Vertical Interaction in Open Software Engineering Communities

Patrick Adam Wagstrom

CMU-ISR-09-103

March 2009

Carnegie Insitute of Technology and School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
James D. Herbsleb, Co-Chair
Kathleen M. Carley, Co-Chair
M. Granger Morgan
Audris Mockus

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Engineering and Public Policy and Computation, Organizations and Society.

This research was supported by the National Science Foundation through the Graduate Research Fellowship Program, National Science Foundation Grant No. IIS-0414698, the National Science Foundation IGERT Training Program in CASOS(NSF,DGE-9972762), the Office of Naval Research under Dynamic Network Analysis program (N00014-02-1-0973, the Air Force Office of Sponsored Research (MURI: Cultural Modeling of the Adversary, 600322), the Army Research Lab (CTA: 20002504), and the Army Research Institute (W91WAW07C0063) for research in the area of dynamic network analysis. Additional support was provided by CASOS - the center for Computational Analysis of Social and Organizational Systems at Carnegie Mellon University.

The views and conclusions contained in this thesis are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the the National Science Foundation, the Office of Naval Research, the Air Force Office of Sponsored Research, the Army Research Lab, or the Army Research Institute.

Keywords: Open Source, software engineering, software development, socio-technical systems, socio-technical congruence, computer supported cooperative work

To Kristina and my parents.

Abstract

Software engineering is still a relatively young field, struggling to develop consistent standards and methods across the domain. For a given project, developers can choose from dozens of models, tools, platforms, and languages for specification, design, implementation, and testing. The globalization of software engineering and the rise of Open Source further complicate the issues as firms now must collaborate and coordinate with other firms and individuals possessing a myriad of goals, norms, values, expertise, and preferences. This thesis uses four empirical studies to take a vertical examination of Open Source ecosystems and identify the way that foundations, firms, and individuals come together to create large scale software ecosystems and produce world class software despite their differing goals and values.

First, I examine Open Source as a collaborative phenomenon between firms and non-profit foundations that support many communities and identify the ways in which non-profit foundations enable member firms to create value in the ecosystem. Next, an empirical study of direct collaboration between firms within the Eclipse system reveals that most firms operate relatively independently, but there is still heavy reliance on a single dominant player for core portions of the ecosystem. I then evaluate how the presence of commercial firms affects the attraction and retention of volunteer developers in an Open Source community. The final study examines how individual developers manage their dependencies in Open Source and extends the socio-technical congruence metric to address changing requirements and facilitate the metric as a tool for continual use. Finally, based on the findings of these studies, I close with a set of recommendations for stakeholders investing in Open Source.

Acknowledgments

A huge amount of thanks goes to my primary thesis advisors, Jim Herbsleb and Kathleen Carley. Thank you for giving me the freedom to design and implement a course of research that I wanted. Thank you for gently guiding me back when I would stray from the course. And thank you for your input, direction, and most importantly, patience with me.

I've had the privilege to work with and build my ideas through interaction with many other amazing researchers, engineers, and scientists at Carnegie Mellon. In particular, I am in debt to Marcelo Cataldo for his work laying the foundation for socio-technical congruence. Thanks to all the folks in the CASOS lab and COS seminar for your feedback and advice on my research. Thanks to Peter Landwehr for his help in focusing some of the ideas regarding Eclipse. Thanks to Anita Sarma for her feedback and willingness to take on additional responsibilities so I could finish. I'm also grateful for the support and help from Robert Kraut, particularly for his help in understanding volunteer motivation and the social aspects of volunteer participation in open communities.

I've also had the pleasure of working with numerous individuals outside of Carnegie Mellon who have greatly shaped my research. Sonali Shah was invaluable in organizing, planning, and analyzing the interviews with the Eclipse Foundation. Thanks to the software governance team at IBM research: Clay Williams, Kate Ehrlich, Mary Helander and Peppo Valetto for their help in exploring some of the preliminary ideas with expanding the problem space around socio-technical congruence. I'm eternally in debt to my previous academic mentors at Illinois Institute of Technology, Xian-He Sun and Gregor von Laszewski who were willing to help direct an undergrad in a class he wasn't supposed to take on how to do research and be successful as a researcher and scientist.

Finally, this wouldn't be possible without the support of my friends and family. Thanks to everyone who has been in my small groups for letting me vent. Thanks to my parents for giving me just enough support so I still felt independent. Finally, thanks to my wonderful wife Kristina, whom I love very much and who tolerates all of my mood swings, listens to my frustrations, and is a sane force in my life. Thanks for being willing to move out here and share in the pain when you decided to come to Carnegie Mellon and get a Ph.D. for yourself.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software is embedded in almost everything that uses electricity. Often it is found in places that were once far removed from the domains of computer scientists. Devices thought of as primarily mechanical, such as automobiles, now contain millions of lines of code developed by thousands of software engineers[7, 43]. Traditionally passive receiving devices, such as televisions and phones, now routinely run operating systems designed for desktop computers. Faced with increased customer expectations for rich environments and increasing complexity of software, many firms have sought out pre-developed components that can easily be linked together, an no where is the supply of components richer, or more accessible, than Open Source software[6, 77].

1.1 A Brief History of Open Source

The roots of Open Source software go back to the dawn of the computing era. With early computers, the value of the machine was largely placed on the hardware around the machine, and the software was often given away or shared very liberally. One of the most prominent examples of this was the distribution of Unix from AT&T Bell Labs[73]. When AT&T first made Unix available, it was provided on a single magnetic tape that included the entire source code to the operating system. Unix was notable because it was written in a new machine independent programming language, C. This innovation allowed computer operators to enhance the operating system and share the changes with their friends and colleagues[73, 92].

In the 1980's the market for proprietary Unix and Unix-like operating systems began to emerge from companies like Sun, DEC, and HP. At the same time, however, Richard Stallman was laying the foundation for the Free Software movement through the creation of the GNU C Compiler (now renamed to the GNU Compiler Collection and referred to as GCC) and EMACS, a powerful and extensible text editor[103, 104]. Rather than licensing the software under traditional licenses, which prohibited replication and redistribution, the licenses of software from the Free Software Foundation, such as the GNU General Public License[32], actively promote redistribution of the software – with the caveat that if you modify and redistribute the software, you must make your modifications available under the same license. For his innovation in creating and distributing software, Richard Stall-

man was recognized with a MacArthur foundation "genius" grant. A nascent community formed around the Free Software Foundation and the project advanced slowly, but individuals were still forced to run software from the Free Software Foundation on proprietary Unix platforms[130].

It wasn't until a young Finnish student, Linus Torvalds, made the code for Linux, his personal Unix-like operating system, Linux, freely available under the GNU General Public license in 1991 that the Free Software Movement really took off. The components were now in place for a complete operating system and development environment that had zero monetary cost and was freely redistributable. Over the next six years the community around Free Software grew dramatically, thanks in large part to the selection of least common tools for developers and the growing prevalence of internet access for university students and technically inclined home users. Several companies began to find ways to eek out a niche business in the emerging Open Source market. Primarily these businesses were repackaging that software to make it easier for consumers to use[134]. The overall market for Free Software was still very small.

In early 1998, however, everything changed. In a surprise announcement, and largely in response to increasing competition from Microsoft, Netscape announced that they would release the entire source code to their Netscape web browser under a license similar to the GNU General Public License. Netscape's reticence to utilize the license of the Free Software Foundation highlighted key issues related to corporate adoption of Open Source; the term "Free", and the requirement that modifications to the source code be redistributed

under the same license as the original software. A group of luminaries in the community were summoned together and the term "Open Source" was created as a more palatable term that brought together almost all software that had source code freely available[117]. This change in terminology not only made the concept more acceptable to commercial endeavors, but also helped solidify the community as large projects like the Apache web server and FreeBSD were available under licenses that in many ways were more liberal than Free Software, but lacked the requirement that modifications to the software be redistributed under the same license.

The open sourcing of Netscape's browser source code was largely a disaster – it was several years before Netscape managed to ship a browser based on the released source code, by which time Netscape had been purchased by AOL and ceded the "browser war" to Microsoft. This wasn't because of a general repulsion to Open Source, but more because as the first large scale commercial project released as Open Source, there was no predecessor to base decisions on. Large issues were not addressed that today are second nature, such as ensuring the code can easily compile for home users, establishing public forums, and not requiring access to proprietary tools [80].

Despite the initial failure of Netscape to capitalize on Open Source, the movement continued to grow. Some of the biggest IPOs of the dot-com boom of the late 1990's, Red Hat and VA Linux, had business models that built directly on Open Source. Established tech giants began to take advantage of Open Source projects to build their own product lines. In June 1998, IBM made the announcement that the web server component of their

WebSphere line of products would be the Apache web server. In their rationale for the adoption, IBM representatives explained that the Apache web server was a high quality project, and it made little sense to continue to develop their own proprietary solution when it was not the primary value driving component of WebSphere[12, 53]. This was an early example of what would soon become a broad industry trend – the utilization of Open Source technologies to serve as critical components in commercial products across many sectors of the economy[5, 41].

