker is a badge of honor within the F/OSS community, as opposed to its pejorative use in popular media. The hacker identity includes solving programming problems, having fun, and sharing code at the same time. Private gain-seeking within the community is minimized by adherence to software licenses like the GPL and its derivatives, which allow for user rights to source code and subsequent modification.

4.2.2: Extrinsic Motivation

Economists have contributed the most to our understanding of how extrinsic motivations drive human behavior. "The economic model of human behavior is based on incentives applied from outside the person considered: people change their actions because they are induced to do so by an external intervention. Economic theory thus takes extrinsic motivation to be relevant for behavior" (Frey 1997, 13).

Lerner and Tirole (2002) posit a rational calculus of cost and benefit in explaining why programmers choose to participate in F/OSS projects. As long as the benefits exceed the costs, the programmer is expected to contribute. They propose that the net benefit of participation consists of immediate and delayed payoffs. Immediate payoffs for F/OSS participation can include being paid to participate and user need for particular software (von Hippel 2001). Similarly, von Hippel and von Krogh (2003) have proposed a "private – collective" model of innovation in F/OSS projects where developers are motivated to contribute based on balancing private incentives (need for code) with collective requirements for disclosure.

Although the popular image of the F/OSS movement portrays an entirely volunteer enterprise, the possibility of paid participation should not be ignored as an obvious first-order explanation of extrinsic motivations. Firms might hire programmers to participate in F/OSS projects because they are either heavy users of F/OSS-based information technology (IT) infrastructure or providers of F/OSS-based IT solutions. In either case, firms make a rational decision to hire programmers to con-tribute to F/OSS projects.

Another immediate benefit relates to the direct use of the software product. Research on the sources of innovation has shown that users in general and lead users in particular have strong incentives to create solutions to their particular needs (von Hippel 1988). Users have been shown to be the source of innovations in fields as diverse as scientific instruments (Riggs and von Hippel 1994), industrial products (Urban and von Hippel 1988), sports equipment (Franke and Shah 2003), and library information systems

(Morrison, Roberts and von Hippel 2000). Thus user need to solve a particular software problem may also drive participation in F/OSS projects. Similarly, Raymond (1999), with a participant-observer perspective on open source, has claimed that a primary driver of F/OSS participation is "scratching a coding itch." User need can be both work related (e.g.: a librarian improving a library information system) and/or non-work related (e.g.: a recreational biker adding shock absorbers to a mountain bike). In either case, the users combine their immediate needs with their own skills to solve problems that they best understand (von Hippel 2005).

Career concerns, in the form of job market signaling, is another extrinsic motivation for active participation in F/OSS projects. Participants indicate to potential employers their superior programming skills and talents by contributing code to projects where their performance can be monitored by any interested observer [66]. Similarly, firms looking for a particular skill in the labor market can easily find qualified programmers by examining code contributions within the F/OSS domain. Lerner and Tirole (2002: 214) (citing Holmström (1999)) indicate that "the signaling incentive is stronger:

1. the more visible the performance to the relevant audience;
2. the higher the impact of effort on performance; and
3. the more informative the performance about talent."

Therefore current effort on projects is expended in the expectation of future (delayed) payoffs in the form of better a better job. Lancashire(2001) has used a similar labor-market lens to explain the significantly greater per capita participation in F/OSS projects (Linux and GNOME) by contributors from outside of the United States. His central claim is that in 1990s the locus of F/OSS development shifted over to Europe. This shift is due to the opportunity costs faced by US based software developers—soaring demand and high wages for computer professionals decrease the attractiveness of unpaid activities. On the other hand, European developers not only face lower opportunity costs, but they also benefit by gaining a reputation from participating in open source projects.

[66] The widespread archiving of all F/OSS project related materials like e-mail lists and code commits can allow for a detailed assessment/proof of individual performance.

They can then gain access to higher wage jobs abroad. Thus F/OSS participation is a kind of fixed cost of acquiring reputation and signaling to the job market.

Participants also improve their programming skills through the active peer review that is prevalent in F/OSS projects (Moody 2001; Wayner 2000). Software code contributions are typically subject to intense peer review both before and after a submission becomes part of the official code base. Source code credit files and public e-mail archives ensure that faulty programming styles, conventions, and logic are communicated back to the original author. Peers in the project community, software users, and interested outsiders readily find faults in programming and often suggest specific changes to improve the performance of the code (von Krogh, Spaeth and Lakhani 2003). This interactive process improves both the quality of the code submission and the overall programming skills of the participants.

4.2.3: Current findings on motivations in F/OSS projects

There have been three[67] substantive studies (Ghosh et al. 2002; Hars and Ou 2002; Hertel, Niedner and Herrmann 2003) focused on the motivations of individual developers in F/OSS. Overall the findings from these studies indicate that there is significant heterogeneity in motivations for developers in F/OSS projects. The studies are reviewed below.

Hertel et al. (2003) did a study of the motivation and effort 141 Linux developers (n=69) and observers/interested readers (n=72) subscribed to the Linux kernel mail list. Forty three percent of the Linux developers had received payment (salary or other remuneration) for their contributions. Developers were found on average to have dedicated 18.4 hours/week on Linux development. Using a social movements framing they found that developers rated the following seven motivations (Hertel et al. 2003 , table 2, pg 12) in order of importance (scale = 1-5, 1-very unimportant, 5 – very important): a) hedonistic motives (enjoying programming) – 4.7, b) pragmatic motives (improve software, enhance career) – 4.3, c) social/political motives (software freedom) –

[67] A study by Lakhani and von Hippel (2003) was dedicated to motivations to provide user support for open source projects and did not focus on developers exclusively.

4.1, d) developer identification – 4.0, e) Linux user identification – 3.9, f) norm-oriented motivations related to the reaction of others (family, friends, colleagues) – 3.9, and g) tolerance for time loss for time devoted to development – 3.6. In a regression on the effort expended (hours/week), the authors found that identification as Linux developer had a significant positive effect (beta =0.28, $p<0.01$), identification as Linux user had a significant negative effect (beta = -0.33, $p<0.001$) and as expected, tolerance for time losses had a significant positive effect (beta = 0.5, $p<0.04$(directional)). All other motivations were found to have no significant effects. Interestingly pragmatic motives became significant only when considering the likelihood of continued contributions to Linux. There were also no differences in effects between paid and volunteer contributors.

Hars and Ou (2002) conducted a study of motivations of 79 developers participating in 41 different F/OSS projects, with Linux developers constituting 27% of the sample. They found that 16% of the developers were paid to work on F/OSS projects and that this group contributed 38% of all hours spent on the projects. Using an internal/external motivational framing they found the following ranking of their eight motivational constructs (only those that listed high/very high): a) human capital (skill improvement) – 88.3%, b) self-determination – 79.7%, c) peer recognition – 43.0%, d) user need – 38.5%, e) self-marketing – 36.7%, f) community identification – 27.8%, and g) altruism – 16.5%. They did not conduct any regression analysis to discover which motivations had significant impact on effort, however in contrast with Hertel et al, they found a negative correlation between effort and community identification for paid programmers.

Ghosh et al (2002) conducted a broad based study of users and developers of F/OSS projects for the European Commission. A component of that study included understanding motivations for contribution to developers. Their study, based on snowball sampling on various F/OSS e-mail lists, obtained responses from 2774 developers[68]. They found that the top motivations (>50% agreement) to contribute was

[68] Ghosh et al noted that they could only verify 487 respondents as actual developers. However, they reported results on the entire sample.

"to learn and develop new skills" (~70%)and "to share knowledge and skills with others" (~65%). Based on a preliminary heuristic grouping of all the responses they created four types of participants within their sample: a) those motivated primarily by social reasons – 53.2% , b) career and monetary concerns – 31.4%, c) political motivations – 12.7%, and d) product related (user need) 2.6%. The authors did not conduct any regression analysis to indicate the salience of these motivations in determining effort.

4.3: Research Methods and Sample Characteristics

4.3.1: Sample Selection

Prior studies on motivations in F/OSS projects have been limited by: including developers and interested non-developers in sample; single project focus; relatively small size samples; non-random and uncontrolled sample selection. The sample for this survey was selected from individuals listed as core developers on F/OSS projects hosted on the SourceForge.net website. Von Krogh et al (2003), in their detailed case study of community joining scripts of a Sourceforge.net project, have shown that becoming an core developer requires significant investment in time and effort by the prospective person. Core developers typically have the authority to change and modify source code, or integrate non-official developers changes, without seeking permission from the rest of the project team[69]. Core developers have also been shown to do a majority of the code writing effort and communications in F/OSS projects (Koch and Schneider 2002). Thus the sampling strategy should yield individuals that have significant roles in their respective projects.

At the start of the study period (Fall 2001), SourceForge.net listed 26, 245 active projects. The site requires project administrators to publicly disclose their development status (readiness of software code for day-to-day use) as: Planning, Pre-Alpha, Alpha, Beta, Production/Stable or Mature. Projects that are in the Planning or Pre-Alpha stage typically do not contain any source code and were eliminated from the population under study resulting in 9,973 available projects for the sample. The distribution of the

[69] Although most projects discuss changes quite intensely.

projects, in terms of their development status, was as follows as follows: Alpha-32%, Beta-37%, Production/Stable-28% and Mature-3%.

Two separate but identical surveys were launched over two time periods, the first survey was targeted at Alpha, Beta and Production/Stable projects and the second at the Mature projects. Due to the large number of Alpha, Beta and Production/Stable projects, a 10% random sample was chosen from projects that listed more than one developer. The greater than one developer criteria was used to ensure selection of projects that were not 'pet' software projects parked on SourceForge.net, rather projects that involved some level of coordination with other members. The above activity led to 1825 e-mail addresses and 550 projects. A closer examination of the e-mail addresses revealed that some individuals were on more than one project and were in the sample twice. Eliminating duplicates resulted in 1648 unique individuals in the first survey's sample target. The second survey's target sample was selected by obtaining e-mail addresses of all participants that were on multi-person projects. This identified 103 projects (out of 259 projects) with 573 individuals (out of 997 individuals).

4.3.2 Data Collection

Data collection was accomplished via identical web-based survey instruments (see Appendix for survey). The survey consisted of 6 sections, consisting primarily of closed questions. Participants could fill in their own responses if a question had a response option for "Other". The last section consisted of open questions on the participant's overall impressions about the F/OSS movement. The survey asked questions about their participation in the focal project from which their name was obtained as well as other F/OSS projects that they had participated in. The survey also had demographic, geographic and attitudinal and motivational questions. The survey questions were pre-tested by 5 members of the F/OSS community. The web based survey itself was tested for accuracy in presentation and response recording on three different Internet browsers (Microsoft, Netscape and Opera). The survey took about 30 minutes to complete.

Personalized e-mails were sent to each individual, indicating their first name and their specific project, inviting them to participate in the survey. Each individual in the sample was assigned a random personal identification number (PIN) that gave them access to the survey. The survey form could not otherwise be accessed. The survey respondents were ensured of complete confidentiality in their responses. The PIN system also prevented spurious/accidental users from accessing the website and entering false data into the survey. Respondents were offered the opportunity to participate in a random drawing for gift certificates upon completion of the survey. As soon as an individual completed a survey, the data was automatically recorded in a database and to ensure further back-up, an automatic e-mail was sent to the survey team containing all of the relevant responses from each individual instance of the survey.

The survey was conducted over two time periods. In both cases the invitation e-mails to respondents were sent over a period of two days to avoid over loading the web server hosting the survey. The first survey was from October 10, 2001 to October 30th, 2001. 1530 e-mails reached their destinations and 118 e-mails bounced back due to invalid accounts at host systems. The survey generated 526 responses resulting in a response rate of 34.3%. The second survey was from April 8, 2002 to April 28, 2002. 573 e-mails were sent with all e-mails reaching their destinations. The second survey generated 173 responses resulting in a response rate of 30.0%. Within both the surveys, 98% of the respondents answered at least 90% of all of the questions and 85% of the respondents answered the open ended questions. Close examination of the data revealed that 18 respondents had not completed a majority of the survey or had answered the survey twice (hitting the send button more than once) and were thus eliminated from analysis. Overall the survey had 684 respondents from 287 distinct projects yielding a response rate of 34.3%. The mean number of responses per project was 4.68 (sd = 4.9, median = 3, range = 1-25).

4.3.3 Sample Characteristics

Survey respondents were primarily male (97.5 percent) with an average age of 30 years and living primarily in the developed Western world (45 percent of respondents

from North America (U.S. and Canada) and 38 percent from Western Europe). Table 4.1 summarizes some of the salient characteristics of the sample and their participation in F/OSS projects.

The majority of respondents had training in IT and/or computer science, with 51 percent indicating formal university-level training in computer science and IT. Another 9 percent had on-the-job or other related IT training. Forty percent of the respondents had no formal IT training and were self taught. Overall, 58 percent of the respondents were directly involved in the IT industry, with 45 percent of respondents working as professional programmers and another 13 percent involved as systems administrators or IT managers. Students made up 19.5 percent of the sample and academic researchers 7 percent. The remaining respondents classified their occupation as "other." As indicated by Table 4.1, on average the respondents had 11.8 years of computer programming experience.

4.4: Findings - Payment Status and Effort in Projects

A significant minority of contributors are paid to participate in F/OSS projects. When asked if they had received direct financial compensation for participation in the project, 87 percent of all respondents reported receiving no direct payments. But, as Table 4.2 indicates, 55 percent contributed code during their work time. When asked: "if a work supervisor was aware of their contribution to the project during work hours," 38 percent of the sample indicated supervisor awareness (explicit or tacit consent) and 17 percent indicated shirking their official job while working on the project. The sum of those who received direct financial compensation and those whose supervisors knew of their work on the project equals approximately 40 percent of the sample, a category we call "paid contributors."