The market has continued to evolve, and Open Source plays an even more critical role. In addition to Open Source being wildly successful on "back-end" server processes, many desktop applications are based on Open Source technologies. The Firefox web browser, is the heir to the code first released by Netscape in 1998. Apple's Safari web browser, available on Mac, Windows, and mobile phones, Google's Chrome browser, and Adobe's AIR environment are based on the Open Source web browser Konqueror. Eclipse is one of the dominant integrated development environments for software engineers, openly challenging Microsoft for the most popular development environment[37] and OpenOffice.org produces a completely Open Source suite of office programs that provides most of the functionality of Microsoft's Office suite of programs.

Parallel to the evolution of project source code, the communities around Open Source continue to evolve and adapt. In the 1990's communities around Open Source projects were almost entirely volunteers[71]. Collaboration was done almost exclusively over project mailing lists[44, 132]. High status in the projects was earned through a meritocratic sys-

tem that rewarded the best contributors to the ecosystem. As commercial interest in Open Source communities increased in the early part of the 2000's and tools for creating rich experiences on the world wide web . The community also grew. Many projects now feature user-friendly web forums where users can easily post questions and receive answers. One of the best examples of this evolution in tools is the suite of tools provided by Canonical to support the Ubuntu distribution of Linux[11]. These tools include many of the traditional Open Source tools, but also a framework called Launchpad that unifies source code management, bug tracking, and software distribution. Strategic use of these tools and a strong focus on the community have allowed the company to quickly expand to $30 million in annual revenues with only about 200 employees[9, 121].

The community around the Eclipse Foundation is another excellent example of a community that has evolved and is pushing Open Source in new directions. While many portions of the project operate as a traditional Open Source project would, the community actively recruits new firms to join the foundation and has instituted a rigorous intellectual property review process. This process serves to ensure that all code contributed to Eclipse can be legally used and helps to provide a guarantee to commercial partners building on the Eclipse framework[109].

1.2 Academic Research on Open Source

There has now been substantial research on Open Source. It can be categorized into two broad groups: research utilizing Open Source as a convenience sample for research on software development and research on the processes and phenomenon of Open Source. For the purposes of this thesis, which seeks to better understand the processes that underlie Open Source software development, the latter category is of much greater relevance.

Upon initial observation, Open Source appears to fit a classical "Tragedy of the Commons" paradigm. Individuals need not contribute to extract value from the completed software, therefore, few, if any individuals will contribute[45]. Much of the early research on Open Source sought to address this issue by attempting to understand the motivations of individual developers and the process by which such software is created. In 1997 Eric Raymond, an Open Source developer and lead of the project "fetchmail" first began to formalize some of the differences that made Open Source successful in a document that later became "The Cathedral and the Bazaar". Raymond noted that much of the success of Open Source, which at the time was very minor relative to the success it enjoys today, was due to the organizing principles of Open Source software and the way that it allows many people to do small amounts of self-directed work. Although based only on his observations as an Open Source developer, as one of the earliest works analyzing the Open Source movement, it remains an important contribution to the field[93].

The work of Raymond laid the foundation for additional practitioner/researchers to

write about their experiences in Open Source projects. Senyard and Michlmayr expanded on the work of Raymond and described the attributes necessary for a successful Open Source project. They found that more than just a "bazaar" was necessary for project success. Rather, they posited that strategic initial architectural decisions are necessary to drive the diverse innovations that lead to a successful project. In particular, projects should be modular to maximize the degree to which individual developers can work on discrete components[101]. The importance of modularity in Open Source was also found by MacCormick in his study of the Netscape web browser and Linux Kernel[66]. Much of this research echoes the ideas originally put forward by Parnas in his original argument for modularity and information hiding in software engineering[87].

Mockus, Fielding and Herbsleb performed one of the first broad analyses of how Open Source communities function when they examined the processes behind the Apache Software Foundation and the Mozilla project. They found that although there is a robust process of cooperation in most Open Source projects, there was great inequity in the distribution of work. Within Apache 85% of the work was performed by only 15 people, while there were hundreds of individuals with only small contributions to the project[75].

Some research has also attempted to bridge the worlds of Open Source and models of team work from organizational behavior literature. Crowston et. al. proposes a model of hierarchical participation, with varying levels of core developers, co-developers, active users, and passive users[18]. Beyond this, much research has addressed issues with the public goods nature of Open source and considered why under-provision is not more com-

mon in the community. Von Hippel and von Krogh address this question by speculating that Open Source implements a hybrid cooperation model dubbed the "private-collective", where innovation happens at an individual level, but is then shared with the collective community[124]. This view is shared by Osterloh and Rota who speculate that the success of such a collective is due to the norms of the communities and licensing of the software, but they caution that development in patent law may hinder future growth[86].

Significant research has also been done on the population of developers who participate in Open Source projects. However, much of this research is reliant on the belief that most Open Source developers are volunteers. Lakhani and Wolf conducted a survey of 684 developers in 287 different projects where it was found that intrinsic enjoyment of participating in Open Source software was a primary driver of participation[62]. Ghosh et. al. conducted a much larger survey of Open Source developers in which any individual was allowed to participate. Their work found that for many Open Source developers patterns of behavior resembled those of hobbies and other activities driven by intrinsic motivations. However, they did find a small number of developers for whom their activity strongly resembled that of professional developers[40]. Additional research examined learning as a primary motivating factor for many developers[133] and suggested that financial compensation was a low priority[3].

Ideology of individual developers has also been identified as a key component of participation in Open Source. Many developers cited their "belief" in Free Software as a motivating factor participation[40]. Research by Stewart and Gosain found that Open Source teams

may be too adherent to social norms of not forking source code, and therefore wait until the team achieved consensus before taking actions. Likewise, they found that teams that stressed the "freedom" of using Open Source were likely to encourage individuals to participate in the broad community, rather than focusing their time and skill on a smaller set of projects[106]. These findings echoed evidence from Kogut and Meitu that the governance of projects effectively discourages forking and helps to enforce a consistent ideology[58]. Further research by Stewart et. al. found that projects with more liberal Open Source licenses, typically those that stress the "freedom" aspect of Open Source less, attracted greater interest[105].

In a similar vein to the survey work and broad community analysis, Madey et. al. analyzed the community around SourceForge.net, the largest Open Source project hosting website, to generate large scale social networks of project membership in the Open Source community. Their analysis found that projects attracted developers in a pattern that roughly followed a power law distribution, with many projects that had only a single developer and very few projects with many developers[67]. Crowston and Howison performed a more in depth analysis of 120 projects hosted on SourceForge and found that while large projects do exist within the community, individual developers frequently serve the role of linchpins for large amounts of participants on the periphery of the community[19].

More recently, research has examined how economic incentives and commercial firms play a role in Open Source participation. Lerner and Tirole speculated that participation in Open Source acted as a signaling effect for potential employers – giving a developer a

chance to display and improve their skills beyond what would be possible in more conventional a work environment[64]. Mustonen proposed that developers in open source communities extracted economic benefits from their jobs and that this recognition fed back into the projects[78]. Roberts et. al. followed numerous developers in the Apache Software Foundation and found that developers who were elevated to the status of Apache Software Foundation Member, were likely to receive increased compensation from work[96].

As the Open Source movement has grown, so to has the commercial interest in Open Source. While at first commercial ventures and Open Source may be at odds, there are many ways in which they compliment one another. Von Hippel examined Open Source communities and proposed that commercial organizations could benefit from allowing user communities to innovate on top of commercial solutions, drawing a parallel to individual developers testing out new ideas on small sections of Open Source project code[122]. Mockus et. al. proposed that hybrid commercial/Open Source projects can be successful, but they need to adopt Open Source strategies of small teams and well componentized structure for maximum success[75].

O'Mahony addressed some of the tension between Open Source projects and commercial firms. She speculated that Open Source projects chose to band together in a foundation because of the increased protection for trademarks and project code that the pooled resources of a foundation would have against possible exploitation by commercial firms[82]. West and O'Mahony presented research on spinning out Open Source projects from previously proprietary code, a trend that continues to develop and gain momentum. They

contrast commercially created Open Source with community created projects and note that while commercial projects are more likely to have adequate resources for project infrastructure and marketing, they may find it difficult to attract developers with sufficient skill to work on the project[128]. Stewart et. al. examined a number of Open Source projects and found that when a large organization sponsors a project, it will attract additional interest than those without sponsors. However, they note that non-market sponsors attracted more individuals than market based sponsors[105].

While the work of West and O'Mahony and Stewart et. al. begins to examine how commercial interests work in Open Source environment, it focuses on single projects, and neglects the trend toward large communities with many commercial players, such as the Eclipse Foundation and Symbian Foundation. Indeed, as software continues to increase in complexity and costs continue to spiral, more and more firms will turn to Open Source and collaborate across organizational boundaries.

This thesis seeks to expand the knowledge around these emerging Open Source communities by examining how communities with multiple commercial firms interact with governance structures, amongst firms, and with individuals.

1.3 Overview of Thesis

This thesis presents four empirical studies that advance our understanding of Open Source software development and the communities that build and support the software. The work is

based on the observation that many mature Open Source communities have three different levels of players involved in the community: a non-profit foundation that owns the rights to the project code and other intellectual property, firms that contribute financially to the project and provide developers to work on the project, and the individuals who actually write the code and work together to make the community function.