Effort was measured as the number of hours per week spent on a project. This measure has been used in previous F/OSS studies (Hars and Ou 2002; Hertel, Niedner, and Herrmann 2003) and provides an appropriate proxy for participant contribution and

interest in F/OSS projects. Other measures of effort can include lines of code contributed and frequency of changes made to the source code. However, all of them can create a

Table 4.1: General Characteristics of Survey Respondents

Variable	Obs	Mean	Std. Dev.	Min	Max
Age	677	29.80	7.95	14.00	56.00
Years Programming	673	11.86	7.04	1.00	44.00
Current F/OSS Projects	678	2.63	2.14	0.00	20.00
All F/OSS Projects	652	4.95	4.04	1.00	20.00
Years since first contribution to F/OSS community	683	5.31	4.34	0.00	21.00

Table 4.2: Location and Work Relationship for F/OSS Contributions

Survey Question: Is supervisor aware of work time spent on the F/OSS project?	N	Percent
Yes, aware	254	37.69
No, not aware	113	16.77
Do not spend time at work	307	45.55
Total	674	100

Table 4.3 - Hours Spent / Week on F/OSS Projects

	Average (sd)	Paid Contributor (sd)	Volunteer (sd)	t statistic (p-value)*
Hours/week on all F/OSS projects	14.3 (15.7)	17.7 (17.9)	11.7 (13.5)	4.8 (0.00)
Hours/week on focal F/OSS project	7.5 (11.6)	10.3 (14.7)	5.7 (8.4)	4.7 (0.00)

* Two tailed test of means assuming unequal variances

bias based on individual styles and habits of programming and committing source code changes (von Hippel and von Krogh 2003; von Krogh, Spaeth and Lakhani 2003). Hours per week, on the other hand, provides an indication of individual commitment to F/OSS projects regardless of skill and style.

Survey respondents were asked how many hours in the past week they had spent working on all their current F/OSS projects in general and "this project" (the focal project about which they were asked motivation questions) in particular. Respondents said that they had, on average, spent 14.1 hours (sd = 15.7, median = 10, range 0-85 hours) on all their F/OSS projects and 7.5 hours (sd = 11.6, median = 3, range 0-75 hours) on the focal project. The distribution of hours spent was skewed, with 11 percent of respondents not reporting any hours spent on their current F/OSS projects and 25 percent reporting zero hours spent on the focal project. Table 4.3 indicates that paid contributors dedicate significantly more time (51 percent) to projects than do volunteers.

Overall, paid contributors were spending more than two working days a week and volunteer contributors were spending more than a day a week on F/OSS projects. The implied financial subsidy to projects is quite substantial. The 2001 United States Bureau of Labor Statistics wage data[70] indicated a mean hourly pay of $30.23 for computer programmers. Thus the average weekly financial contribution to F/OSS projects is $353.69 from volunteers and $535.07 from paid contributors (via their employers).

4.5: Findings - Creativity and Motivation in Projects

Respondents noted a very high sense of personal creativity in the focal projects. They were asked: "imagine a time in your life when you felt most productive, creative, or inspired. Comparing your experience on this project with the level of creativity you felt then, this project is. . . ." Table 4.4 describes the response patterns. More than 61 percent of the survey respondents said that their participation in the focal F/OSS project was their most creative experience or was equally as creative as their most creative experience. The remaining 31% rated the focal projects as "somewhat less creative" and 8% found the projects to be "much less creative" as compared to their most creative experience.

[70] Available at http://www.bls.gov/oes/2001/oes_15Co.htm, accessed April 2, 2003.

Table 4.4: Creativity in F/OSS Projects

Compared to your most creative endeavor, how creative is this project?	N	Percent
Much less	55	8.16
Somewhat less	203	30.12
Equally as creative	333	49.41
Most creative	83	12.31
Total	674	100.00

Table 4.5: Responses to "Flow" questions

Ratings on "Flow" Variables	How likely to lose track of time when programming (%)	How likely to devote extra hour in the day to programming (%)
Always	21.39	12.92
Frequently	51.33	47.14
Sometimes	22.27	34.51
Rarely	4.28	4.11
Never	0.74	1.32
Total	100	100

Paid contributors rated the project creativity at a mean of 2.7 (sd = 0.80) and volunteers rated projects at 2.6 (sd = 0.8), however there was no significant difference between the means (t = 1.59, p = 0.11, two tailed test.

It may seem puzzling to nonprogrammers that software engineers feel creative as they are engaged in writing programming code. However, as Csikszentmihalyi (1975; 1990; 1996) has shown, creative tasks often cause participants to lose track of time and make them willing to devote additional hours to the task, a psychological state he calls "flow." It appears that our respondents may experience flow while engaged in programming. Table 4.5 indicates that 73 percent of the respondents lose track of time "always" or "frequently" when they are programming and more than 60 percent said that they would "always" or "frequently" dedicate one additional hour to programming ("if there were one more hour in the day").

Table 4.6 provides a ratings breakdown of the motivations to contribute to the focal F/OSS project. Respondents were asked to select up to three statements (the table shows the exact wording used in the survey) that best reflected their reasons for participating and contributing to "this" project. As discussed in the literature review, motivations can be put into three major categories: (1) enjoyment-based intrinsic motivations, (2) obligation/community-based intrinsic motivations, and (3) extrinsic motivations. We find evidence for all three types of motivations in F/OSS projects.

User needs for the software, both work- and nonwork-related, together constitute the overwhelming reason for contribution and participation (von Hippel 1988; 2001a; 2002; 2005), with more than 58 percent of participants citing them as important. But, since we asked separate questions about work- and nonwork-related user needs, we also report that 33.8 percent of participants indicated work-related need and 29.7 percent participants indicated nonwork-related need as a motive for participation. Less than 5% of respondents rated both types of user needs as important.

Table 4.6 – Motivation to contribute to F/OSS projects

Motivation	% of respondents indicating up to 3 statements that best reflect their reasons to contribute (%)	% volunteer contributors	% paid contributor	t statistic (p value)
Enjoyment based Intrinsic Motivation				
Code for project is intellectually stimulating to write	44.9	46.1	43.1	-
Like working with this development team	20.3	21.5	18.5	
Economic/Extrinsic based Motivations				
Improve programming skills	41.3	45.8	33.2	3.56 (p=0.0004)
Code needed for user need (work and/or non-work)*	58.7	-	-	-
- Work need only	33.8	19.3	55.7	10.53 (p=0.0000)
- Non-work need	29.7	37.0	18.9	5.16 (p=0.0000)
Enhance professional status	17.5	13.9	22.8	3.01 (p=0.0000)
Obligation/Community based Intrinsic Motivations				
Believe that source code should be open	33.1	34.8	30.6	
Feel personal obligation to contribute because use F/OSS	28.6	29.6	26.9	
Dislike proprietary software and want to defeat them	11.3	11.5	11.1	
Enhance reputation in F/OSS community	11.0	12.0	9.5	

N =679

Table 4.7: Project Topics By User Need

Project Topic Area*	Work Need	Non-work Need
Internet	24%	15%
Software Development	17%	10%
System	10%	10%
Multi-media	9%	18%
Office/Business	7%	2%
Communications	6%	14%
Scientific/engineering	5%	4%
Database	5%	4%
Text Editors	4%	3%
Printing	3%	0%
Games/Entertainment	2%	12%
Desktop	1%	6%
Other	6%	3%
Total Topic Counts	221	191

* Projects can be in multiple topics

To better understand the differences between work and non-work related projects I examined the self-reported project topics for all respondents that indicated a user need. Table 4.7 shows that the top three topics, comprising 44% of all topics, for non-work projects are in multi-media, communications and games/entertainment, whereas, the top three topics, comprising 52% of all topics, for work related projects are in Internet, software development and systems. Interestingly, there is quite a bit of overlap in topics between work and non-work related needs, for example, 36% of all non-work related projects are in the same topic categories as the top three topics for work related projects. An informal follow-up e-mail to 10 participants revealed that the projects initiated for non-work needs could have been equally done for work related needs in a firm setting. For example, one participant was developing special hardware device drivers for an operating system that he used at home, one can imagine a similar effort within a firm setting doing exactly the same project. Another participant had developed advanced software to display digital pictures for a family album on the World Wide Web, similar projects exist within firms to display employee picture books and customer photo albums (e.g.: AOL, MSN etc). There is therefore a blurring of distinction in the software produced for work and non-work purposes. The general purpose nature of computing and software creates conditions such that a similar user need can be high in both work and non-work settings.

The top single reason to contribute to projects is based on enjoyment-related intrinsic motivation: "Project code is intellectually stimulating to write" (44.9 percent). This result is consistent with the previous findings regarding creativity and flow in projects. Improving programming skills, an extrinsic motivation related to human capital improvement, was a close second, with 41.8 percent of participants saying it was an important motivator.

Approximately one-third of the sample indicated that the belief that "source code should be open," an obligation/community motivation, was an important reason for their participation. Nearly as many respondents indicated that they contributed because they felt a sense of obligation to give something back to the F/OSS community in return for

the software tools it provides (28.6 percent). Approximately 20 percent of the sample indicated that working with the project team was also a motivate for their contribution. Motivations commonly cited elsewhere, like community reputation, professional status, and defeating proprietary software companies (Lerner and Tirole 2002), were ranked relatively low.

Another source of an obligation/community motivation is the level of identification felt with the hacker culture. Presumably self-identification with the hacker community and ethic should drive participation in projects. Respondents to the survey indicated a strong sense of group identification with 42% indicating that they "strongly agree" and another 41% 'somewhat agree" that the "hacker" community was a primary source of their identity. Nine percent of the respondents were neutral and eight percent were somewhat to strongly negative about the hacker affiliation[71].

Table 4.6 also indicates significant differences in motivations between paid contributors and volunteers. The differences between the two groups are consistent with the roles and requirements of the two types of F/OSS participants. Paid contributors are strongly motivated by work-related user need (55.7 percent) and value professional status (22.8 percent) more than volunteers. On the other hand, volunteers are more likely to participate because they are trying to improve their skills (45.8 percent) or need the software for non-work purposes (37%).

To better understand the motivations behind participation in the F/OSS community, and the fact that no one motivation, on its own, had more than 50% importance, I decided to do a cluster analysis of the motivations to see if there were any natural groupings of the individuals by motivation types. I ran a k-means cluster analysis, with random seeding. I conducted analysis with three, four and five clusters on the cases which had reported up to three motivations for contribution. The analysis showed that the three and five cluster solutions were giving unstable and unbalanced clusters. The four cluster solution provided the best balance of cluster size and motivational aggregation

[71] The results were identical when controlling for paid contributor status on a project.

and is presented in Table 4.8. The motivations that came out highest in each cluster have been highlighted.

Cluster membership can be explained by examining the motivation categories that scored the highest in each cluster. Cluster 3 (29 percent of the sample) consists of individuals who contribute to F/OSS projects to improve their programming skills and for intellectual stimulation. None of the members of this cluster noted nonwork-related need for the project and very few, 12 percent, indicated work-related need for the code. Members of this group indicated an affinity for learning new skills and having fun in the process. The actual end product does not appear to be a large concern; both enjoyment-based intrinsic motivation and career-based extrinsic motivation are important to this group.

All members of cluster 2 (27 percent of the sample) indicate that nonwork-related need for the code is an important motive for their participation. The primary driver for this group is extrinsic user need. Similarly, cluster 1 (25 percent of the sample) represents individuals who are motivated by work-related need with a vast majority (86 percent)

Table 4.8: Cluster results based on motivations and paid status

Motivations	Cluster 1 (%)	Cluster 2 (%)	Cluster 3 (%)	Cluster 4 (%)
Work need	**91**	8	12	28
Non-work need	11	**100**	0	2
Intellectually stimulating	41	45	**69**	12
Improves skill	20	43	**72**	19
Work with team	17	16	**28**	19
Code should be open	12	22	42	**64**
Beat proprietary software	11	8	9	**19**
Community reputation	**14**	8	11	13
Professional status	**25**	6	22	18
Obligation from use	23	20	6	**83**
Paid for contribution	**86**	18	26	32
Total % of sample in each cluster	25	27	29	19

n = 679

paid for their contributions to F/OSS projects. This cluster can also be thought of as composed of people with extrinsic motivations. Cluster 4 (19 percent of the sample) consists of people motivated primarily by obligation/community based intrinsic motivations. A majority of this cluster report group identity-centric motivations derived from a sense of obligation to the community and a normative belief that code should be open.

The cluster solutions should not be viewed as definite and final. By definition, there cannot be a penultimate cluster solution. The cluster provide an exploratory view into the data to see patterns of response amongst the motivations and the motivations. Overall note that the cluster solution has revealed a "logical" grouping of individuals by their motivations, in particular observe that clusters can be classified by hybrid intrinsic/extrinsic motivations for intellectual stimulation and skill improvement, extrinsic user need motivations (both work/non-work), and obligation/community based intrinsic motivations. A clear finding from the cluster analysis is that the F/OSS community has distinct heterogeneity in motivations to participate and contribute. Individuals may join for a variety of reasons and no one reason tends to dominate the community or cause people to make distinct choices in beliefs. These findings are consistent with collective action research where group heterogeneity is considered an important trait of successful movements (Marwell and Oliver 1993).

4.6 Findings: Determinants of Effort

The literature review indicated the potential presence of three motivation types underlying sustained participation in F/OSS projects: 1)Extrinsic, 2)Enjoyment/Intrinsic and 3)Obligation-Community/Intrinsic. The findings so far have confirmed the presence of all three types of motivations with no clear and obvious determinants of effort. Do note that paid contributors work more hours. However given that there were not that many significant differences in motivations amongst paid and volunteer contributors there is still an open question regarding the effect of motivation types on effort in projects. To address the question I ran an ordinary least squares (OLS) regression on log

of hours/week[72] dedicated to the focal project to discover significant effects of motivations on project effort. The results of the regression are provided in Table 4.9.

Model 1 of the regression introduces control variables to account for hours/week dedicated to the focal F/OSS projects. Formal training in information technology (IT Training) can reduce the number of hours/week dedicated to projects as such training may provide the necessary coding background and experience that non-trained individuals may not have. An information technology related job may provide the impetus to contribute more hours as the participant may choose to leverage the F/OSS community for directly work related projects. Project founders may have an incentive to make sure that the project succeeds and thus may dedicate more hours to the project as compared to non-founding developers. Finally, the number of hours spent on other F/OSS projects may reduce the hours per week spent on the focal project. An individual may have a set quota of time for such projects and other projects may detract from working on the focal project. Model 1 shows that IT training has a marginally significant and negative impact on hours per week spent on the project. A founder will spend significantly more hours per week on the project as compared to non-founders. Time spent on other F/OSS projects will significantly (highly) reduce the hours per week spent on the focal project.

In model 2 extrinsic motivators are introduced to the regression. Now the founder effect in model 1 becomes non-significant. The only extrinsic motivation that is significant with regards to hours per week spent on a project is the survey participant's paid status. Paid developers spent significantly more time on projects as compared to non-developers. This result is consistent given the differential in hours noted between volunteers and paid developers in Table 4.3. All other extrinsic motivators like user need (work and non-work), skill improvement and seeking professional status do not impact hours per week dedicated to the projects.

[72] We chose to use log of project hours/week because of the skewness in the reported data. A log transformation allows us to better represent the effects of small changes in the data at the lower values of project hours/week. It is safe to argue that there is significant difference between 2 and 3 project hours/ week as compared to 25 and 26 project hours/week. The magnitude of the effort expended is much greater at the lower values of the measure and the log transformation allows us to capture this shift.

Table 4.9: OLS Regression on Log Project Hours/Week

Variable	Model 1			Model 2			Model 3			Model 4			Model 5		
Controls	Coef	S.E.	Sig	Coef	S.E.	Sig	**Coef**	S.E.	Sig	Coef	S.E.	Sig	Coef	**S.E.**	Sig
IT Training	-1.51	0.55	*	-1.57	0.55	*	-1.52	0.55	*	-1.24	0.55	*	-1.23	0.55	*
IT Job	0.00	0.54		-0.17	0.55		-0.29	0.55		-0.47	0.54		-0.49	0.54	
Founder	1.41	0.68	*	1.33	0.68		1.43	0.69		0.74	0.68		0.73	0.68	
Differential hours	-0.13	0.02	**	-0.14	0.02	**	-0.15	0.02	**	-0.13	0.02	**	-0.14	0.02	**
Economic/Extrinsic															
Paid Status				1.99	0.59	**	2.04	0.59	**	1.79	0.57	**	1.28	0.82	
Work need				-0.37	0.64		0.21	0.67		0.93	0.71		0.88	0.71	
Non-work need				0.03	0.61		0.49	0.67		0.84	0.70		0.85	0.70	
Improve skill				0.65	0.55		1.06	0.61		0.94	0.64		0.99	0.64	
Professional Status				0.21	0.71		0.78	0.75		0.93	0.78		0.95	0.78	
Obligation-Community/Intrinsic															
Code should be open							1.00	0.61		1.17	0.66		1.18	0.66	
Feel obligation							-0.41	0.64		0.13	0.68		0.16	0.68	
Defeat proprietary software							-0.61	0.86		-0.22	0.89		-0.19	0.89	
Enhance community reputation							1.10	0.88		1.81	0.90	*	1.82	0.90	*
Like team							1.88	0.72	**	2.07	0.75	**	2.13	0.75	**
Hacker affiliation							0.53	0.27		0.18	0.28		0.18	0.28	
Enjoyment/Intrinsic															
Intellectual stimulation										1.17	0.65		1.21	0.65	
Creative project experience										2.04	0.34	**	1.95	0.35	**
Lose track of time while programming										0.39	0.33		0.37	0.33	
Devote one more hour to programming										0.56	0.35		0.57	0.35	
Interaction Effect															
Paid Status * Creative													0.27	0.31	
Constant	-0.87	0.52		-1.70	0.70		-5.07	1.52		-13.69	2.14		-13.53	2.14	
R Square	0.07			0.09			0.11			0.18			0.18		
N	651.00			646.00			642.00			630.00			630.00		

* p <= 0.05, ** p<=0.01

Model 3 includes obligation/community based intrinsic motivators to the regression. The control and extrinsic motivators remain consistent with the previous model. The only significant variable relates to a sense of affinity with the focal project's development team. "Liking the team" has a significant and positive impact on hours per week dedicated to the project. Other obligation/community based intrinsic motivators like the belief in source code being open, feeling an obligation towards the community for using F/OSS software, antagonism towards proprietary software, enhancing community reputation and strength of the hacker identity did not have any significant effects.

The regression in model 4 includes enjoyment/intrinsic motivators. The control and extrinsic motivators have the same impact as model 3. Along with liking the team, enhancing community reputation gains marginal positive significance in the obligation/community intrinsic motivation category. Self-assessment of a creative project experience has the largest and most significant positive impact on hours per week dedicated to a F/OSS project. Individuals that report a high sense of creativity on the project spend significantly more time on the project than those that do not. Also note that the percentage of variance improved greatly between models 3 and 4. Introduction of enjoyment related motivators caused the R^2 value to go form 11% to 18%. Other motivators like intellectual stimulation and flow related variables like losing track of time while programming and devoting one more hour to programming are not significant.

Table 4.10 shows the relative impact of each of the significant variables in model 4. I standardized[73] all the variables in model 4 of the regression to allow for direct comparisons of the magnitudes across variables. Personal sense of creativity on a F/OSS project has the largest positive impact on hours per week amongst all the variables. Being paid to write code and liking the team have significant positive effects that are approximately half the size as a sense of creativity. Caring about reputation in the F/OSS community has about one third an impact as feeling creative in a project. Hours dedicated to other F/OSS projects have an equal and negative impact as creativity on

[73] All variables transformed so that the mean = 0 and the variance = 1.

current project. We can see that various F/OSS projects compete for time and distractions from other projects can reduce the overall number of hours spent on the focal project. Having formal IT training also reduces the number of hours spent on a project.

As mentioned in the literature review, proponents of intrinsic motivation theories have assembled an impressive array of experimental evidence (Deci, Koestner and Ryan 1999) to demonstrate that extrinsic rewards have a negative impact on intrinsic motivations. An obvious test in the study would be to see the impact of the interaction between being paid and feeling creative in a project on the number of hours per week dedicated. Model 5 in Table 4.9 provides the results of the test. There is no significant impact on the hours per week dedicated based on the interaction of being paid and feeling creative. Hours per week dedicated to a project do not decline given that those that are paid to contribute code are also feeling creative in that project.

Researchers engaged in studying creativity have traditionally used third-party assessments of innovative output as measures for creativity. Thus the findings regarding a sense of personal creativity as the biggest determinant of effort in F/OSS projects may have occurred due to the inherent innovativeness of the project itself and not based on any personal feelings of being creative. Since there are multiple responses from many projects a test if creativity the felt is endogenous to the project or to the individual can be utilized. Results from a fixed-effects regression (Greene 2000) on the key variables are presented in Table 4.11. A fixed effect regression holds constant the project and determines within project effects of the variables being tested. The results should be read to indicate the significance of each variable within projects. Creativity in a project is still positive and significant in the regression, indicating that the sense of creativity is endogenous and heterogeneous to the people within projects. Concern for gaining a reputation within the F/OSS community is also positive and significant. Interestingly, paid status in the fixed effects regression is not significant. This indicates that within projects being paid does not impact hours/week dedicated.

Table 4.10: Impact of Significant Variable on Project Hours/Week (Standardized Coefficients)

Variable	Coeff.	t-statistic (p-value)
Creative project experience	1.6	6.00 (0.000)
Paid status	0.88	3.12 (0.002)
Like team	0.84	2.76 (0.004)
Enhance community reputation	0.56	2.00 (0.046)
Differential hours	-1.6	-6.00 (0.000)
IT training	-0.6	-2.28 (0.023)

Table 4.11: Fixed Effects OLS Regression on Log Project Hours/Week

	Coef	S.E.	Sig
Controls			
IT Training	-1.22	0.72	
IT Job	-0.65	0.74	
Founder	1.48	1.03	
Differential hours	-0.12	0.03	**
Economic/Extrinsic			
Paid Status	1.31	0.81	
Work need	-0.06	0.95	
Non-work need	0.20	0.94	
Improve skill	0.00	0.86	
Professional Status	-1.14	1.00	
Obligation-Community/Intrinsic			
Code should be open	0.34	0.85	
Feel obligation	-0.36	0.86	
Defeat proprietary software	-0.26	1.16	
Enhance community reputation	2.80	1.21	**
Like team	0.05	1.03	
Hacker affiliation	0.00	0.37	
Enjoyment/Intrinsic			
Intellectual stimulation	0.22	0.84	
Creative project experience	2.27	0.46	**
Lose track of time while programming	0.49	0.45	
Devote one more hour to programming	0.54	0.44	
F Statistic	3.72		**
Constant	-11.33	2.78	
R Square	0.17		
N	511.00		

4.7 Discussion and Conclusion

The most important findings in the study relate to both the extent and impact of the personal sense of creativity developers feel with regard to their F/OSS projects. A clear majority (more than 61 percent) stated that their focal F/OSS project was at least as creative as anything they had done in their lives (including other F/OSS projects they might have engaged in). This finding is bolstered by the willingness of a majority of survey participants to dedicate additional hours to programming, and, consistent with attaining a state of flow, frequently losing track of time while coding. These observations are reinforced by the similar importance of these creativity-related factors for both volunteer and paid contributors.

The importance of the sense of creativity in projects is underscored by examination of the drivers of effort in F/OSS projects. The only significant determinants of hours per week dedicated to projects were (in order of magnitude of impact):

- Enjoyment-related intrinsic motivations in the form of a sense of creativity
- Extrinsic motivations in form of payment
- Obligation/community-related intrinsic motivations

Furthermore, contrary to experimental findings on the negative impact of extrinsic rewards on intrinsic motivations (Deci, Koestner, and Ryan 1999), I find that being paid and feeling creative about F/OSS projects does not have a significant negative impact on project effort.

Therefore, work on the F/OSS projects can be summarized as a creative exercise leading to useful output, where the creativity is a lead driver of individual effort. Programming has been regarded as a pure production activity typified as requiring payments and career incentives to induce effort. However, this maybe a limited view. At least as applied to hackers on F/OSS projects, activity should be regarded as a form of joint production-consumption that provides a positive psychological outlet for the participants as well as useful output.

Another central issue in F/OSS research has been the motivations of developers to participate and contribute to the creation of a public good. The effort expended is

substantial. Individuals contribute an average of 14 hours per week. But there is no single dominant explanation for an individual software developer's decision to participate in and contribute to a F/OSS project. Thus an interplay between extrinsic and intrinsic motivations: neither dominates or destroys the efficacy of the other. It may be that the autonomy afforded project participants in the choice of projects and roles one might play has "internalized" extrinsic motivations.

Therefore, an individual's motivation containing aspects of both extrinsic and intrinsic is not anomalous. Dominant motives do not crowd out or spoil others. It is consistent for someone paid to participate in the F/OSS movement to be moved by the political goals of free software and open code.

Other issues merit further investigation. The presence of paid participants, 40 percent of the sample, indicates that both IT-producing and IT-using firms are becoming important resources for the F/OSS community. The contribution of firms to the creation of a public good raises questions about incentives to innovate and share innovations with potential competitors. In addition, the interaction between paid and volunteer participants within a project raises questions about the boundaries of the firm and appropriate collaboration policies.

In conclusion, this chapter has advanced understanding of the motivational factors behind the success of the F/OSS community. Findings indicate that the F/OSS community does not require any one type of motivation for participation. It is a "big tent." Its contributors are motivated by a combination of intrinsic and extrinsic factors with a personal sense of creativity being an important source of effort.

References

Amabile, Teresa M. 1996. *Creativity in context*. Boulder, CO: Westview Press.

Csikszentmihalyi, Mihaly. 1975. *Beyond Boredom and Anxiety: The Experience of Play in Work and Games*. San Francisco: Jossey-Bass, Inc.

Deci, Edward L, R Koestner, and Richard M Ryan. 1999. "A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation." *Psychological Bulletin* 125:627-688.

Deci, Edward L, and Richard M Ryan. 1985. *Intrinsic motivation and self-determination in human behavior*. New York, NY: Plenum Press.

Franke, Nikolaus, and Sonali Shah. 2003. "How communities support innovative activities: an exploration of assistance and sharing among end-users." *Research Policy* 32:157-178.

Frey, Bruno. 1997. *Not just for the money: an economic theory of personal motivation.* Brookfield. VT: Edward Elgar Publishing Company.