The community around Open Source has grown to the point where many of the most successful projects are not independent, but rather parts of large scale ecosystems guided by non-profit foundations that work to foster collaboration and create value around a set of Open Source projects. Whereas much of the prior research has focused on individual projects or firms, understanding these new communities requires a holistic view of the entire community – firms, projects, foundations, and individuals. I begin with a qualitative study in chapter 2 based on interviews with representatives of the Eclipse Foundation and the member companies of the foundation. These interviews are analyzed to understand what actions the Eclipse foundation undertakes to promote a healthy and vibrant ecosystem and to foster collaboration amongst member companies.

Chapter 3 presents a second empirical study on the Eclipse Foundation that examines how firms actually collaborate in the Eclipse ecosystem. I use archival data from the project source code repository and information from project intellectual property management logs to identify which firms work on each project within the Eclipse ecosystem and to what extent those firms actually collaborate. General patterns are observed about the amount of actual collaboration occurring in Eclipse, both in terms of number of projects firms are

involved with, number of firms involved with each project, and the distribution of contributions across firms for each project. These results are compared with data from the GNOME ecosystem that show a very different pattern of collaboration around core components. This research fills several gaps in knowledge about Open Source ecosystems. First, it provides an overview of contributions and collaborations across firms in an ecosystem, providing a corollary to previous research on individual contributions[75]. Secondly, it provides insight with regards to the stability of Open Source communities – communities that are heavily reliant on a single firm face significant challenges if that firm leaves, while communities that share responsibility for core components amongst many firms have much higher coordination and collaboration needs.

Chapter 4 reports a study of the interactions between firms and individuals. Although many prominent Open Source projects began as commercial projects, there are a numerous examples of those that began as hobbyist and non-commercial projects. When commercial firms find value in these Open Source projects previous research suggests that their presence could either attract or disenfranchise existing members of the community. Using a substantially volunteer community, the GNOME project, I examine the extent to which commercial participation in an Open Source community affects volunteer participation at both the project and module level within projects. This provides valuable insight for commercial firms considering contributing to Open Source and also assists volunteer community managers in understanding how best to work with commercial firms that have shown an interest in their project.

Finally, in chapter 5 I advance the state of knowledge of individual interactions and cooperation by expanding the socio-technical congruence (STC) metric. I first validate the use of STC in an Open Source context, showing that when communication is aligned with coordination requirements defect resolution time decreases, as was previously found in a commercial context by Cataldo et. al[15]. I then explore the implications of several modifications to the metric, including weighing edges, modification of the formula, and addition of decay metric for long term observation of projects. Finally, I perform a large scale sensitivity analysis to understand the potential problems with collecting data from Open Source contexts and how missing and wrongly-inferred data affect the stability and viability of the metric.

The thesis concludes in chapter 6 with a set of recommendations for individuals, firms, and other organizations participating in Open Source communities

Chapter 2

Firms and Foundations: Guiding an Ecosystem To Promote Value

Early Open Source projects were comprised primarily of individuals working disparately with their own sets of goals that happened to align and form a community. Central coordination was sometimes absent, or managed only by a single central individual[59, 93]. The unit of participation was the individual, regardless of whether or not they worked for a commercial firm. Most of these projects had little commercial value or marketability and therefore lacked the need for more advanced governance structures.

However, modern Open Source projects are very different. Recent acquisitions of Open Source projects by commercial firms have frequently been over $100 million and Sun MicroSystems' purchase of MySQL was $1 billion[95]. With such high commercial values,

Open Source projects have experienced a need to preserve and protect the intellectual property for the project source code. This has led many popular Open Source projects to evolve and adopt new forms of government. The largest communities such as Linux, Apache, Mozilla, Python, Eclipse, and GNOME all have non-profit foundations that protect the intellectual property of the project and coordinate interaction with commercial firms[83]. Despite this, there has been relatively little research on how these foundations interact with firms. In this chapter I briefly review some of the various models of Open Source governance that led to the creation of non-profit foundations. I then report on the results of an empirical study of Eclipse, a large Open Source ecosystem with a foundation form of governance. I identify the primary functions that the foundation performs and how those actions serve to benefit the commercial members of the foundation.

2.1 Governance and Intellectual Property

Popular portrayals of Open Source have typically focused on the lone super programmer working for long hours in isolation to create a magnum opus. This is the pattern or work that gave rise to original versions of the GNU Compiler Collection, EMACS, and the Linux kernel[76]. However, with very few exceptions, even the best programmers work with a number of other individuals. Together these individuals form a community, with norms, processes and values, and all have a stake in the development of the software and the protection of the rights associated with the software[125].

Within Open Source, the issues of governance and intellectual property are often closely related. When starting a project, one of the first decisions that is made is the license of the project source code. Unfortunately for Open Source developers, there exists a myriad of Open Source licenses each of which implements subtle differences and incompatibilities[98]. In addition to the choice of license, which can have a dramatic effect on the ability of a project to attract new participants[105], projects must choose how to manage their intellectual property and who will own the rights to the software[120]. Often there exists a direct relationship between the ownership of the code and the governance structure in the community.

There are a number of ways in which the issue of rights in Open Source software are handled. Many smaller programs tend to ignore the rights issue all together, simply accepting contributions from all individuals without performing any sort of diligence on where the contributions came from or requiring that the participant sign over their copyright to a controlling body. This model is very common in volunteer projects with only a single primary developer who may not be fully aware of the legal issues around software rights and licensing, or where insufficient resources exist to manage the resources[30].

Another model, which was used by Netscape after the release of the code to their Netscape web browser, which has since provided the groundwork for the very successful Firefox web browser, was to allow individuals to retain the rights to their code contributions and include those contributions in the distribution of the software only if they were the same license as the rest of the code. While this provides full rights for contributors,

as they continue to own the licenses to their own contributions, it makes it difficult for a project to make even small changes to its license, something that was all too apparent when the maintainers of the Mozilla code base wished to update the licenses on the software and many original code contributors could not be reached to verify that their contributions could be relicensed[68].

An alternative to these two ownership models is to have a single entity own all the code and require individuals to assign some of the rights of their code to a third party. This is a common model for mature projects and is used in OpenOffice.org and by many projects from the Free Software Foundation. The recipient organizations then agree to take a role in protecting the intellectual property of the project and ensuring that the work remains open and accessible for all[83]. Often times, but not always, this recipient organization is a non-profit foundation.

2.2 Foundations in Open Source

Even though many of the responsibilities of an Open Source foundation are similar across communities, they vary greatly in the degree to which they manage the activity and development of the project. The GNOME Foundation, which oversees the GNOME Desktop Environment, discussed in more detail in chapters 4 and 5, has generally taken a very hands off stance toward project development. While members of the foundation board of directors are elected by individual members of the foundation, the board makes no decisions

regarding project code. Rather, the board manages the legal aspects of the project, ensures the project servers remain accessible, and organizes the annual conferences. Individual projects within the community are largely free to use their own development methodologies, release code at their own pace, and decide who can contribute directly to project source code archives. In this way individual projects and project maintainers retain the most control over the project. The foundation is quite small and has few employees – most funds from corporations are devoted to bringing developers together to collaborate. Membership in the foundation is merit based, reserved for individuals, and is independent of corporate employment. The majority of the budget comes from a handful of corporate sponsors who choose to join the GNOME foundation advisory board – however, this role does not guarantee influence over the community.

The Apache Software Foundation (ASF), which oversees the development of the Apache Web Server, among other projects, functions similarly to the GNOME foundation, but also enforces some norms of software development on the projects within the foundation. Apache has a highly developed "incubation" process for new projects, that sees potential Apache projects mentored by experienced ASF members to ensure that development is proceeding properly and the rules and norms of the community, such as making all decisions over mailing lists, are followed[27]. However, for the most part ASF allows different projects to decide their own direction and plan their own releases. Like the GNOME Foundation, the ASF takes a significant leadership role in the annual conference for the community, ApacheCon.

Still other foundations focus on only a particular project, such as the Python Software Foundation and Linux Foundation. Both of these organizations serve primarily as holding bodies for the rights related to the software, and do very little in terms of setting development policies. While these foundations engage in limited marketing and create focus groups for specific issues, their overall involvement with the project beyond holding the intellectual property is minimal[83].

The Eclipse Foundation, which manages the ecosystem around the Eclipse integrated development environment and associated tools, is the logical evolution of Open Source foundations. Like many other Open Source foundations, the Eclipse Foundation is a not-for-profit entity that has rights to the code and organizes events for the community. However, rather than working primarily with independent software developers and having membership comprised of individuals, members of the Eclipse Foundation are corporations and institutions. Individuals representing those entities are the ones who do the primary work. In this way, Eclipse has an explicit commercial focus that other communities typically do not[110]. It also reflects the commercial success of the Eclipse Foundation[37, 42]. It is precisely this interaction between the not-for-profit foundation and the firms in the Eclipse ecosystem that is interesting, because of it's uniqueness, strength, and the fact that it serves as a role model for other Open Source foundations, such as the Symbian Foundation.