Ghosh, Rishab Ayer, Ruediger Glott, Bernhard Krieger, and Gregorio Robles. 2002. "Free/Libre and Open Source Software: Part IV Survey of Developers." International Institute of Infonomics, University of Maastricht.

Greene, William H. 2000. *Econometric Analysis*. Upper Saddle River, NJ: Prentice- Hall Inc.

Hars, Alexander, and Shaosong Ou. 2002. "Working for free? Motivations for participating in Open-Source projects." *International Journal of Electronic Commerce* 6:25-39.

Hertel, Guido, Sven Niedner, and Stefanie Herrmann. 2003. "Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel." *Research Policy* 32:1159-1177.

Himanen, Pekka. 2001. *The Hacker Ethic and the Spirit of the Information Age*. New York: Random House.

Holmström, Bengt. 1999. "Managerial Incentive Problems: A Dynamic Perspective." *Review of Economic Studies* 66.

Koch, S, and G Schneider. 2002. "Effort, Cooperation and Coordination in an Open Source Software Project: GNOME." *Information Systems Journal* 12:27-42.

Kollock, Peter. 1999. "The Economies of Online Cooperation." Pp. 220-239 in *Communities in Cyberspace*, edited by Peter Kollock and Marc A. Smith. New York, NY: Routledge.

Lancashire, David. 2001. "Code, culture and cash: The fading altruism of Open Source development." *First Monday* 6.

Lerner, Josh, and Jean Tirole. 2002. "Some Simple Economics of Open Source." *Journal of Industrial Economics* 50:197-234.

Lindenberg, Siegwart. 2001. "Intrinsic motivation in a new light." *Kyklos* 54:317-342.

Markus, Lynne M, Brook Mannvile, and Carole E Agres. 2000. "What make a virtual organization work? Lessons from the open-source world." *Sloan Management Review* Fall 2000, 42:13-26.

Marwell, Gerald, and Pamela Oliver. 1993. *The Critical Mass in Collective Action: A Micro-Social Theory*. Cambridge, UK: Cambridge University Press.

McGowan, David. 2001. "The Legal Implications of Open Source Software." *Illinois Law Review*.

Moody, Glen. 2001. *Rebel Code: Inside Linux and the Open Source Revolution.* New York: Perseus Press.

Morrison, Pamela D, J H Roberts, and Eric von Hippel. 2000. "Determinants of user innovation and innovation sharing in a local market." *Management Science* 46:1513-1527.

Nakamura, Jeanne, and Mihaly Csikszentmihalyi. 2003. "The construction of meaning through vital engagement." in *Flourishing: positive psychology and the life well-lived*, edited by Corey L Keyes and Jonathan Haidt. Washington, DC: American Psychological Association.

Raymond, Eric. 1996. *The New Hacker Dictionary - 3rd Edition.* Cambridge, MA: MIT Press.

—. 1999. *The Cathedral and the Bazaar: Musings on Linux and Open Source from an Accidental Revolutionary.* Sebastopol: CA: O'Reilly and Associates.

Riggs, William, and Eric von Hippel. 1994. "Incentives to innovate and the sources of innovation: The case of scientific instruments." *Research Policy* 23:459-469.

Ryan, Richard M, and Edward L Deci. 2000. "Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions." *Contemporary Educational Psychology* 25:54-67.

Stallman, Richard. 1999. "The GNU Operating System and the Free Software Movement." Pp. 53-70 in *Open Sources: Voices from the Open Source Revolution*, edited by C DiBona, S Ockman, and Mark Stone. Sebastopol, CA: O'Reilly.

Torvalds, Linus, and David Diamond. 2001. *Just for fun: the story of an accidental revolutionary.* New York, NY: HarperCollins.

Urban, Glen L, and Eric von Hippel. 1988. "Lead User Analyses for the Development of New Industrial Products." *Management Science* 34:569-582.

von Hippel, Eric. 1988. *The Sources of Innovation.* New York, NY: Oxford University Press.

—. 2001. "Innovation by User Communities: Learning from Open Source Software." *Sloan Management Review* 42:82-86.

—. 2005. *Democratizing Innovation.* Cambridge, MA: MIT Press.

von Hippel, Eric, and Georg von Krogh. 2003. "Open Source Software and the Private-Collective Innovation Model: Issues for Organization Science." *Organization Science* 14:209-223.

von Krogh, Georg, Sebastian Spaeth, and Karim R Lakhani. 2003. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study." *Research Policy* 32:1217-1241.

Wayner, Peter. 2000. *Free For All: How Linux and the Free Software Movement Undercuts the High-Tech Titans.* New York: HarperBusiness.

Appendix – Survey Instrument Used

BCG Free/Open Source software survey

Section 1 - Motivation, inspiration and leadership

1 2 3 4 5 6
Section 1 of 6

Why do you contribute to Free/Open Source projects? This information will help us to understand what holds the community together. When we use "this project" in the following questions, we mean to imply the Free/Open Source project referenced in the email you received from us.

1) Please select up to 3 statements that best reflect your reasons for contributing to this project (i.e., the Free/Open Source project referenced in the cover email). Choose fewer than 3 statements if the 1 or 2 you do select cover all of your significant motivations.

- ☐ a) My contribution creates specific functionality in the code needed for my work.
- ☐ b) My contribution creates specific functionality in the code needed for my non-work life.
- ☐ c) The code for this project is intellectually stimulating to write.
- ☐ d) My activity on this project improves my programming skill.
- ☐ e) I like working with the development team on this project.
- ☐ f) I believe source code should be open.
- ☐ g) I dislike proprietary software or the companies that produce it and want to help the Free/Open Source software community defeat them.
- ☐ h) My contributions will enhance my reputation in the Free/Open Source software community.
- ☐ i) My contributions will enhance my professional status.
- ☐ j) I feel a personal obligation to contribute since I use Free/Open Source software
- ☐ k) The license for this project forces me to contribute my changes.
- ☐ l) Other 1, please specify: _____
- ☐ m) Other 2, please specify: _____
- ☐ n) Other 3, please specify: _____

2) Did you know anyone from this project prior to your participation in it?
Please select

3) If you knew members of the project team before, was your decision to participate influenced by knowing these people?
Please select

4) Please indicate the most important way one gains or enhances his or her reputation in the Free/Open Source software community
Please select
If other, please specify _____

5) Imagine a time in your life when you felt most productive, creative or inspired. Comparing your experience on this project with the level of creativity you felt then, this project is:
Please select

6) Please indicate up to 3 important ways in which project leaders can enhance Free/Open Source projects.

- ☐ a) Create a plausible promise for the project (vision)
- ☐ b) Create the initial code base for the project
- ☐ c) Continue to contribute code through the duration of the project
- ☐ d) Determine the appropriate (programming & other) tasks for the project
- ☐ e) Delegate appropriate (programming & other) tasks for the project
- ☐ f) Integrate various submissions
- ☐ g) Initiate constructive dialogue with the developer community on project issues
- ☐ h) Open minds to alternative approaches

- [ ] i) Providing motivation
- [ ] j) Help people get started
- [ ] k) Provide specific help or responses to questions
- [ ] l) Manage the timing of project contributions
- [ ] m) Recruit additional project contributors
- [ ] n) Not much impact
- [ ] o) Other 1, please specify
- [ ] p) Other 2, please specify
- [ ] q) Other 3, please specify

Proceed to the next section

BCG Free/Open Source software survey

Section 2 - Attitudes and views

1 2 3 4 5 6
Section 2 of 6

How do you perceive and identify with the world of coding and programming? Your responses will help us to identify patterns among programmers.

1) If there were 1 more hour in the day, I would devote it to programming.
Please select

2) Hackers (as in the linked definition from Eric Raymond) are a primary community with which I identify.
Please select

3) When I program, I lose track of time.
Please select

4) The following are statements related to hacking and the Free/Open Source software community. To focus your attention on the issues in which we are most interested, some of the quotes have been edited and none have been attributed. Please select up to 3 statements with which you most strongly agree and up to 3 with which you most strongly disagree. Select only those statements about which you feel strongly (i.e. could be fewer than 3).

	Statement	Strongly agree (max of 3)	Strongly disagree (max of 3)
a	"With enough eyeballs, all bugs are shallow."	☐	☐
b	"We reject kings, presidents, and voting. We believe in rough consensus and running code."	☐	☐
c	"Free software matters because all freedoms matter, and software happens to be the domain in which I can contribute most."	☐	☐
d	"Free software is a matter of liberty, not price. Think of 'free' as in 'free speech,' not as in 'free beer.'"	☐	☐
e	"When we prepare a program, the experience can be just like composing poetry or music."	☐	☐
f	"The classic hacker life: Feats of virtuoso coding that ignore minor irritations like times of day or night; sleeping on the floor when exhaustion finally wins out over inspiration; the countless Chinese meals; the heated conversations; the love of word-play; the pranks.	☐	☐
g	"Open Source is subversive."	☐	☐
h	"When you lose interest in a program, your last duty is to hand it off to a competent and eager successor."	☐	☐
i	"The next best thing to having good ideas is recognizing good ideas from others. Sometimes the latter is better."	☐	☐

Proceed to the next section

BCG Free/Open Source software survey

1 2 3 4 5 6

Section 3 - Your Free/Open Source software project contributions

Section 3 of 6

We're interested in past, present and future contributions. This information will help us understand how individuals contribute to the community over time. When we use "this project" in the following questions, we mean to imply the Free/Open Source project referenced in the email you received from us.

1) When did you first contribute to a Free/Open Source software project?
(contribute = create, debug, patch, extend program, document or provide extensive user support)
Please select

2) To how many Free/Open Source software projects do you currently contribute? [ ] # of projects

3) To how many Free/Open Source software projects have you contributed in total? [ ] # of projects

4) For how many months have you been working on this project? [ ] # of months

5) What is your primary role on this project?
Please select
If other, please specify: [ ]

6) Please rank the following as sources of code for your project. [1=most important, 3=least important. "Don't know" is also an answer choice for each option]"
a) Project team members developing original code
Please Select
b) Project team members 'borrowing' compatible licensed code
Please Select
c) Project users contributing code
Please Select

7) Have you been financially compensated in any way for participating in this project?
Please select

8) How likely would you be to contribute without compensation to a Free/Open Source software project that delivers value primarily to users other than you or your peer group?
Please select

9) Under what circumstances would you work on a closed-source software project? Select all that apply:
If it would lead to significant advances in software development
If it would make me famous in the software world
If it would pay me enough to support my lifestyle
Other reason, please specify [ ]

10) Have your contributions to the Open Source community had a substantial impact on you attaining any of the following? (Please indicate all that apply in the first column and then select the one of those that has been most important for you in the second column.)

Benefit	All that apply	Most important (only one)
New job offer(s)	☐	○
Paid consulting opportunities	☐	○

Job promotion(s) (in current job)	☐	○
Cash reward for work done in the Sourceforge community	☐	○
Stock options or other claims (realized or not) on a company's future performance	☐	○
Increased personal knowledge base	☐	○
Improved reputation in the Sourceforge community	☐	○
Improved reputation in professional arena (outside of the Sourceforge community)	☐	○
Personal sense of accomplishment for contributions	☐	○
Personal sense of connection in the Sourceforge community	☐	○
Other 1, please specify	☐	○
Other 2, please specify	☐	○
Other 3, please specify	☐	○

11) Have your contributions to the Open Source community cost you any of the following? (Please indicate all that apply in the first column and then select the one of those that has been worst for you in the second column.)

Cost	All that apply	Worst (only one)
Time to make money	☐	○
Social time	☐	○
Social relationship(s)	☐	○
Professional/career advancement	☐	○
Academic performance	☐	○
Sleep	☐	○
Stress/health	☐	○
Cost of hardware, software, or bandwidth dedicated to Sourceforge development	☐	○
Other 1, please specify	☐	○
Other 2, please specify	☐	○
Other 3, please specify	☐	○

Proceed to the next section

BCG Free/Open Source software survey

Section 4 - How you spend your time

How much time and at what location(s) do you contribute to Free/Open Source projects? This will help us to understand different patterns of involvement in this community. When we use "this project" in the following questions, we mean to imply the Free/Open Source project referenced in the email you received from us.

1) How many hours in the past week have you spent on this project? [ ] hours

2) How many hours per week do you spend on this project at the following locations? Please select:

· work [ ] hours/week
· home [ ] hours/week
· school [ ] hours/week
· other [ ] hours/week

if other, please specify location
[ ]

3) How many hours in the past week have you spent on all your current Free/Open Source software projects?
[ ] hours

4) On how many project(s) have you spent >10 hours in the past week? [ ] projects

Questions 5-7 pertain to how your Free/Open Source software contributions relate to your current job. We recognize the potential sensitivity in some workplaces to issues raised in the following questions and reiterate that all responses are confidential and will be reported in aggregate only.

5) If you spend work time on this project, is your direct supervisor aware of this?
Please select

6) If s/he is aware of this, does s/he regard this allocation of work time to this project as part of your core job?
Please select

7) If your work on this Free/Open Source software project is part of your core job, please select the most appropriate explanation:
Please select

8) Since your initial involvement, how has the amount of time you spend contributing to all Free/Open Source projects changed?
Please select

Proceed to the next section

BCG Free/Open Source software survey

Section 5 - About you

1 2 3 4 5 6
Section 5 of 6

Please tell us a bit about you so we can better understand the spectrum of participants in the Free/Open Source community. We reiterate that all responses are confidential and will be reported in aggregate only.

1) What is your gender?
Please select

2) What is your year of birth?

3) In what country do you reside?
Please select

4) In what greater metropolitan area do you work or attend school? (leave blank if you live in a rural area)

5) What is your current occupation?
Please select
if other, please specify:

6) How many years have you been programming? yrs.

7) Have you been formally trained in programming?
Please select

Proceed to the next section

BCG Free/Open Source software survey

Section 6 - Views on the future of the Open Source movement

1 2 3 4 5 6
Section 6 of 6

1) What 3 elements are most crucial to the success of the Free/Open Source software movement/community?

a)
b)
c)

2) What 3 issues most threaten the Free/Open Source software movement/community?

a)
b)
c)

3) In view of your responses to questions 5 & 6, how do you see the future of the Free/Open Source software movement/community?

Would you like to participate in our gift certificate draw? ( ) yes ( ) no

Would you like to receive our analysis? ( ) yes ( ) no

If we have further questions, may we contact you via e-mail? ( ) yes ( ) no

What is your preferred e-mail address?

Submit survey

END of Survey

Chapter 5 Summary and Conclusion

The purpose of this dissertation has been to examine how distributed and self-organizing innovation systems operate. In particular I have focused on the differing roles of core and peripheral participants in the distributed innovation process and how they jointy contribute towards problem solving effort. Overall I find that heterogeneity in information, knowledge and use is an important component of a distributed innovation system. "Traditional" innovation processes emphasize an external search for heterogeneous information by the problem holder and once found it is brought inside the organization for problem resolution. In contrast, distributed innovation systems demonstrate a variety of ways in which internal and external sources of knowledge can be integrated.

The first study showed that unsolved scientific problems inside the R & D labs of firm can be solved by a widely distributed population of individuals who may not be known to the firm. In this system, the laboratory, i.e. the core, poses the problem and someone not known to the firm solves it. The second study showed an innovation process where the peripheral members of an open source software community pose a majority of the problems and these are solved through a collective core-periphery problem solving effort. Interestingly, the periphery solves a majority of the problems that add novel functionality to the community's software output. The third study examined motivations of core contributors in open source communities to participate. Enjoyment-based intrinsic motivation, namely how creative a person feels when working on the project, is the strongest and most pervasive driver of effort by core contributors. In addition user need, intellectual stimulation derived from writing code, and improving programming skills are top motivators for project participation. Similar motivations were also reported by peripheral participants in the first study.

In this chapter I summarize the results of the three studies and then link the findings to extant literature with the aim towards developing an understanding of the possible generality and robustness of this form of innovating.

5.1: Overview of the empirical studies

5.1.1: Study 1 – "Broadcast Search and Solution Finding from the Periphery"

The first study explores an alternative mechanism of scientific and technological problem solving that focuses on solution generation from a wide range of dispersed and peripheral problem solvers. It is based on the von Hayek's (1945) central insight that knowledge is unequally and widely distributed amongst individuals and the central challenge in society is to find ways to access this knowledge.

I find that innovative solutions to difficult scientific and technical problems can be effectively identified by broadcasting problems to a large group of diverse solvers in different fields and providing incentives for external solvers to solve them. Broadcasting problems to a group of diverse solvers is a radical departure from traditional problem solving search as it inverts the typical problem solving process by focusing the efforts of the problem holders (e.g. R&D labs) into attracting solutions from external solvers instead of creating solutions themselves and that it allows for the mitigation of some of the negative issues (e.g.: competency traps, excessive reliance on existing knowledge) associated with "local search" (March & Simon, 1958; Nelson & Winter, 1982; Podolny, Stuart, & Hannan, 1996).

I analyzed 166 discrete life sciences and chemistry and applied science problems that originating in the R &D labs of 26 firms from 10 countries. These problems were broadcasted to a network of 80,000 independent solvers, for a financial reward for successful solution, from over 150 countries via InnoCentive.com, an independent subsidiary of Eli Lilly. The analysis shows that the broadcast search method yields a 29.5% solving rate for problems that well-renowned and large R & D intensive firms had not been successful in solving themselves.

Table 5.1 shows the probit regression results of the likelihood of a problem being solved as a function of its characteristics (e.g.: solution requirement (RTP or paper), award size, time window to solve problem) and the characteristics of the solver base

(e.g.: total number of potential solvers, number of solution submissions, diversity of scientific interests and generalist/specialist orientation of solvers) that each problem attracted. I find that the problems that can attract a solver base with more heterogeneous scientific interests the more likely they will be solved. I also find that the average number of scientific interests per solver per problem is significantly and negatively correlated with solvability, implying that problems which attract relatively more specialized solvers, i.e. those that express fewer scientific interests, are more likely to be solved. This indicates that the problems that end up being solved are able to attract specialists from different fields.

Table 5.1 also shows that the number of days a particular problem is open for resolution is negatively and significantly correlated with problem solvability. The number of days a problem is open is an indication of problem complexity as assessed by the seeker firm. Thus, controlling for all other variables, the more days a problem is open, the more complex it is, and the less likely it is to be solved. Since broadcast search is a nontraditional method of problem solving, research and development labs inside firms may learn over time how to become better at broadcast searches. I measured seeker learning by counting the number of previous problems a firm had previously broadcasted with IC. The results show a marginally positive effect of seeker learning. That is, firms are learning over time how to select and articulate appropriate problems for heterogeneous solvers.

A web-based survey of 370 solvers indicated that 72.5% of the winning solvers based their submissions, partially or fully, on previously developed solutions from their own and/or someone else's work. Solutions to problems that firms seek to solve encounter their resolution in already existing (complete or partial) solutions in the distributed solvers' domains, which are then reused or transformed, showing that broadcast search effectively utilizes existing distributed knowledge.

Table 5.2 shows that the probability of being a winning solver is significantly correlated with both a desire to win the award money as well as intrinsic motivations like

enjoying problem solving and being intellectually challenged. However, even though there was a substantial prize award for creating the best solution, the effect of intrinsic motivation is stronger and more significant. I also find that individuals who are successful problem solvers report that the broadcasted problem was at the boundary or outside their field of expertise. This has a positive and significant effect in predicting who becomes a winning solver and may be due to the ability of "outsiders" from relatively distant fields to see problems with fresh eyes and apply solutions that are novel to the problem domain. Importantly, it implies that problem broadcasting to solvers in diverse fields triggers productive cross-fertilization of knowledge bases between scientific disciplines.

Consistent with the finding about specialization (Table 5.1), I find a significant negative correlation between the number of scientific interests expressed and the probability of being a winning solver. Thus, on the margin, being more specialized (expressing fewer scientific interests) results in a higher probability of creating a winning solution. In this context, specialists from outside the problem domain bring knowledge and solutions from their own "distant" domains to create winning solutions.

The broadcast search methodology draws on distributed knowledge for solving problems and changes the problem holder's task of problem solving and solution creation to that of defining the problem and attracting a large and diverse pool of capable problem solvers. Problems which attract submissions from a solver base with a more diverse set of scientific interests are more likely to be solved. Problems which attract, on average, more specialized solvers are also more likely to be solved. This thus indicates that in broadcast search, the central characteristic of readily solved problems is the seeker firm's ability to articulate problems in a way that attracts specialists from different fields. Further, solutions to problems that firms' seek to solve encounter their resolution in already existing (partial) solutions in the distributed solvers' domains. Problems are more likely to be solved by solvers who report themselves to be experts in fields outside or at the boundary of the problem field. Thus, broadcast search may not only effectively reuse knowledge, but also transfer and transform (*25*) knowledge from one field to others.

Table 5.1: Probit Regression on Problem Being Solved
(N=132 Problems)

	Coefficient	Robust Standard Error
Problem Characteristics		
RTP Problem Type	0.305	0.224
Award Value	-0.256	0.247
Days Problem Open	**-0.956****	0.278
Seeker Firm Experience		
Previous problems posted by seeker firm	**0.362†**	0.204
Solver community		
Solver base size	-1.044	0.637
Number of submissions	0.041	0.175
Types of Solvers Attracted		
Distinct scientific interests attracted	**1.260****	0.367
Generalist orientation of solvers	**-0.915****	0.326
Log Pseudolikelihood	-50.52	
Wald's Chi Square	57.02***	
Df	19	
Pseudo R Square	0.39	

† significant at 10%; * significant at 5%; ** significant at 1%; *** significant at 0.1%
Controlled for year effects and scientific disciplines of problem

Table 5.2: Probit Analyses Predicting Which Solver Submits A Winning Solution (N=295 Respondents)

	Coefficient	Robust Standard Error
Expertise		
Interest count (at registration)	**-0.179†**	0.092
Problem distance from field of expertise	**0.209***	0.104
Motivations		
Win award money	**0.263***	0.107
Social and work related motivations	**-0.243***	0.112
Intrinsic motivations	**0.374****	0.115
Beating other solvers	**-0.206†**	0.117
Unsatisfactory job	-0.033	0.134
Had free time	**0.284***	0.119
Interaction Effect For Motivations		
Money X Intrinsic	-0.025	0.120
Control Variables		
RTP Problem Type	0.188	0.233
Time to develop solution	**0.002***	0.001
Log Pseudolikelihood	-85.27	
Wald's Chi Square	32.37***	
Df	10	
Pseudo R Square	0.16	

† significant at 10%; * significant at 5%; ** significant at 1%; *** significant at 0.1%

5.1.2 Study 2 – "The Primacy of the Periphery in Open Source Software Development"

The second study examines the problem solving and software development process in an open source community to gain insight into the differing roles of core and periphery community members. Empirical studies of OSS projects have shown participation of hundreds of individuals in a core-periphery community structure with a relatively small number of core members contributing most of the software code and dominating the technical discussions (Koch & Schneider, 2002; Lee & Cole, 2003; von Krogh, Spaeth, & Lakhani, 2003). Using both quantitative and qualitative analyses, my study examines the value of the peripheral members to the software development effort and the community work practices that enable core-periphery integration.

I define core members as those who have full access to make official changes to the community's source code repository and periphery as those community members who do not. I analyzed a one year period, from November 2002 to November 2003, of software development activity in the PostgreSQL (PG) database community by creating an analytic tool called an "innovation process history." This consisted of matching 241 concrete software features to 2,402 changes in the software source code repository and 20,129 email messages exchanged between 798 individuals.

My quantitative analysis shows the primacy of peripheral participants in the PG community. As Table 5.3 shows, periphery members are responsible for developing a majority of the functionally novel innovations in the community. While core members concentrate on developing dimension of merit (performance) features. Beyond writing novel code, as table 5.4 shows, peripheral members play a critical role in the collective problem solving effort by initiating the majority of development activity in the community and byproviding critical solution and, use information during the development process.

Overall I find that the periphery members are essential to the community in three ways: they provide new use information about the product, they provide unique pre-made solutions and they provide solution-related information. Practically, periphery members

initiate a majority of code writing episodes; they write about half the code; they participate in collective problem solving with other code writers; they test and evaluate newly written code; and they are a source of new core members. More importantly they are the primary source of *new needs and solutions* adopted by the communities and that ongoing core and periphery interactions are critical to community success. This finding makes sense. Organizational research on the nature of knowledge has shown that it is both sticky and leaky (Brown & Duguid, 2001). Existing research has shown that when knowledge is totally sticky and distributed, functionally novel innovation – involving novel need information - will occur primarily in the periphery (Riggs & von Hippel, 1994; Tyre & von Hippel, 1997; von Hippel, 2005). When knowledge is totally non-sticky, problem-solving might occur only in the core. However another perspective on knowledge is to emphasize that it is not an object but rather "knowing in practice" (Orlikowski, 2002). Thus core and periphery interactions are not about just about transferring knowledge or the degree of stickiness of knowledge, but instead they represent the practice of sharing "know how" amongst the participants.

I show that ongoing interactions between core and peripheral members are the primary driver of problem solving and knowledge creation in the OSS community. Specifically, as Table 5.5 shows, the following six set of work practices enable core and periphery members to produce software in a distributed and virtual setting: 1) *work broadcasting; 2) building and using community memory; 3) distributed decision making, 4) choosing type and level of participation, 5) using the community's output;* and *6) coordinating action and building through evidence.* Using vignettes from the innovation process histories, I demonstrate that these practices are not separate activities from the work of software development itself. Rather, they are embodied in the way these communities produce software, and are at the heart of collective problem solving in a distributed innovation setting.

Table 5.3: Feature Type Author by Member Status [74]

	Member Status		
Innovation Categorization	**Core**	**Periphery**	**Total**
Functionally Novel (Adaptive)	27 (38%)	44 (62%)	71
Dimension of Merit (Corrective and Perfective)	105 (62%)	65 (38%)	170
Total	132 (55%)	109 (45%)	241

Table 5.4 : Role of Periphery in Community Problem Solving

Actions Taken by the Peripheral Participants	**% of Changes**
Triggered feature development	70
Confirmed need information	59
Helped formulate exact problem	58
Proposed solution information	51
Provided information used in final solution	42

N = 241

[74] Author here refers to the person given credit on the PG release note.