While there exists some literature on the role of foundations in Open Source [82, 83, 128], there is little research on how foundations interact with commercial firms in Open Source ecosystems. Given the prominence of Open Source foundations which focus on

corporate members rather than older models of individual membership, such as the Eclipse Foundation, Linux Foundation, and LiMo foundation, it is critical to understand how these foundations operate and how they provide value to member firms, which often must pay substantial membership fees. This chapter seeks to understand how an Open Source foundation attracts members, develops members, and provides continuing value for members through an empirical study of the Eclipse ecosystem.

2.3 Description of Data

To better understand the Eclipse ecosystem, a combined qualitative/quantitative strategy was pursued. The qualitative component of the research consisted of 38 interviews with 40 individuals employed by member companies of the Eclipse Foundation and the Eclipse Foundation itself. These interviews began in November 2006 and the final interview was conducted in July 2008. A snowball sampling strategy was used beginning with individuals employed at the Eclipse Foundation, including the executive director and directors of marketing and ecosystem development. Introductions were graciously provided by the executive director of the Eclipse Foundation and served to open doors to additional companies and individuals. After the first round of interviews concluded in 2007 we went back to the Eclipse Foundation for followup interviews and to obtain introductions to several firms that we had previously been unable to contact. Interviews were typically done via

telephone and consisted of the researchers and a single interviewee[1].

Additional individuals were interviewed using opportunistic sampling at the 2007 and 2008 EclipseCon conferences in Santa Clara, CA. This three day gathering of the Eclipse community consists of technical presentations, training sessions, formal membership meetings, an exhibit floor, and a small store marking Eclipse books and products. This conference is typically attended by approximately one thousand individuals – a significant portion of which are developers who write the code that makes up the Eclipse ecosystem. At this conference efforts were made to seek out individuals that represented view points that we had thus far been unable to locate. Much of this was done through the use of "topic tables" at lunch, and strategically attending sessions presented by individuals who may prove helpful. A handful of informal interviews were conducted at EclipseCon, these interviews were not recorded, but extensive notes were taken. The remainder of the individuals identified at EclipseCon were scheduled for telephone interviews at a later date. A table summarizing the interviewees is presented in table 2.1. Some interviewees were interviewed multiple times, and a handful were group interviews.

Finally, I have attended and presented at the Eclipse Foundation's annual membership meeting. This afforded a chance to present preliminary findings and obtain feedback on many of the ideas and conclusions resulting from this research. The membership meetings serve a variety of purposes in the community including introducing new members, review-

[1]There were two interviews in this set that had multiple interviewees in a single interview. Although this was a deviation in our sampling and interviewing strategy, the value of the information from the perspective of those companies was judged to be more valuable than the potential damage to the interview strategy.

Table 2.1: Summary Information of Interview Participants

Total Interviewees	40
Total Interviews	38
Telephone Interviews	24
In-Person Interviews	14
Number of Firms	15
Developers	24
Executives	6
Other Role	8
Volunteers	1

ing the goals of the boards and councils that make up the Eclipse Foundation, and voting on any changes that require ratification by members of the Eclipse Foundation. The meeting at EclipseCon is the primary in-person membership meeting, with other meetings taking place via teleconference. Minutes of meetings are made available on the project web site, allowing me to review meetings that I was unable to attend.

2.3.1 Interview Methodology

Telephone interviews were scheduled at the convenience of the interviewee and usually involved at least two researchers taking notes[2]. In-person interviews were conducted on-site at EclipseCon with only the interviewee and author. Interviews were semi-structured with a set of general background questions about the interviewee and their role in the Eclipse process asked of all interviewees. Some of the commonly asked questions were:

- How did your organization decide to start contributing to Eclipse?

[2]One interview with a volunteer in the Eclipse community had only one researcher, the author.

- What are the major ways that your organization contributes to the Eclipse project?
- How does your organization decide what to contribute as Open Source code to Eclipse and what to keep proprietary?
- How do you work with your competitors in Eclipse?
- How do you work with IBM in the Eclipse Ecosystem? If you've worked with other large vendors of development environments (e.g. Microsoft), how does this experience differ?
- In what ways does your organization work with Eclipse Foundation?
- What are the benefits of being a member of the Eclipse Foundation?
- How does your experience in the Eclipse foundation compare to experiences in other Open Source environments?
- How do the "4 values" of Eclipse affect your organization?

At the end of the interview each interviewee was given a chance to bring up any additional issues they believed would be helpful for our research. Each interviewee was also asked if there were other individuals who we should seek out for feedback and additional interviews and if they would be willing to provide a letter of introduction. While this biased the sample in favor of social contacts, it also dramatically increased the response rate and success of the study relative to contacts without such direct introductions.

Interviews were recorded and coded based off the notes and recordings. Where needed, portions of the interviews were transcribed for further analysis. Coding of interviews was

broken into several categories corresponding to major themes of the interviews:

- Central Functions: General functions that Eclipse Foundation fostered for the community. Major sub-categories were governance, enforcement of rules, roadmapping/planning, and other (fund-raising, marketing, etc).
- Distributed Functions: Functions that were important for the Eclipse ecosystem, but generally done with little intervention from the Foundation, such as creation of code, firm and project selection, selection of committers.
- Business Strategies: Ways in which the firms participating in Eclipse can extract value from the ecosystem. Major sub-categories were product strategy, what to reveal, selection of project, how to make money, and other benefits and competitive tactics.

After coding of interviews, the data were analyzed for trends across interviewees with special focus paid to sections of the interviews that discussed the relations to the Eclipse Foundation. The major aspects of this relationship were found in three different major categories: the design of the Eclipse Foundation and community around it, the stated purposes of the Eclipse Foundation, and the actions that the Eclipse Foundation undertakes to drive value to member companies. In the next sections I examine how interviewees viewed each of these topics.

2.4 Community Design in Eclipse

Initially, the community around Eclipse was not much different than a standard industry consortium. In 2001 when IBM first released the code to Eclipse to the world, they sought to create a community around the project, the Eclipse Consortium. The Consortium was structured in a way that reflected the license of Eclipse at the time, one which allowed anyone to utilize the code to Eclipse and to freely expand it. The initial members of the consortium were primarily firms that had previously been involved with IBM's suite of VisualAge tools and were already active in IBM's developer ecosystem. For many of the smaller members of the Consortium the change to this open model was viewed as a great success that would give them better access to the internals of IBM's tools and allow them to better compete with larger competitors. Larger firms also saw benefits in the consortium construction as it allowed them to easily access all of the code for Eclipse under a single uniform license, without the need to relicense differing parts of code for further releases.

At the dawn of the consortium, the code was 100% created by IBM and IBM retained the rights to lay out the roadmap for the project and approve the architecture of the project, in a very similar manner to their previous ecosystem with the VisualAge suite of tools. A major change, however, was the newly found ability of firms to contribute directly to the Eclipse code. Rather than requesting an API modification or enhancement and waiting and in the hope that IBM would see fit to create the enhancement, firms were free to implement the modification themselves and then submit the changes back to IBM for inclusion into the

main code tree for Eclipse. Furthermore, the licensing agreement allowed firms to begin these efforts and distribute these efforts without permission from IBM – allowing greater innovation to occur at a more rapid pace. As one executive of an early participant in the community put it, "The Open Source license of Eclipse allowed us to have just one license agreement for everything and not [individually] negotiate a license."

While the license allowed firms to avoid the pain of individually negotiating licenses for every project, there were still significant issues with the license and the structure of the community. The license under which the Eclipse code was released, the IBM Common Public License, contained several terms that individuals found difficult to handle, including a patent grant clause that required contributors to grant an unlimited royalty free license to portions of their code they contributed to Eclipse. There was also some confusion amongst early firms in the consortium about the requirement to assign copyright to IBM for the pieces of code they contributed. It is not clear whether this was ever a formal policy of IBM, but an individual who had been involved with Eclipse since well before the Consortium was founded indicated that IBM requested copyright assignment because of the nature of their business around the Eclipse project which was substantially different from other Open Source business models of the time that primarily consisted for redistributing and repackaging Linux[3]. IBM's strategy was to build upon the Open Source project and sell it

[3]The Linux Kernel is licensed under the GNU General Public License, version 2. This license allows individuals to make any changes they wish to the code of a software project, however if the modified version of the project is distributed the source code to those modifications must also be included. This is in contrast to other more "business friendly" licenses, such as the Apache Software License and the BSD license that allow a company to redistribute a modified version of the software while keeping the source code to those modifications private. The early license of Eclipse, the IBM Common Public License did allow commercial use and sales similar to the Apache license.

as a greatly enhanced tool in WebSphere developer studio. Ownership of the copyrights to all code was seen as a step that would make distribution of the modified product easier.

In addition to issues with licensing, the original structure of the Eclipse Consortium placed IBM at the top of the hierarchy and afforded IBM the ability to road map the product and decide on all architectural innovations. According to one participant it "seemed just like the old model of working with IBM, except now there was only one license. We still lacked a say in the direction of the project and were subject to [IBM's] whims". In response to many of the issues raised by members of the Eclipse Consortium and the changing models of Open Source governance, in 2004 IBM shepherded the Eclipse Consortium in the transition to the Eclipse Foundation. This change addressed both issues of governance and ownership in the Eclipse community by creating the new Eclipse Foundation and making it the ultimate arbiter of the community. In this process all of the intellectual property rights of the Eclipse code base which had previously been vested with IBM and internally valued at more than $40 million were donated to the new foundation[131].