Table 5.5: Practices for Community-Based Distributed Problem Solving

Practices	Activities	How they enable distributed problem solving
	Collective Practices	
Work Broadcasting	Posing issues and problems Providing ideas and solution Contributing code Providing feedback	Mobilize community Create, transfer and transform knowledge
Building and using community memory	Global and local archiving of community activity Retrieval and use of community activity	Build mutual understanding over time and space
Distributed decision making	Incremental and local planning Provisional settlements Lazy consensus	Preserving choices and making choices in the community
	Individual Practices	
Choosing type and level of participation	Identifying tasks Doing tasks	Doing the actual work of the community
Using the community's output	Using latest code Integrating latest code	Connects individuals to community Testing community output
Coordinating action and building trust through evidence	Providing evidence of claims Seeking evidence of claims Testing evidence of claims	Enable self and others to assess claims and outputs Share and assess knowledge

5.1.3 Study 3 – "Motivations of Core Developers to Contribute to Open Source Projects"

The third study examined the motivation and effort of core participants in OSS communities. "What drives OSS developers to contribute their time and effort to the creation of free software products?" is an often posed question by software industry executives, managers, and academics when they are trying to understand the relative success of OSS communities. Many people are puzzled by what appear to be irrational and altruistic behavior by movement participants: giving code away, revealing proprietary information, and helping strangers solve their technical problems. I used a web-based survey, administered to 684 software developers in 287 OSS projects, to learn what lies behind the effort put into such communities.

Core members were contributing on average 14 hours per week to F/OSS projects with about 40% being sponsored or paid by their employers to participate. Table 5.6 provides data on the core member's motivation to contribute to OSS projects. I find that user need (von Hippel, 1988), intellectual stimulation (Nakamura & Csikszentmihalyi, 2003) derived from writing code, and improving programming skills (Lerner & Tirole, 2002) are the top motivators for project participation and contribution. Interestingly motivations like an ideological belief in defeating proprietary software or personal gain like trying to enhance professional and/or community reputation were not widely reported.

A significant findings in the study relates to both the extent and impact of the personal sense of creativity core members feel with regard to their F/OSS projects. A clear majority (>61%) stated that their focal F/OSS project was at least as creative as anything they had done in their lives (including other F/OSS projects they might engage in). This finding was bolstered by the willingness of a majority of survey participants willingness to dedicate additional hours to programming and, consistent with a state of flow (Csikszentmihalyi, 1975), the observation of frequently losing track of time while programming. Academic theorizing on individual motivations for participating in OSS projects has posited that external motivational factors in the form of extrinsic benefits

(e.g.; better jobs, career advancement) as the main drivers of effort (Lerner et al., 2002). In contrast, as table 5.7 shows, enjoyment-based intrinsic motivation (Csikszentmihalyi, 1975; Deci & Ryan, 1985; Frey, 1997; Lindenberg, 2001), namely how creative a person feels when working on the project (Amabile, 1996), was the strongest and most pervasive driver of effort (hours per week devoted to a project).

Table 5.6: Motivation to contribute to F/OSS projects

Motivation	% of respondents indicating up to 3 statements that best reflect their reasons to contribute (%)	% volunteer contributors	% paid contributor	t statistic (p value)
Enjoyment based Intrinsic Motivation				
Code for project is intellectually stimulating to write	44.9	46.1	43.1	-
Like working with this development team	20.3	21.5	18.5	
Economic/Extrinsic based Motivations				
Improve programming skills	41.3	45.8	33.2	3.56 (p=0.0004)
Code needed for user need (work and/or non-work)*	58.7	-	-	-
- Work need only	33.8	19.3	55.7	10.53 (p=0.0000)
- Non-work need	29.7	37.0	18.9	5.16 (p=0.0000)
Enhance professional status	17.5	13.9	22.8	3.01 (p=0.0000)
Obligation/Community based Intrinsic Motivations				
Believe that source code should be open	33.1	34.8	30.6	
Feel personal obligation to contribute because use F/OSS	28.6	29.6	26.9	
Dislike proprietary software and want to defeat them	11.3	11.5	11.1	
Enhance reputation in F/OSS community	11.0	12.0	9.5	

Table 5.7: Impact of Significant Variable on Project Hours/Week (Standardized Coefficients)

Variable	Coeff.	t-statistic (p-value)
Creative project experience	1.6	6.00 (0.000)
Paid status	0.88	3.12 (0.002)
Like team	0.84	2.76 (0.004)
Enhance community reputation	0.56	2.00 (0.046)
Differential hours	-1.6	-6.00 (0.000)
IT training	-0.6	-2.28 (0.023)

5.2: Distributed Information and Knowledge and the Value of the Periphery

The first two studies have demonstrated the important role of peripheral participants in two different distributed innovation settings. Findings from research on the sources of technological innovation, the sociology of science, and information flow in communities have indicated that peripheral participants provide novel information and innovations and are thus critical to the advancement of knowledge in dynamic settings. Underlying these findings is the notion that information and knowledge are widely distributed and finding ways to access this knowledge is a central challenge for society (Hayek, 1945).

Research on the sources of technological innovation has repeatedly highlighted that novel innovations arise when problem solving activity is decentralized (von Hippel 1988; von Hippel 2005). Users have been shown to innovate in a variety of consumer, industrial and scientific settings (von Hippel 2005), often preceding and initiating firm-based efforts (von Hippel 1978; von Hippel 1982; von Hippel 1988; von Hippel 1989). Here users are on the periphery and the firms that commercialize and sell products which have innovations are the core. Thus in the field of scientific instruments, Riggs and von Hippel (1994) found that 44% (n=64) of the innovations emerged from users dispersed in industry, universities and government laboratories, while the remaining 56% emerged from a handful of manufacturers. They further found that the vast majority of functionally novel innovations, that is, enabling new technical capability in the equipment, were developed by dispersed users and "dimension of merit" improvements, i.e. convenience or reliability, were developed by manufacturers. More recently, DeMonaco, Ali & von Hippel (2005) have shown that in pharmaceuticals industry, 76% (n=29) of the new drugs introduced in 1998 had significant "off-label", that is, novel uses not in the original drug approval process, applications. They found that 59% (85/144) of the "off-label" drug therapy innovations were discovered by distributed and peripheral practicing clinicians via field discovery as compared to the scientists working inside of the pharmaceutical companies.

One stream of research in the sociology of science has argued that the flow of ideas and innovation in scientific communities is centripetal instead of centrifugal (Chubin 1976), that is, the margins of the scientific community are the drivers of change and progress. Thus Crane (1969: 349) in her study of the "invisible college" in the natural sciences speculated that "outsiders" were a likely source of new ideas and innovation: "Most problem areas are open to influence from other fields. The desire for originality motivates scientists to maintain contacts with scientists and scientific work in areas different from their own in order to enhance their ability to develop new ideas in their own areas."

A review of six scientific disciplines, (radio astronomy, bacteriology, psychology, phage group, physical chemistry, x-ray protein crystallography) by Edge and Mulkay (1974) (cited by Chubin 1976) showed that innovations from the margins and the mobility of scientists across fields were the only consistent factors in scientific innovation and specialty development across these fields. Edge and Mulkay did express concern that very little was known about the social process underpinning their findings: "If we are correct in suspecting that many major scientific innovations come from the outside, or from the margins of, established research communities (either from applied research contexts, or by migration between research networks), then it is surprising that so little is known about this process" (Edge and Mulkay (1974) cited in Chubin (1976: 457)).

Within sociology, Weiman's (1982) study of the flow of information and influence in the personal network of an Israeli kibbutz community also shows "the importance of marginality" or peripheral participation. Weiman gathered sociometric data from 270 members of the kibbutz regarding conversational ties with other members of the community yielding 2511 conversation ties. Weiman then used matrix algebra to determine cliques in the community and then derived a network position of each individual based on the number of times a person was chosen as a conversational tie by someone else. "Centrals" and "Marginals" were then determined by using the upper and lower quartiles of the choice distribution in a clique. As expected centrals, dominated in

all types of communication patterns. In addition centrals, were more efficient in the flow of information. Information originating from centrals flowed more faster, was deemed more accurate and more credible than the information activated by the marginals. However, marginals were key for inter-group or inter-clique communication. Marginals were both receivers and transmitters of information amongst the 16 distinct groups in the kibbutz. Weiman showed that marginals were the importers of new information across groups and that centrals then served as the transmitters of that information within groups. Implying that centrals rely on marginals for imported information while the marginals required the enlistment of centrals for spreading the information in the group.

5.3: The Advantage of the Periphery

There are two theoretical perspectives underpinning the findings related to the importance of the periphery. Granovetter's (1973) seminal article on the strength of weak ties posits that weak ties amongst individuals allow for the transfer of non-redundant and novel information amongst colleagues as opposed to strong ties amongst friends. Strong ties imply that the information flow amongst strongly connected individuals will be homogenous and already known, while weak ties may enable the transfer of new and heterogeneous information. Thus those on the periphery of a community are more likely to be weakly tied to the core, while they may serve as "bridges" between other communities and thus transfer novel information amongst them. This theoretical perspective is the basis for both Weiman's empirical findings about the importance of marginality and Chubin's assertion regarding the centripetal flow of novel information in science communities. Although not mentioned by Granovetter, Hayek's (1945) central insight about the unequal and distributed nature of knowledge in society explains why non-redundant information may exist in the first place. If knowledge is both spatially and intellectually distributed – then gaining access to this knowledge via weak ties may be one mechanism by which peripheral members provide advantages to communities.

The other theoretical perspective on the value of the periphery arises from von Hippel's findings about the critical role of users in the innovation process. Here the

theoretical perspective is the relative stickiness of information. Von Hippel argues that the locus of innovation shifts to where the information is the most stickiest (von Hippel 1994a; von Hippel 1994b; von Hippel 1999). Thus users innovate in areas where they have needs not met by manufacturers, typically in using technologies in novel ways, while manufacturers innovate in areas where they have pre-existing expertise, typically manufacturing the technology or improving it on the dimensions of merit instead of novelty. It is not just a matter of the presence of non-redundant information. Rather users or peripheral members in problem solving communities experience novel issues not foreseen by manufacturers or core members and in many cases the transfer of this use experience is very difficult and expensive, if not impossible. A strong tie between a core developer and a peripheral user does not mean the core will now have the information needed to innovate. Rather the use environment will dictate that such innovation has to be primarily driven by the periphery. Thus the periphery has to first innovate and then transfer the newly created knowledge to the core, regardless of the strength of ties. Underlying this perspective is the that when knowledge is totally sticky and distributed, functionally novel innovations – involving novel need information - will occur primarily in the periphery (Riggs et al., 1994; Tyre et al., 1997; von Hippel, 2005). When knowledge is totally non-sticky, problem-solving might occur only in the core. The stickiness of knowledge then determines the locus of innovation and peripheral users who experience novel use conditions provide functionally novel innovations.

Another advantage that the periphery has is that novel problems in one field may have related solutions or solution information in another field. An example from modern finance theory illustrates the applicability of solutions developed in a distant field, i.e. physics. The development of modern finance theory is directly linked to the insights of M. F. M. Osborne, a physicist in the U.S. Navy, who realized in 1959 that financial market prices followed the equations of Brownian motion that Albert Einstein and Norbert Wiener had developed many years earlier (Chance & Peterson, 1999; Osborne, 1959). Furthermore, the original solution to the Black-Sholes option pricing model relied on a extremely complex calculation of parabolic partial-differential equation based on the premise that stock prices exhibit Brownian motion. However, Black later realized that

the complex equation could be easily transformed into the heat-diffusion equation of thermodynamics, for which the solution was already well known and understood (Chance et al., 1999). Peripheral members may be able to then bring unique solutions to problems in distant fields.

Findings from the first study show that a central feature of a periphery-centered problem solving system is the relatively high degree of reuse and recombination of previously developed solutions in the creation of new submissions. A majority of the winning solvers (55%) indicated that they had made major modifications to pre-existing solutions in their submissions with 17.5% reporting a direct port of previously developed solutions with little or no modifications. A periphery-centered problem solving system exposes potential solvers to problems that they do not routinely encounter and it triggers creative analogizing between their own expertise and knowledge base and the "new" distant problem. This burst of creativity causes the solvers to either significantly modify and recombine pre-existing solution knowledge that they already had or come up with entirely new ideas (as done by 27.5% of the solvers) for effective problem resolution. The net benefit of the juxtaposition of a solver and a distant problem is the creation of a solution that is both novel to the seeker and also to the solver. It also implies that problem broadcasting to heterogeneous periphery participants can trigger cross-fertilization of knowledge fields.

Periphery members may also incur low social costs for innovative activity. Edge and Mulkay have noted that in a scientific discipline the centripetal flow of innovations occurs either through low status individuals in the discipline or through migration of high status individuals from outside the disciplines. Low status individuals have low visibility in the dense communications networks of the discipline's core participants and are thus free to innovate with radical ideas and have relatively little to loose if their work does not deliver. Whereas high status individuals inside of a field may not be able to experiment as much with new ideas because the cost of failure and embarrassment is much higher. High status outsiders can make a contribution to another field by bringing tools and techniques to bear on the target field that have been developed elsewhere. Typically this

migration is from a field that has high paradigm coherence to one that has low paradigm coherence. Schrödinger's Dublin lectures on "What is Life" (Schrodinger, 1951) can be seen as a migration from physics to biology of a high status individual bringing with him the tools of the physical sciences to biology.

Peripheral users often need to solve their own problems and thus can gain high benefit by innovating for themselves. It may not be feasible for them to transfer the problem to the core membership due to time constraints, i.e. they need a solution right away and may not want to wait for someone else to create a solution or it may be too difficult to explain or recreate the problem for others. In either case, once the problem has been solved, the periphery member has the option to give the solution to the core for possible inclusion into the base system. Work by Harhoff, Henkel and von Hippel (2003) indicates that there is strong economic rationale for periphery users to freely reveal their innovations to others in a distributed setting.

5.4: Heterogeneity in Motivations to Participate in Distributed Innovation Systems

The first and third study have highlighted that both peripheral and core members in distributed innovation settings indicated heterogeneous motivations to participate. In both settings, peripheral or core members participate voluntarily and exert effort without an explicitly guaranteed reward. Thus incentives to contribute need to appeal to both intrinsic and extrinsic motivations of the participants. Across both studies, intrinsic motivational factors had a statistically more significant and stronger effect on effort and outcomes over financial other extrinsic motivations.

The first study showed that broadcast search in scientific problem solving attracts solvers who have varied motivations to participate. The probability of being a winning solver was significantly correlated with both a desire to win the award money as well as intrinsic motivations like enjoying problem solving and being intellectually challenged. Surprisingly, even though there was a substantial prize award for creating the best solution, the effect of intrinsic motivation was stronger and more significant. The stronger effect of intrinsic motivation is consistent with theory and empirical findings

which indicate that scientists have a "taste" for science (Stephan & Levin, 1992; Stern, 2004), that is, they will sacrifice financial gain for the chance to do the type of work they prefer.. There was significant negative correlation between intrinsic motivation and financial motivation indicating that broadcast search attracted solvers who were either financially driven or intrinsically motivated with no crowding out between the motivations (Deci, Koestner, & Ryan, 1999).

The third study on motivations of core members to participate in OSS communities also demonstrated heterogeneity in motivations to contribute. As table 5.6 shows there was no one majority preference for motivations (user need was split between work and non-work). Exploratory cluster analysis of the motivation response patterns revealed the existence of four stable cluster grouping amongst the respondents corresponding to motivations classified by hybrid intrinsic/extrinsic motivations for intellectual stimulation and skill improvement, extrinsic user need motivations (separate clusters for work and non-work), and obligation/community based intrinsic motivations. Core members may participate for a variety of reasons and no one reason tends to dominate the community or cause people to make distinct choices in beliefs or activities. These findings are consistent with collective action research where group heterogeneity is considered an important trait of successful social movements (Marwell & Oliver, 1993).

However it is important to note that that most significant and strongest correlate of effort (hours/week spent) was a contributor's own sense of creativity in their work on the project. A majority of the respondents had noted that their work in the OSS community was the most creative or equivalent to most creative activity that they had done in their lives. The effect of this response was significant across and within the various OSS communities studied. Indicating that the sense of creativity is not related to the type of project but a function of the work that was being accomplished by the participant.

5.5: Generalizable Conditions for Distributed and Self-Organizing Innovation Systems

In this dissertation I have sought to develop insight into the problem solving mechanisms and motivations of participants in two distinct, distributed and self-organizing innovation systems. The first system utilized problem broadcasting to a distributed and "unknown" periphery of scientists in the hope of solving previously unsolved and commercially valuable scientific problems. Potential solvers self-selected themselves to undertake the problem solving activity and to submit innovative solutions for evaluation and possible financial reward. The second system, as typified by open source software communities, is a completely decentralized distributed innovation setting where the product development process integrates core and peripheral members in collective problem solving episodes. Here peripheral members create a majority of the functionally novel solutions and play an active and collaborative part with other members (core and periphery) who are also attempting to create software. In both systems, participants indicated heterogeneous motivations to participate with intrinsic motivations being significant correlates of both successful output and effort.

In this final section of the dissertation I will discuss the generalizable conditions under which distributed and self-organizing innovation systems operate and their potential robustness and extensions to other domains. My discussion will center around the problem solving approach utilized in these settings and will specifically consider the circumstance of the entry of participants, the problem definition phase, the solution generation phase, and how trust and management are established.

5.5.1 Entry and Participation

A key component of distributed innovation systems is entry and participation by diverse individuals who may not necessarily be formally affiliated with the focal innovating entity. Three elements appear to be common in both the setting and need to be considered: 1) self-selection onto tasks and problems; 2) lowering social and technical barriers to participation; and 3) allowing for heterogeneity in motivations by participants.

Participation and activity in distributed innovation systems is not typically driven by managers who are commanding employees to follow orders and execute on pre-prepared tasks or optimally matching employees to tasks. Instead, members (both core and periphery) participate by voluntarily self-selecting themselves onto the problems at hand. Members determine for themselves if they have the ability, inclination, need or motivation to participate in the problem solving process. Self-selection then distributes the resource allocation problem to individual solvers instead of centralized managers. Since problem solving resources are not owned by the focal innovating entity, distributed members make their own decisions regarding participation in the process and bear the risk of failure (not solving the problem or not having a submission accepted) or reap the benefits of success.

Distributed innovation systems demonstrate relatively lower social and technical costs for participation and entry. These systems require a continual influx of new members to bring forth new ideas and concepts and consequently have entry requirements that are designed to encourage participation. On the social side, in the F/OSS community setting, members do not have to first ask permission to work on any particular part of the software code on their own. Often, periphery members attempt code changes privately and then reveal their results to the development e-mail list only if they were successful. Failure does not have to be publicly revealed thus lowering the social costs of community participation. If members had to make an ex-ante revelation of intent or ask for permission before hand – then a lot fewer attempts would be expected as potential contributors would have their public reputations at stake and would need to be confident in their ability to deliver on their claims. Similarly in the broadcast search setting, individuals can become members of the solver network by simply signing up for a user name and password on InnoCentive.com's website. There is no ex-ante pre-qualification and credentialing process. In addition, solvers remain anonymous to both seekers and to other solvers, thus if an attempt at problem solving is unsuccessful, there is no concern that others will know about their failed attempts. This lowering of social costs may encourage risk taking and innovative solution development as members will not be constrained by the social setting.

On the technical side, the costs of participation in a distributed setting are also relatively lower as well. In F/OSS communities, micro-contributions (Sproull, Conley, & Moon, 2005: 144), in the forms of reporting a bug, sharing an idea, requesting a feature, participating in a discussion, doing translations, testing software changes, contributing code are the essential currency of generalized exchange (Ekeh, 1974) and participation. Membership and entry in the community can take many forms and does not require a significant up-front investment for initial participation. As Sproull et al note (2005: 144): "Not only is it easy to make a helpful contribution, it is also easy to control the extent of further involvement. In the offline world, a person may hesitate to offer help for fear that helpful response will lead to further demands on one's time or emotional energy. In the online world, a person offering help may feel in complete control of how much further involvement will ensue; he or she can simply ignore further requests."

Another reason for relatively low technical costs is the participants are typically using information at hand for a majority of problem solving effort. Thus most participants do not invest in any significant de-novo problem solving effort instead their knowledge at hand, and in some cases, combined with the knowledge of others helps to solve problems. Lakhani and von Hippel (2003) have shown that F/OSS community participants are typically providing information that they have readily at hand for solving technical support problems. The broadcast search study also demonstrated that the majority of winning and non-winning solvers were using solution information that they had developed themselves or obtained from others. In either case the relatively low technical costs for participation means that there is a significant asymmetry in costs and benefits for participation. Problem solvers in the distributed setting incur relatively low or trivial costs for participation – whereas – those with solutions obtain significantly high benefit. Raynor and Panetta (2005) have shown that firms can obtain an ROI of up to 2175 % for winning solutions in the broadcast search setting and do not have to pay for the effort of unsuccessful solutions.

The ex-ante risk of an unsuccessful submission and wasted effort is borne by the individual participants in a distributed innovation setting. Thus the incentives to participate need to appeal to a wide set of motivations amongst individuals with no one type of motivation dominating in the explicit or implicit incentive scheme. Just as individuals self-select on to tasks – they will also select the task based on their own internal assessment of the costs and benefits for participation. The primary direct cost of participation is the time required to create a submission, however, the benefits of participation need to accrue to individuals regardless of a successful or unsuccessful submission. The statistically significant results for intrinsic motivations in both distributed innovation settings indicate that individual participation is driven by factors that appeal to a personal sense of creativity, enjoying the problem solving process and the associated learning. This does not mean that other motivations, example financial or reputational are not important. Indeed in the commercial setting of broadcast search, being motivated by the award money was also a significant correlate of creating a winning solution. Rather, distributed innovation systems in their requirement for periphery participation need to be able to absorb heterogeneity in participant motivations and ensure that situationaly appropriate extrinsic and intrinsic rewards are available.

5.5.2 Decomposable and "Well-Structured" Problems

Simon's work on problem decomposability (Simon, 1969) and the structure of ill-structured problems (Simon, 1973) provide a lens into thinking about the types of problems that may be suitable for solving in a distributed innovations setting. Simon's first paper (1969) created a classification of complex systems as being either *decomposable, non-decomposable and nearly decomposable*. With decomposability being a function of the level of interactions amongst subsystems within a system. In Simon's second paper he made the strong contention that: *"In general, the problems presented to problem solvers by the world are best regarded as [ill-structured problems] (ISPs). They become [well-structured problems] WSPs only in the process of being prepared for the problem solvers. It is not exaggerating much to say that there are no WSPs, only ISPs that have been formalized for problem solvers."* Thus WSPs are the

result of a problem-defining process for ISPs (Foss & Foss, 2004) and that there is a continuum of problem types between ISPs and WSPs.

A central feature of both the distributed innovation settings in my dissertation research was the presence of relatively well structured and discrete "problems." Thus finding a "cure for AIDS" would not be a suitable broadcast search problem but finding "a method for isolating a specific gene" might be reasonable. Similarly, in the F/OSS example, trying to implement a particular software feature or solving a particular software bug drove the community activity instead of generalized requests for abstract new features. Underlying these settings is a process of decomposing and structuring problems that enable multiple heterogeneous solvers to attempt to create solutions.

In the F/OSS setting, problem decomposition and structuring occurred in a completely distributed fashion. Typically peripheral members would initiate the collective problem solving episodes by reporting system bugs or desire for new functionality to all members in the community. Subsequently, interested community members would first attempt to characterize the exact nature of the problem and then work towards creating a solution. Substantial effort was put forth in defining the exact source of the problem report and/or the proper requirements of a new feature need. In other cases, when the software code was already pre-developed, periphery members would first articulate the problem they had attempted to solve and provide evidence of their solution. In either case effort was put forth in creating a bounded and solvable problem.

In the broadcast search study, problem decomposition and structuring was done exclusively by the scientists inside of seeker firms in cooperation with InnoCentive.com. Hence the core in this distributed innovation setting does problem definition by itself. I found an overall marginally positive effect for firm learning (as measured by the number of problems previously posted) on the probability of a problem being solved. Innocentive.Cpm scientific operations staff in discussing this finding have told me that scientists inside seeker firms value the upfront and separated problem and solution

definition stage. Generally speaking, the critical first step for success in a broadcast search environment is to provide structure to ill-structured problems. I speculate that such an explicit problem structuring process is not routine inside of firms and thus experience with this approach may allow for learning over time on how to accomplish it. Furthermore, this learning around problem structuring and decomposition may be the driver that allows specialists from different fields to make the connection between their own domain knowledge and the problem.

5.5.3 Solution Generation

Two important streams in the individual/cognitive basis of problem solving are the Gestalt-based tradition on important role of insight in problem solving and the Simonian view of problem solving as a search in a problem space. Both of these traditions provide a lens into the solution generation process within a distributed innovation setting.

The Gestalt tradition, as best represented by the work of Duncker, argued that solutions occurred when the individual gained "insight" into the problem. Sudden insight enabled problem solvers to discover a crucial part of the problem and once that was discovered all other parts automatically fell into place and the problem was solved. Duncker (1945) and colleagues (Adamson, 1952; Birch & Rabinowitz, 1951) have shown the existence of "functional fixedness" where problem solvers have difficulty in using familiar tools in novel ways as a major source of insight blockage. In one of Duncker's experiments, he created five problems which could only be solved by applying a new way of using a tool. The first of the two groups of experimental subject saw the tool being used in a usual way while the second group did not. The result of the exercise was that subjects were more likely to solve the problems requiring a novel way of using the tool if they had not observed how that tool was used in the usual way while the problem solving success of subjects that had previously observed it used was hampered.

In Duncker's terms the subjects were "fixated" on the tool's normal function and could not re-conceptualize it in a way that permitted them to solve the problem. The way

around functional fixedness and the generation of new insight is for the individual solver to work on restructuring the problem so that it is amenable to insight generation. Restructuring of problem could be done via hints to the problem solver or by the problem solver rethinking through the constraints of the problem description and solving it non-obvious ways.

Distributed innovation and problem solving settings have the potential to overcome functional-fixedness from prior experience by distributing the problem to many potential solvers. Core and periphery members in a distributed innovation setting would have different prior experiences and would thus attempt to create solutions that were different from the one created by the problem holder. Accessing the periphery may also enable multiple attempts at insight generation or the leveraging of existing insights into new problems. Thus increasing the probability of a successful outcome.

The work of Newell and Simon and colleagues (Newell & Simon, 1972; Simon & Newell, 1962) has described the problem solving process as an attempt of getting from the present to desired situation by a process of "searching through a large maze." The maze depicts the problem space; the nodes of the problem space represent situations; and the paths joining one node to another are the actions that will transform one situation into another. A problem space has an initial state and a goal state and a set of means that allows a solver to move from one state to another. Specifically, Newell and Simon modeled the problem solving process as following two steps: representation of the problem, a perspective, and the application of a search heuristic (Hong & Page, 2001). Thus solution generation requires the use of a perspective-heuristic pair, where the potential solution space is defined by the internal problem representation, the perspective, and a search algorithm to find the solution through the space, a heuristic (Hong & Page, 2004).