The new foundation adopted a hierarchical structure guided by a board at the top level with a series of councils and boards underneath. At the top level, the Eclipse Foundation Board was composed of designated members from Foundation member companies – primarily companies that were contributors at the highest strategic level. A smaller number of individuals were elected by the lower level member companies and to represent to the committers who wrote code for the project. Beneath this are three primary councils - the requirements, architecture, and planning council. As defined by the Eclipse foundation,

their purposes are as follows[112]:

- **Requirements:** The Requirements Council is responsible for capturing and organizing requirements for all of the projects in the Eclipse community. The Requirements Council reviews and categorizes all of these incoming requirements - from all residents of the Ecosystem - and proposes a coherent set of themes and priorities that drive the roadmap.
- **Planning:** The Planning Council is responsible for establishing a coordinated Platform Release Plan that supports the roadmap, and balances the many competing requirements. The Platform Release Plan describes the themes and priorities that focus these Releases, and orchestrates the dependencies among Project Plans.
- **Architecture:** The Architecture Council is responsible for the long-term technical health of the Eclipse platforms and frameworks. More explanation of the Architecture Council can be found in the Eclipse Development Process and in the guidelines and checklists for the Architecture Council.

It is worth noting the composition of these councils that direct the community. The requirements council is composed entirely of individuals representing strategic developers and the Eclipse Foundation. The planning council which has a slightly more technical role still has many representatives of strategic developers, but also an individual from each of the project management committee from each top level project. Finally, the architecture council, which handles very in-depth technical issues and mentors upcoming projects has a small number of representatives from strategic developers, with many appointed repre-

sentatives from add-in providers to ensure expertise over the complete ecosystem. Interviewees believed this led to a much more open and accessible community. However, it did incur more work for the individuals in the community, leading to additional stresses especially upon smaller firms which may have executive officers as their representatives on the various boards.

At a more technical level, each of the projects within Eclipse is managed by a Project Management Committee (PMC). The PMCs serve to ensure that a project is healthy and guide the project in its development. The initial lead for new PMCs is appointed by the board and that PMC lead then selects the initial members of the PMC. Additional people can be added to the PMC by a unanimous vote of the existing PMC members. All PMCs must operate under the rules of Open Source engagement which stress meritocracy, transparency, and open participation as primary values[114]. These values mirror the values of the Eclipse Foundation itself and will be discussed more in section 2.5.

Perhaps one of the most remarkable aspects of this transition is that rather than evolving organically, the community around the Eclipse Foundation was planned from the beginning and it has managed to continually release and improve upon itself since its genesis. In contrast, many of the most prominent Open Source communities have evolved organically, adding structure over time as they needed it. The communities around Apache and GNOME both have exhibited such development[38]. As a result of this organic growth, in those communities the development methodology drives the foundation and the rules and values are embedded in the development community rather than codified in the foundation

as is the case with Eclipse. This design strategy is very similar to the strategy employed by various projects from Sun MicroSystems, specifically around the OpenJDK and OpenSolaris communities, however, those communities have not flourished to the degree of the Eclipse community [4].

2.5 Dominant Purposes of the Eclipse Foundation

The foundation operates using two major tools to guide its evolution and the evolution of its partner companies: a set of four core values of the Eclipse Foundation and an Open Source Maturity Model from Carleton University[13]. Those values as enumerated by foundation chair Mike Malinkovich are briefly described as follows:

- Openness - the code and all other artifacts for the project are available for examination and use by anyone
- Transparency - all decisions within the project are recorded and available for public review.
- Meritocracy - roles within the project are given on the basis of contributions directly to the project and not on any other criteria.
- Permeability - projects are open to new ideas and implementations

[4] Sun MicroSystems hosts a variety of Open Source projects including the very successful MySQL and OpenOffice projects. The OpenJDK and OpenSolaris communities are fairly new projects and rather than allowing a community to grow organically, they took an approach similar to Eclipse and mandated a structure, however lacking a community to initially participate, these communities have quickly become bogged down in bureaucracy their success is not certain.

Each of these four values were enforced through a set of rules and conventions given to projects and member companies in the Eclipse ecosystem. While most of the time projects had little problems following these rules, members of foundation pointed out that the foundation had the "ultimate stick" of expulsion from the community.

One of the biggest problems that firms raised about the four values is that they are, to some degree, in opposition to many forms of business. For example, almost every firm that marketed products based on Eclipse described an internal struggle to decide what was going to be open source and what would remain proprietary. On the one hand, Open Sourcing a component could give the business a great competitive advantage by allowing that firm to dictate the direction of Eclipse for a small component. However, if they chose not to donate the code, there was the possibility of monetizing the code either as an independent project, or as part of a larger software release.

While many of the interviewees were familiar with the four values of the Eclipse Foundation and sought to work them into their daily practices, the relationship to the Open Source Maturity Model[13] was a bit more diffuse. The model, pictured in figure 2.1 tracks the progression of firms those that deny the use of Open Source to those that utilize Open Source to redefine the market place and provide additional value to their customers.

The model proposes six different levels of Open Source use and adoption and positions bubbles for each level to indicate rising commitment and profit from Open Source utilization. The assumption is that firms gradually move from the lowest level to the highest level as a natural progression of their exposure and use of Open Source. At the lowest level are

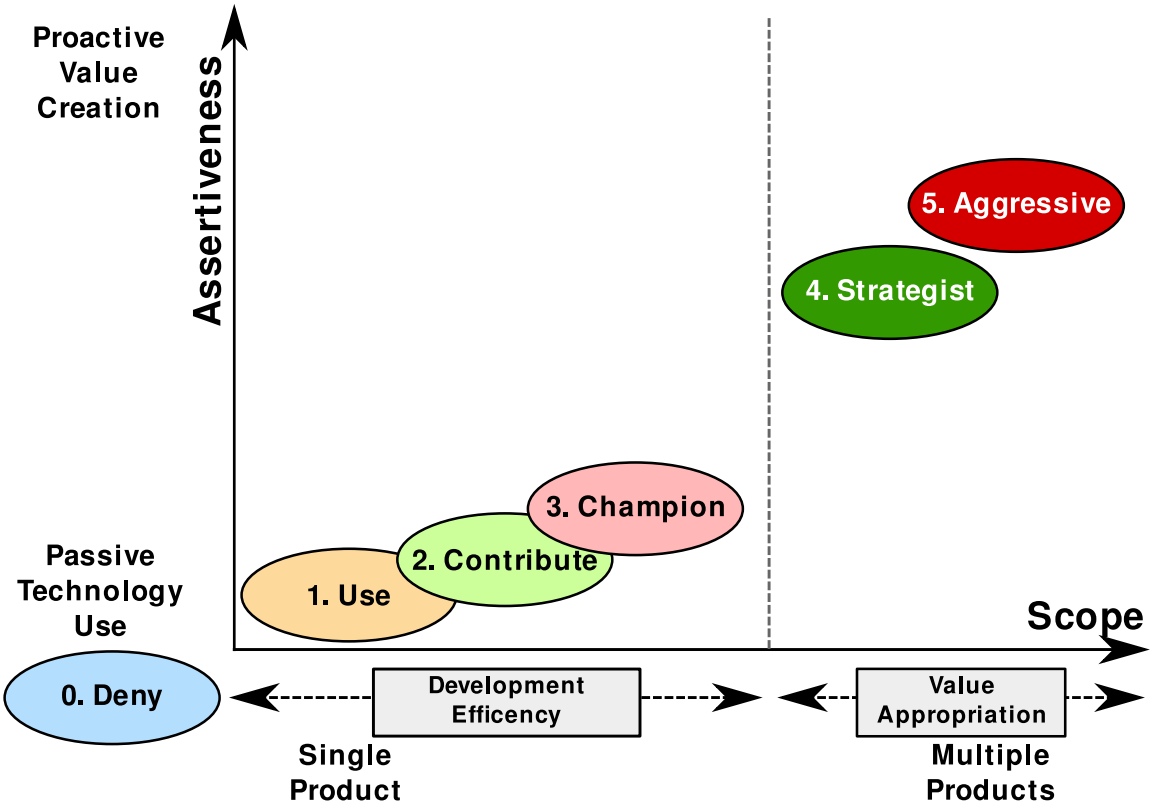

Figure 2.1: The Open Source Maturity Model[13]. Employees of the Eclipse Foundation frequently talk of working with member companies to advance them to higher levels of membership.

those firms that use no Open Source and deny its impact on their company. The next three levels are organizations that utilize Open Source software and contribute back to the project communities, but are doing so mainly out of pragmatic development reasons. After a period of simply using the software, firms begin to contribute to the software so the software addresses more of the concerns of that firm. After contributing, a firm at stage three begin to champion support of the project and build an ecosystem around the project.

Moving beyond simply using the software, at the strategist level executives in a firm have committed to making Open Source a major focus of their business strategy. For example Sun MicroSystems and their commitment to making Java and the Solaris operating system Open Source may be considered a strategist in the model. At the highest level are the firms that have adopted an aggressive strategy for Open Source and leverage participation across multiple projects and ecosystems, seeking synergies to extract multiple value from their participation. According to the model author, very few firms fit into this, however, IBM, the original benefactor of Eclipse is perhaps the best example.

Although this model was frequently highlighted by employees of the Eclipse Foundation in presentations and meetings, many employees of member companies seemed unaware of it. Of those that were aware of the model, two firms self-described their pattern of action as one that is moving through different stages of the maturity model, none said they felt that the Eclipse Foundation itself was pushing them. A large firm that several years ago had no relation to Open Source but now self-identified as being between “strategist” and “aggressive” on the model indicated that their movement between different levels wasn’t

because of the Foundation's actions but because of the changing market and the way that the Foundation had pushed Eclipse to become the dominant tool set for Java tools.

2.6 Driving Value Creation

Membership for commercial firms in the Eclipse Foundation costs between $5,000 and $500,000 a year, plus a possible commitment of developers to work on Eclipse related technologies. A member of the Eclipse Foundation staff indicated that sometimes he was "shocked" when less active member companies sent checks in every year. This section examines concrete ways in which the foundation drives value for its member companies and why, even if a company is only marginally active, they continue to be members of the Eclipse Foundation.

2.6.1 Non-Market Player

One of the primary methods the Foundation drives value for the members companies is through the non-market nature of the foundation itself. While it publishes Eclipse and hosts the official repositories for the Eclipse source code and binaries, there is no attempt to sell modified copies of Eclipse. This allows a number of smaller firms such as Genuitec and Innopract to freely participate in the community by creating Eclipse "distributions" without the fear of the Eclipse Foundation creating a competitive product. This is in marked contrast to the old Eclipse Consortium structure in which IBM was actively monetizing Eclipse and

also had control of the intellectual property around the Eclipse source code. Rather, it is nearly the same strategy that is employed around the Linux kernel, which for years was managed through a set of ad-hoc organizations and now is formally managed by the Linux Foundation and relies on a set of Linux distributors to package up Linux and distribute it to users and developers.

2.6.2 Introduction of Process

A pleasant side-effect of the non-market nature of the Eclipse Foundation is that it allows the foundation to exert quality control in a different manner than an organization controlled by a market player. One of the key ways this is exerted is through the standard set of processes that is mandated across all projects. By adopting a set of practices that is not based solely off the practices of the dominant organization, the Eclipse Foundation can ensure that all participants understand the process and no one firm has an undue advantage because of the formalities of the process.

One of the major aspects of the set of processes around the Eclipse is the project mentoring and review cycle. In order to create a new project within the Eclipse ecosystem firms must submit a plan that details the functionality of the new project and lists what firms support the project. Although it is stated that a new project should show interest from more than one company, in practice many projects have the entirety of their code base from a single firm (more information on firm to firm collaboration is discussed in chapter 3).

A recently controversial role that the foundation has been forced to take is to remove dead projects from the ecosystem. In Fall 2008 the Eclipse Foundation chose to archive two projects that were no longer being developed, despite the fact that there were commercial application built based on the technology. However, with no one currently contributing to the code, employees of the foundation indicated that it would be improper and not-consistent with their process and standards to leave abandoned projects in an indeterminate state. While controversial, this action continued to enforce the standards and norms of the foundation and may have served as a notice for firms that seek to focus on commercial marketing of products based on Eclipse source code without contributing to the Open Source aspect of the community.

2.6.3 The Value of the Eclipse Brand and Joint Marketing

One benefit that was almost universally highlighted was the value of the Eclipse brand name and how being a part of Eclipse allowed firms, both large and small to better market their products. As Eclipse has solidified its position as the dominant Java IDE, many firms have adopted Eclipse technologies for their own IDEs. A representative from one firm that used to ship no less than thirteen different IDEs highlighted how switching to a new Eclipse-based framework was initially painful, as not all the custom features from each of the older projects was available in the initial release of the new Eclipse based tools. However, customers were quickly impressed once they heard it was based on Eclipse and discovered that developers with skills in Eclipse could easily transfer them to the new tools.

Indeed, there were several smaller companies who indicated that one of the primary reasons for their involvement in Eclipse was to harness the power of the brand name and the community. This was particularly common for new startup firms who were still seeking to get customers. The CEO of one of these companies highlighted how his company had an idea and the beginning of a product, but nothing they felt they could show publicly at a large scale industry trade show. After joining the Eclipse foundation they were given an opportunity to provide an introduction about their phone at the upcoming member meeting, which directly led to their first customer, another Eclipse member company that had heard their introduction. In fact, four out the first five customers for this small firm were as a direct result of either presenting at the Eclipse members meeting or having their company information available on the Eclipse website.

Members within the foundation also coordinate their marketing and market research efforts. Early in the life of the ecosystem, much of the marketing and research was done by consortium members who agreed to share the results with member companies that contributed to the costs (time, manpower, and money) of doing the research. Once the ecosystem transitioned from the consortium to Eclipse Foundation, this has become a major role of the foundation. The foundation now conducts an annual marketing survey of users of Eclipse to understand how the ecosystem is developing and how developers use Eclipse. This survey, which is made available to foundation members, tracks usage of particular components, identifies programming languages Eclipse is being used for, and more recently has begun to address how companies are using Eclipse based technologies outside

of the IDE.

From time to time the Eclipse Foundation also promotes activities by member companies. For example, a substantial number of the member companies in the Eclipse ecosystem focus on training either in the use of Eclipse, or how to develop using Eclipse based technologies. Several times the Foundation has coordinated global training sessions that feature these firms in their local environments. A representative of one of these firms indicated that the global push of the foundation's effort made it much easier to attract participants than if they just ran a training session by themselves.

The foundation also organizes webinars based on particular technologies. A member company can choose to sponsor one of these events and in exchange they receive contact information for the participants in the webinar. Smaller companies that create developer tools for the Eclipse ecosystem found the leads from these webinars to be particularly helpful as they not only had a chance to introduce their product to potential users, but also their full contact information for sales followups. Although we were not able to get direct sales figures from member companies who took advantage of this, an executive at one member company described the sales as substantial.

A final common thread illustrating how the foundation supported the ecosystem with marketing and branding is through the planning and management of the major Eclipse conferences, EclipseCon and Eclipse Summit Europe. Obviously, these venues provide opportunity for sponsors to market their products via sponsor booths, but a more subtle form of marketing takes place in the technical sessions, which one interviewee termed

"marketing by osmosis".

> The sessions at EclipseCon are really important for us. Even though we're not out there selling [product name], it obviously plays a role in our presentations and we get customers through that. Last year I gave a talk where I never mentioned [product name], but I still had an opportunity to get new customers because people knew [company name] made [product name], and they asked me about it.
>
> –CEO of small Eclipse member company

Another representative of a firm that was not an Eclipse member company, but still sent representatives to EclipseCon liked the fact that the foundation promoted interaction amongst engineers at EclipseCon. He highlighted how this interaction between engineers was far better than any typical marketing presentation and it made it easier for his firm to identify products the would want to license for their own use and development.

2.6.4 Organizational Structure Driving Value

The original structure of the Eclipse foundation had two different "strategic" membership levels, strategic developer and strategic consumer. Both roles were granted seats on the board of the Eclipse foundation, however dues were less for strategic developers as they had an additional commitment of developers toward the project. Strategic developers always outnumbered strategic consumers and today there is no longer a distinction between the two roles made on the project website and most of the strategic consumers have dropped

down to add-in provider status. The strategic consumer role was originally developed to provide firms who could not, or chose not, commit the developers and resources necessary to become a strategic developer to still exercise influence in the Eclipse community. This influence was given to them by the virtue of the board seat strategic consumers obtained. A representative from one form strategic consumer highlighted some of the reasons why this may have not worked as well as first thought:

> The role was supposed to give us additional access and help steer the Eclipse ecosystem. But in the end, it never did because we never had the developers to contribute to get things done.
>
> –Former Strategic Consumer

This perception that the roles within the ecosystem cannot be bought relates strongly to the meritocracy value of the Eclipse Foundation. It provides a sense that everyone is playing by the same rules and acts as an equalizer for small firms, so they need to not be as concerned that a large firm will try to swoop into the community and adversely affect the direction of the Eclipse ecosystem. However, one individual from a large corporation indicated frustration at the fact their voice was only heard in proportion to their contribution to Eclipse, and not to their overall market influence. He believed that his firms expertise in the broader market could be beneficial to the Ecosystem, but because they were only lightly involved with Eclipse, they were largely ignored.

2.6.5 Platform for Innovation

Perhaps the biggest change that the Eclipse ecosystem has experienced is the switch from being a community around a single product, the Eclipse IDE, to being a community around a platform, Eclipse and associated technologies[131]. This development is partially the logical extension of building a community around the Eclipse IDE and partially a result of the hard work of the Eclipse Foundation in attracting firms that are willing to extend Eclipse in new ways. An interviewee from a firm that was working with a firm that was working in a completely novel space highlighted the ecosystem and the platform as a major reason why they chose to work within the Eclipse ecosystem rather than working with another community, such as Apache. One of the greatest advantages of this for the small firm was that their participation in Eclipse opened up the possibility to partner with other much larger firms, such as BEA and IBM, that otherwise would have been difficult with a startup.