In a distributed innovation setting, the independence of problem holders and problem solvers allows for the existence of multiple perspective-heuristic pairs and in collective settings the inter-mingling of pairs across individuals. Thus parallel searches

through a variety of solution landscapes and the utilization of diverse heuristics increase the possibility of the creation of a successful solution. The strongest evidence for this conjecture was obtained in the broadcast search study where the relative scientific interest heterogeneity of the solver base was a significant predictor of problem resolution. In addition, one can interpret the counter-intuitive finding of relative outsiders solving problems that are at the boundary or outside of their field of expertise as the enactment of a sufficiently novel perspective-heuristic pair by a periphery member that was not within in the domain of the core problem holders.

5.5.4 Substitutes for "Trust" and "Management"

The development of trust has been found to be important for success in distributed and virtual teams (Jarvenpaa & Leidner, 1999; Powell, Piccoli, & Ives, 2004; Sarker, Lau, & Sahay, 2001). Nearly all definitions of trust imply that one party, the truster, must willingly place themselves in a position of vulnerability to or risk from another party, the trustee (Gallivan, 2001: 280). Traditional theories of trust development note that trust is built through shared social norms, repeated (face-to-face) interactions, shared experiences, anticipation of future interactions, interpersonal relationship development, and emotional support (Jarvenpaa et al., 1999).

In distributed innovation settings, where many individuals with no prior relationships participate, traditional methods of trust development may not be entirely possible. Instead "substitutes" for trust need to be found so that collective action and problem solving can still take place. One substitute for trust is to require objective measures of performance and means of assessing task accomplishment. In the broadcast search setting, this is easily accomplished by the rules of the game which stipulate that only successful submissions, i.e. those that meet the solution criteria as determined by the seeker firm, are given an award. Seeker firms do not have to monitor and follow up on the activities of the over 200 solvers that may be attempting to develop solutions on any one problem. Objective evidence of successful task accomplishment is the only means by which awards are given out.

For F/OSS communities, the continual public submission of software code provides a substitute for traditional forms of trust. Software development provides for the verification of completed work, i.e. "Does it do what the contributor claims?" via "objective" measures. "*Truth in code*" is achieved when others apply submitted changes to their personal copies of the software repository and then verify if the software works. Verification works at both the basic level, i.e. does the changed system actually work as advertised, and at an advanced level, i.e. is there any degradation in performance (speed, load handling etc) due to the changes. Software development kits provide standard tools that produce objective measures, about software changes, which can be shared with others and serve as a basis for further development. Those submitting problems are also able to utilize "objective measures" to make their claims. Contributors raising issues provide (or are asked to provide) detailed information on how specific problems were created along with associated computer-generated log files and use scenarios which allow others to recreate the exact issues on their own local machines.

Another dimension of distributed innovation systems is that is that coordination happens informally and there is very little explicit management of work and activities of the participants. In the broadcast search example there is no possibility of managing the activities of potential solvers who are attempting to create submissions. In the case of F/OSS, communities do not attempt to (re)create bureaucratic modes of organizing as typified in software engineering projects inside of firms and other formal organizations (Adler, 2003). Synchronicity of action in the community is not primarily determined by plans, schedules and technological road maps or individuals undertaking these activities for ***other*** community members. Instead, individual members choose the task that they will undertake, how it will be initially accomplished and when it will get done. The essential work and task structures are limited to proposing ideas, writing, testing, and using code, and commenting in discussions. Coordination occurs via mutual adjustment and primarily by "organizing work by adaptation" (Hutchins, 1991). Adaptation and adjustment is facilitated by the collective practices and individual practices discussed in section 5.2.1 and in particular by work broadcasting. These practices can be viewed as substitutes for traditional management where planning and doing are considered separate

functions. Similar to findings on communities of practice, where learning is not separated from working (Brown & Duguid, 1991), coordination and management occurs while work is completed and revealed and is not a separate form of activity or specialization within the communities.

****

In this dissertation I have attempted to build an initial understanding about how distributed and self-organizing innovation systems operate. Beyond open source and the broadcast search model, there are now a multitude of natural experiments with distributed innovation systems. Wikipedia has established itself as a viable alternative to closed models for knowledge assembly. The X-Prize has shown that complex aeronautical and space engineering can be accomplished by distributed teams. In a more mundane setting, a company called threadless.com relies on a vast periphery of designers to submit and rate t-shirt designs. As I close this dissertation I would like to reflect on what I consider to be one of the essential attributes of these distributed innovation system; openness to outsiders. Distributed information systems require an organizational architecture that will allow access to peripheral individuals not necessarily affiliated with the focal innovating unit. Ex-ante openness in the problem solving process is an important feature of these systems and will require traditional organizations to re-think how boundaries are conceived. New organizations may as a start be designed so that boundaries for innovation are permeable in both directions. That is, internal members should be free to go outside and external participants are given opportunities to directly innovate on internal issues. members One way or the other distributed innovation systems require openness so that outsiders can participate. Free and open source software communities have shown how this can be done and now the challenge is for more traditional organizations to adapt to this new way of innovating.

References

Adamson, R. E. 1952. "Functional fixedness as related to problem solving: a repetition of three experiments." *Journal of Experimental Psychology* 44:288-291.

Adler, Paul S. 2003. "Practice and process: The socialization of software development." in *Academy of Management Best Papers*. Seattle, WA.

Amabile, Teresa M. 1996. *Creativity in context*. Boulder, CO: Westview Press.

Birch, H. G., and H. S. Rabinowitz. 1951. "The negative effect of previous experience on productive thinking." *Journal of Experimental Psychology* 41:121-126.

Brown, John Seely, and Paul Duguid. 1991. "Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation." *Organization Science* 2:40-57.

—. 2001. "Knowledge and Organization: A Social-Practice Perspective." *Organization Science* 12:198-213.

Chance, Don M. , and Pamela P. Peterson. 1999. "The new science of finance." *American Scientist* 87:256-264.

Chubin, Daryl E. 1976. "The Conceptualization of Scientific Specialties." *The Sociological Quarterly* 17:448-476.

Crane, Diana. 1969. "Social Structure in a Group of Scientists: A Test of the "Invisible College" Hypothesis." *American Sociological Review* 34:335-352.

Csikszentmihalyi, Mihaly. 1975. *Beyond Boredom and Anxiety: The Experience of Play in Work and Games*. San Francisco: Jossey-Bass, Inc.

Deci, Edward L, R Koestner, and Richard M Ryan. 1999. "A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation." *Psychological Bulletin* 125:627-688.

Deci, Edward L, and Richard M Ryan. 1985. *Intrinsic motivation and self-determination in human behavior*. New York, NY: Plenum Press.

DeMonaco, Harold J., Ayfer Ali, and Eric von Hippel. 2005. "The Major Role of Clinicians in the Discovery of Off-Label Drug Therapies." *MIT Sloan School of Management Working Paper Series*.

Duncker, K. 1945. "On problem solving." *Psychology Monographs* 58.

Edge, David O, and Michael J Mulkay. 1974. "Case studies of scientific specialties." University of Edinburge, Science Studies Unit.

Ekeh, Peter P. 1974. *Social Exchange Theory: The Two Traditions*. Cambridge, MA: Harvard University Press.

Foss, Kirsten, and Nicolai J Foss. 2004. "Simon on problem solving: Implications for new organizations forms." *Copenhagen Business School Working Paper*.

Frey, Bruno. 1997. *Not just for the money: an economic theory of personal motivation*. Brookfield. VT: Edward Elgar Publishing Company.

Gallivan, Michael J. 2001. "Striking a balance between trust and control in virtual organization: a content analysis of open source software case studies." *Information Systems Journal* 11:277-304.

Granovetter, M. 1973. "The strength of weak ties." *American Journal of Sociology* 78:1360-1380.

Harhoff, Dietmar, Joachim Henkel, and Eric Von Hippel. 2003. "Profiting from voluntary information spillovers: How users benefit by freely revealing their innovations." *Research Policy* 10:1753-1769.
Hayek, F. A. 1945. "The use of knowledge in society." *American Economic Review* 35:519-530.
Hong, Lu, and Scott E. Page. 2001. "Problem Solving by Heterogeneous Agents." *Journal of Economic Theory* 97:123-163.
—. 2004. "Groups of diverse problem solvers can outperform groups of high-ability problem solvers." *PNAS* 101:16385-16389.
Hutchins, Edwin. 1991. "Organizing work by adaptation." *Organization Science* 2:14-39.
Jarvenpaa, Sirkka L, and Dorothy E Leidner. 1999. "Communicaion and trust in global virtual teams." *Organization Science* 10:791-815.
Koch, S, and G Schneider. 2002. "Effort, Cooperation and Coordination in an Open Source Software Project: GNOME." *Information Systems Journal* 12:27-42.
Lakhani, Karim R., and Eric von Hippel. 2003. "How Open Source Software Works: Free User to User Assistance." *Research Policy* 32:923-943.
Lee, Gwendolyn, and Robert E Cole. 2003. "From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development." *Organization Science* 14:633-649.
Lerner, Josh, and Jean Tirole. 2002. "Some Simple Economics of Open Source." *Journal of Industrial Economics* 50:197-234.
Lindenberg, Siegwart. 2001. "Intrinsic motivation in a new light." *Kyklos* 54:317-342.
March, James G, and Herbert Simon. 1958. *Organizations*: Wiley.
Marwell, Gerald, and Pamela Oliver. 1993. *The Critical Mass in Collective Action: A Micro-Social Theory*. Cambridge, UK: Cambridge University Press.
Nakamura, Jeanne, and Mihaly Csikszentmihalyi. 2003. "The construction of meaning through vital engagement." in *Flourishing: positive psychology and the life well-lived*, edited by Corey L Keyes and Jonathan Haidt. Washington, DC: American Psychological Association.
Nelson, Richard R., and Sidney G. Winter. 1982. *An evolutionary theory of economic change*. Cambridge, MA: Belknap Harvard.
Newell, Allen, and H.A. Simon. 1972. *Human Problem Solving*. Engelwood Cliffs, New Jersey: Prentice-Hall INC. .
Orlikowski, Wanda J. 2002. "Knowing in practice: Enacting a collective capability in distributed organizing." *Organization Science* forthcoming.
Osborne, M.F.M. 1959. "Brownian motion in the stock market." *Operations Research* 7:145-173.
Podolny, Joel M., Toby E. Stuart, and Michael T. Hannan. 1996. "Networks, Knowledge, and Niches: Competition in the Worldwide Semiconductor Industry, 1984-1991." *American Journal of Sociology* 102:659-689.
Powell, Anne, Gabriele Piccoli, and Blake Ives. 2004. "Virtual Teams: A Review of Current Literature and Directions for Future Research." *Database for Advanced in Information Systems* 35:6-36.
Raynor, Michael E, and Jill A. Panetta. 2005. "A better way to R&D?" *Strategy & Innovation: A newsletter from Harvard Business School Publishing and Innosight* 3:14-16.

Riggs, William, and Eric von Hippel. 1994. "Incentives to innovate and the sources of innovation: The case of scientific instruments." *Research Policy* 23:459-469.
Sarker, S., F. Lau, and S. Sahay. 2001. "Using an Adapted Grounded Theory Approach for Inductive Theory Building about Virtual Team Development." *Database for Advanced in Information Systems* 32:38-56.
Schrodinger, Erwin. 1951. *What is life? The physical aspect of the living cell.* Cambridge, UK: Cambridge University Press.
Simon, H.A. 1969. *The Sciences of the Artificial.* Cambridge: Massachusetts Institute of Technology.
Simon, H.A., and Allen Newell. 1962. "Computer Simulation of Human Thinking and Problem Solving." *Monographs of the Society for Research in Child Behavior* 27:137-150.
Simon, Herbert A. 1973. "The structure of ill structured problems." *Artifical Intelligence* 4:181-201.
Sproull, Lee, Caryn Conley, and Jae Yun Moon. 2005. "Prosocial behavior on the net." Pp. 139-162 in *The Social Net: Human Behavior in Cyberspace*, edited by Yair Amichai-Hamburger. Oxford, UK: Oxford University Press.
Stephan, Paula E., and Sharon G. Levin. 1992. *Striking the mother lode in science : the importance of age, place, and time*. New York: Oxford University Press.
Stern, Scott. 2004. "Do scientists pay to be scientists?" *Management Science* 50:835-854.
Tyre, Marcie J, and Eric von Hippel. 1997. "The situated nature of adaptive learning in organizations." *Organization Science* 8:71-83.
von Hippel, Eric. 1978. "Successful industrial products from customer ideas." *Journal of Marketing* 42:39-49.
—. 1982. "Get New Products from Customers." *Harvard Business Review* 60:117-122.
—. 1988. *The Sources of Innovation.* New York, NY: Oxford University Press.
—. 1989. "New Product Ideas from "Lead Users"." *Research Technology Management* 32:24-27.
—. 1994a. "'Sticky information' and the locus of problem solving: Implications for innovation." *Management Science* 40:429-439.
—. 1994b. "Sticky Information and the Locus of Problem Solving." *Management Science* 40:429-439.
—. 1999. "Economics of product development by users: Impact of "sticky" local information." *Management Science* 44:629-644.
—. 2005. *Democratizing Innovation.* Cambridge, MA: MIT Press.
von Krogh, Georg, Sebastian Spaeth, and Karim R Lakhani. 2003. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study." *Research Policy* 32:1217-1241.
Weimann, Gabriel. 1982. "On the Importance of Marginality: One More Step into the Two-Step Flow of Communication." *American Sociological Review* 47:764-773.