The structure of the ecosystem also encourages innovation, as demonstrated by the ability of individual developers and firms to create projects independently and then bring these projects into the main fold of the Eclipse Ecosystem. A prime example of this is the work done in Eclipse plug-in central (EPIC), a repository of add ons for developers using the Eclipse IDE. This project was originally developed by a coalition of a few small firms as a way to market their own products and allow potential customers to learn about their product. As EPIC matured and became more popular it was brought into the main Eclipse infrastructure as a critical component. The original developers believed that their work still provided them a competitive advantage.

However, the innovation in creating new projects for the ecosystem pales in comparison to the innovation in taking portions of the platform and utilizing the technology in new ways. Eclipse is one of the most complex Java programs available and many of the core components of the system are not standard components of the Java programming language. Many of the components and technologies that were originally created for Eclipse have been extracted into independent projects allowing the technologies to be brought to new markets and new companies to form around them.

A primary example highlighted by interviewees was the Eclipse Rich Client Platform (RCP). One interviewee in the mobile device field expressed excitement about his firm's future involvement with the Eclipse Foundation. He believed that the previous generation of mobile phones were constrained by relatively primitive user interfaces and was excited that his firm had chosen to use RCP as a basis because it provided for a rich environment that was proven, had a sizable number of existing developers, and cost them very little to develop. He cited the role of the Foundation in promoting the development of RCP as a viable platform as major reason for adopting the technology.

> When we first starting talking about using RCP people were really hesitant... The fact that the foundation is solid and they were promoting it really helped us sell the technology internally.
>
> –Product Manager at Mobile Device Firm

This change from a community based on the Eclipse IDE to a complete ecosystem based on a family of technologies did not happen overnight, nor did it happen solely as the result of the actions of a single firm. Indeed, if Eclipse had remained as a consortium with

IBM in the controlling role, these advancements may have never happened. The actions of the Eclipse Foundation have created a sort of innovation toolkit for the Eclipse Ecosystem. This strategy of creating a standard set of components that can be easily reused has been successful in fields as diverse as packaged food preparation to sports equipment and has previously been cited as a contributing factor for why the Apache Software Foundation has found such success[31, 123].

2.7 Conclusion

There is no doubt that the Eclipse ecosystem has been an incredible success. Prior to the release of Eclipse the market for Java IDEs was fragmented, while today it has solidified behind Eclipse[5][37, 42]. The openness and success of Eclipse has led other non-Java focused firms, such as Adobe, to utilize Eclipse as the framework for their proprietary development environments. However, even in that context, there is contribution back to the community, as evidenced by the recent donation of translations by Adobe to the Eclipse project.

Indeed, the Eclipse Foundation has succeeded in creating a robust ecosystem and driving significant value to the member firms. Through the skillful creation of a governance hierarchy, application of consistent values across the ecosystem, and actions undertaken by the foundation specifically to drive value, Eclipse has managed the delicate balance between an open core of a project and allowing proprietary firms to survive and thrive.

[5]There is one other major player in the Java IDE market, NetBeans from Sun MicroSystems. NetBeans has gained market share in the last two years, but has yet to garner broad corporate support outside Sun.

2.8 Topics for Future Research

One of the major innovations of the Eclipse foundation has been to seek out individuals and companies in areas not traditionally involved with Open Source. For example, Open Source was initially most successful with small startup firms where it's cost effectiveness and the gumption of employees made it a viable option. While Open Source has gained acceptance across most enterprises, there are many fields that merely use Open Source rather than contribute back to it, for example, banking and finance. The Eclipse Foundation has been very proactive about getting these firms involved in the Eclipse Ecosystem, both in the United States and Europe. This broadening of the ecosystem to include firms not traditionally involved in Open Source no doubt will place additional requirements on the Foundation. Although not all interviewees were asked, those interviewees who were asked about the broadening of the Eclipse ecosystem to "non-traditional" fields were almost universally supportive of this change. As the community expands it will be interesting to see if this view continues.

In addition, the Foundation has been very successful at broadening the ecosystem beyond just the IDE. According to employees of the foundation, one of the major challenges they are facing is conveying that the Eclipse IDE is for more than just Java, and that the Eclipse ecosystem is more than just the IDE. In the future it will be interesting to examine whether early member companies of the Eclipse Foundation, who set their level of membership based on the focus around the IDE, perceive that their influence is waning as the

ecosystem expands to fields such as server and mobile application frameworks.

Chapter 3

Firms and Firms: Business Collaboration Through Open Source Projects

Open Source software communities have typically been described as single developers working alone[59], or a loose collaboration between numerous volunteer developers working with little commercial motivation[61, 75]. From a commercial perspective, many of the early business models related to Linux and Open Source did little more than package the software, provide some degree of support, and add predictability to the release cycle of the software[134]. This is in stark contrast to the large scale commercial involvement found today in projects like Eclipse.

The Eclipse community itself provides some of the functionality that was once reserved for external firms, such as Linux distributors. One of the greatest hallmarks of the success of the Eclipse ecosystem is its ability to release high quality code with substantial improvements on a regular and predictable schedule. This annual effort sees hundreds of developers and dozens of corporations come together to release a yearly update to Eclipse. In the 2008 "release train" 33 projects all simultaneously released new versions of their software[126]. While there are other communities that perform time based releases, such as GNOME and Ubuntu, large amounts of their code are taken from other projects and integrated[10, 91]. Eclipse is different because new versions of the core software are released as a unified and tested package on the same day – an act that would be similar to Microsoft updating all of its developer tools on the same day. This successful release of software is greatly assisted by the fact that most developers are paid full time to work on Eclipse and there are very few volunteers within the community. In a 2006 interview with a member of the Eclipse foundation staff, it was estimated that there was about 800 individuals with commit access, of whom no more than "a handful" were not employed by a company and being compensated for their work in Eclipse.

In contrast to traditional Open Source models which describe open source participants as "user-developers"[54, 100] – highly skilled developers who work on the source code for a project they also have a need for, much of the code in many large Open Source projects is generated by paid professionals. For example, many of the features of the Linux kernel, such as support for IBM S390 mainframes, have no appeal to hobbyists and there is little

chance that the developers are the end users of the technology. Within Eclipse, there are certainly projects in which user-developers are present, such as the Mylyn project[55], a collection of tools to build a task focused workspace on top of Eclipse, and the Bioclipse platform for bioinformatics[102]. However most projects have a commercial focus and are driven by commercial developers being paid to create the code. Even the infrastructure around Eclipse is better designed for corporations, and in the words of one community member, "a monolith targeted at companies[118]."

Much of this corporate focus is due to the origins of Eclipse and the community that makes up the Eclipse ecosystem. Prior to the creation of Eclipse, IBM had a substantial number of partner companies developing technologies to enhance their VisualAge suite of developer tools. As described by an executive at a small firm that had been long term IBM partners, in the VisualAge ecosystem, all communication was mediated through IBM. A developer that wished to create an add-on tool for VisualAge needed to utilize a small number of interfaces, which were documented with varying degrees of care, and had little hope of extending the interface if additional functionality was needed. Interactions between companies in the ecosystem were rare, as there were licensing agreements in place for some firms that restricted their ability to collaborate.

This development style, where developers needed to conform to a fixed API from IBM, is problematic because it forced IBM to anticipate any API calls that add-on applications might one day make. Beyond being fixed on an API, long term IBM partners indicated that these opaque APIs also would have unintended interactions when documentation was lack-

ing – for example, a method may modify a data structure in a way which is not described in the documentation.

Furthermore, there was the constant worry that IBM, as the driver of the VisualAge ecosystem would choose to implement a feature that was remarkably similar to the products offered by smaller firms. This was described as a dance between mice and an elephant because of the great uncertainty it induced.

When IBM began work on Eclipse, its intentions were not to rectify these issues in the VisualAge ecosystem by using Open Source. Indeed, as the origins of Eclipse go back to the mid-1990's such an idea would have been far too radical for the state of the market at the time[1]. Rather the intention was to utilize some lessons learned through the development of Smalltalk programs and implement them in a new IDE for Java. The result was the original version of Eclipse which was novel because everything was designed a plugin, a small piece of code that linked to the other pieces of code at runtime through a set of API function calls[16, 24]. These architectural decisions also eliminated the need for a privileged or private API that previously had been the norm for many tools; most notably this attracted significant attention in the Microsoft antitrust lawsuit in which Microsoft was eventually forced to publish documentation for nearly all of their APIs as part of the settlement[29, 97].

As the Eclipse code matured, and before the decision was made to release the project as

[1]The term "Open Source" wasn't coined until 1998, the same year that saw the watershed release of Netscape's Mozilla source code as Open Source[94].

Open Source, IBM slowly began to show the project to members of its existing developer network. One interviewee who was among the first to see Eclipse outside of IBM said they were initially very excited about the project, and the modular structure of the code, but didn't see the project or community as significantly different from their existing ecosystem with VisualAge tools. However, when IBM announced that the code was going to be given away, a great amount of uncertainty was introduced for his firm.

In addition to launching Eclipse as an Open Source project, IBM did something no other project had previously done; they created a community that expressly focused on corporate participation. Individuals still had roles, and needed to be elected in a meritocratic environment to be approved as committers, but it was clear that rather than individuals guiding development, corporations in the community would learn to cooperate and drive development. In contrast to the large communities around the Linux Kernel, Apache Software Foundation, and Mozilla, for the first time, rather than a community of individuals, some of which were employed by firms, working on an Open Source project, there was now a community of firms which employed individuals working on an Open Source project. This shift in behavior and the different focus of commercial participants necessitates a new way of looking at the community.

While there has been previous research that examines the social networks of independent developers in Open Source[19, 20, 51], and additional work that has examined the case for Open Source business models and participation[2, 28, 60, 124], there has been little work on the actual ways in which corporations involved in Open Source collaborate

in a modern Open Source project. This chapter presents an empirical study of how firms interact in an Open Community using the Eclipse ecosystem as the subject. I begin by presenting an analysis of the modularity of the Eclipse project to show the degree to which components in the ecosystem are coupled and the need for collaboration may be present. Next, I establish the breadth of interest that firms have in Eclipse and the breath of participants that each of the projects attracts. When combined with information about the project modularity, this allows analysis of community stability and power in the community. I then compare these data to another more "traditional" volunteer based Open Source project – the GNOME project. Finally, I compare and contrast these results to the known information about individual participation in Open Source.

3.1 Description of Data

Once again the primary unit of study is the Eclipse project, the successful Open Source ecosystem founded around support for software development tools. This research utilizes data from interviews in chapter 2 and builds on it with quantitative data analysis based on artifacts within the Eclipse ecosystem.

The primary artifacts generated by the Eclipse ecosystem is the source code, which is kept in a concurrent version systems (CVS) repository. A complete copy of the CVS version control system repository was obtained. This repository is a shared resource that all developers in the ecosystem contribute to and it contains all of the official code for the

ecosystem. Each time a developer makes a change to code that they wish to distribute, they publish it back to the version control system repository where the changes between different versions of the same file are saved. In this way, any change can be "rolled back" and multiple branches and configurations of the software can be easily created[30].

Projects in Eclipse are typically not "born" into the official Eclipse CVS repositories, rather most projects begin life in external repositories that are later migrated into Eclipse once the project has reached a sufficient level of development maturity and the project has been officially accepted into the Eclipse community. Typically these projects have their code imported in such a way that the complete history of the project is maintained.

Within the data set there were 11 top level projects and 89 sub-projects in the community. Top level projects in Eclipse correspond to broad areas of interest, such as database interaction or integrated development environments. Each top level project has its own project management council that oversees development and ensures that the sub-projects are proceeding and evolving in a manner consistent with the Eclipse ecosystem[114].

In addition to the CVS archive, data were obtained from the intellectual property management system, IPZilla, that records the provenance of the code and also provides some background identity information for many of the developers in the community. Using this information developers were matched to corporations within the Eclipse ecosystem, and this resulted in the ability to tie a corporation to the pieces of code they contributed to the Eclipse ecosystem. This process of managing intellectual property has become more rigorous over time, so there is some noise in the data from early periods of the Eclipse

Consortium. Many of the early developers are identified as working for "individual" or "unknown". As time progresses these instances of unknown affiliation decrease significantly.

This chapter also performs a comparison with another Open Source community, the GNOME project, a loose collaboration of volunteers which seeks to create a complete desktop environment for Linux and Unix-like operating systems[38]. In contrast to the commercially focused Eclipse Foundation this project has an individual and volunteer oriented ecosystem – corporations are affiliated with GNOME only to the degree they employ individuals working on GNOME or they wish to be members of the GNOME foundation advisory board.

For this portion of the research a complete copy of the version control system archive for the GNOME project was obtained. Developers were then matched to the corporations that employed them to obtain information about the extent of corporate involvement in the GNOME community. More information about the collection of these data is in chapter 4.

3.2 The Architecture of Eclipse

One of the key attributes of the Eclipse source code that allows the project to be divided between firms is the modular nature of the project source code. Modern object oriented programming languages, such as Java, allow collections of files and objects to be grouped together into packages. These packages then can choose what methods to expose outside

the package, thereby allowing a degree of abstraction between methods calling the package. Package designers and maintainers then need only ensure that the these interfaces, typically called APIs, remain relatively stable, while being free to change the underlying source code and implementation.

This concept, which is essentially another form of information hiding, is heavily enforced in the Eclipse project. Furthermore, the community has strict controls on which packages are allowed to be dependencies. This is done to prevent the creation of circular dependencies and also to ensure that the code remains clean and maintainable. Using tools such as Lattix that evaluate call graphs in software packages it is possible to build a dependency network between packages in the Eclipse ecosystem[63]. For many of the largest and most prominent projects in the Eclipse ecosystem, these dependencies can be seen in figure 3.1[2].

These results show that there are very few dependencies across most combinations of modules within the ecosystem. One notable exception is the project called `eclipse`[3] which sees dependencies from almost every other major project in the ecosystem. There are two primary reasons for this. First, the `eclipse` project contains the equinox sub-project – which forms the core of much of the object model for Eclipse. Secondly, and more importantly, it contains the platform sub-project, which in itself operates much like a top level project with numerous sub-projects. The platform sub-project contains the code to

[2]I wish to thank Smita Ramkete for her work in running Lattix on the Eclipse ecosystem and generating this data.

[3]To distinguish between the Eclipse project as a whole, and the top level project within Eclipse called `eclipse`, the latter will be all lowercase and typeset using a fixed width font.

	Tools	STP	BIRT	Eclipse	DSDP	Modeling	Technology	Modeling	DataTools	WebTools
Tools		204	27	45		88	502	381	53	163
STP										
BIRT						85				
Eclipse	365	**6642**	239		**4015**	**5577**	**3702**	**2291**	**6979**	184
DSDP	**3529**	213	361	1754		775	**6268**	462	39	1789
TPTP										
Technology	1888	74	456	171	7	46		57	19	298
Modeling		**3209**	18		57	244	**9421**		1136	**2960**
DataTools										
WebTools	41	15	21			11	85			

Figure 3.1: Dependencies between major components of the Eclipse ecosystem as measured using Lattix. Calling modules are across the top, called modules are along the side. Cells in red and bold indicated instances of more than 2000 calls from the calling module to the called module.

a number of key components of the Eclipse ecosystem, including the widget toolkit, SWT, and the framework for updating components in Eclipse[113]. As a result, nearly every other component of Eclipse that displays information to the user or is able to update itself is tightly linked to the platform sub-project. This is shown in the dependency network, as all of the projects have dependencies on `eclipse` and six of the nine other projects have very strong dependencies on the the `eclipse` project.

Interestingly, there are some modules in the ecosystem that are called by none or very few modules. A good examples of this is the BIRT project, a tool for generating business intelligence reports. It was originally donated to the Eclipse Foundation by Actuate, and allows almost anyone to create high quality reports in a number of formats with very little effort. It is often used to generate reports that are displayed on the screen to developers[111].

However, very few other projects depend on BIRT because it is often seen as a component that developers use in creating applications based on the Eclipse framework, rather than tool that comprises a major portion of the Eclipse IDE. However, that is beginning to change as the modeling project has begun to use BIRT to generate reports for software developers using the Eclipse IDE. Another example of a project which is not a dependency of any other project is the Data Tools project. This project exists to provide access to databases for end users, typically as an assistive component to software developers. Although it is a higher profile project within Eclipse, most of the other projects have not yet been able to harness the data access methods that the Data Tools project provides.

The lack of calls between most modules indicates that most components in the Eclipse ecosystem are relatively independent. For example, a developer working on the Data Tools project requires knowledge about the core Eclipse platform, an attribute common when building within any large scale platform, but only needs very little knowledge about other components in the ecosystem. Furthermore, as no projects are dependent on the Data Tools project, this allows the developers to freely innovate without the need to maintain a legacy API for dependent projects.

3.3 Distribution of work

The first step in understanding how firms interact within the Eclipse ecosystem was to evaluate how individual firms participate in Eclipse. Using the data from the CVS archives

and IPZilla it was possible to identify which firms had made modifications to projects in the Eclipse ecosystem by matching up commits to the repository with developers and the firms they were employed by at the time of the commit. This allowed the generation of figure 3.2 and figure 3.3, which show the number of top level projects and sub-projects each firm has made contributions to.

Both of these figures show that despite the fact that the Eclipse Foundation is the central entity of the Eclipse ecosystem, IBM still dominates involvement. They currently have code in nine different top level projects, and 57 of the 89 sub-projects in the ecosystem. Equally telling, however, is the low levels of involvement from many other firms in the ecosystem. After removing the Eclipse Foundation from consideration, which primary does non-coding work on project source code, such as updating license and formatting repositories, no firm is involved in even half the number of top level projects as IBM or even 15% of the sub-projects of IBM. More than half of the firms are involved in two or fewer sub-projects, providing a testament to the degree that Eclipse ecosystem is structured in such a way that firms can focus primarily on areas of their expertise.

However, a narrow focus on a handful of projects, does not mean that firms never need to collaborate with other firms. To understand the degree to which firms collaborated, a social network of the firms was built covering the entire history of Eclipse. Two firms were linked in this network if they had both contributed code to the same project at any point in history. Using this method, every project was a clique of the firms that had contributed code. Thus, this represents a maximal degree of collaboration between firms using CVS

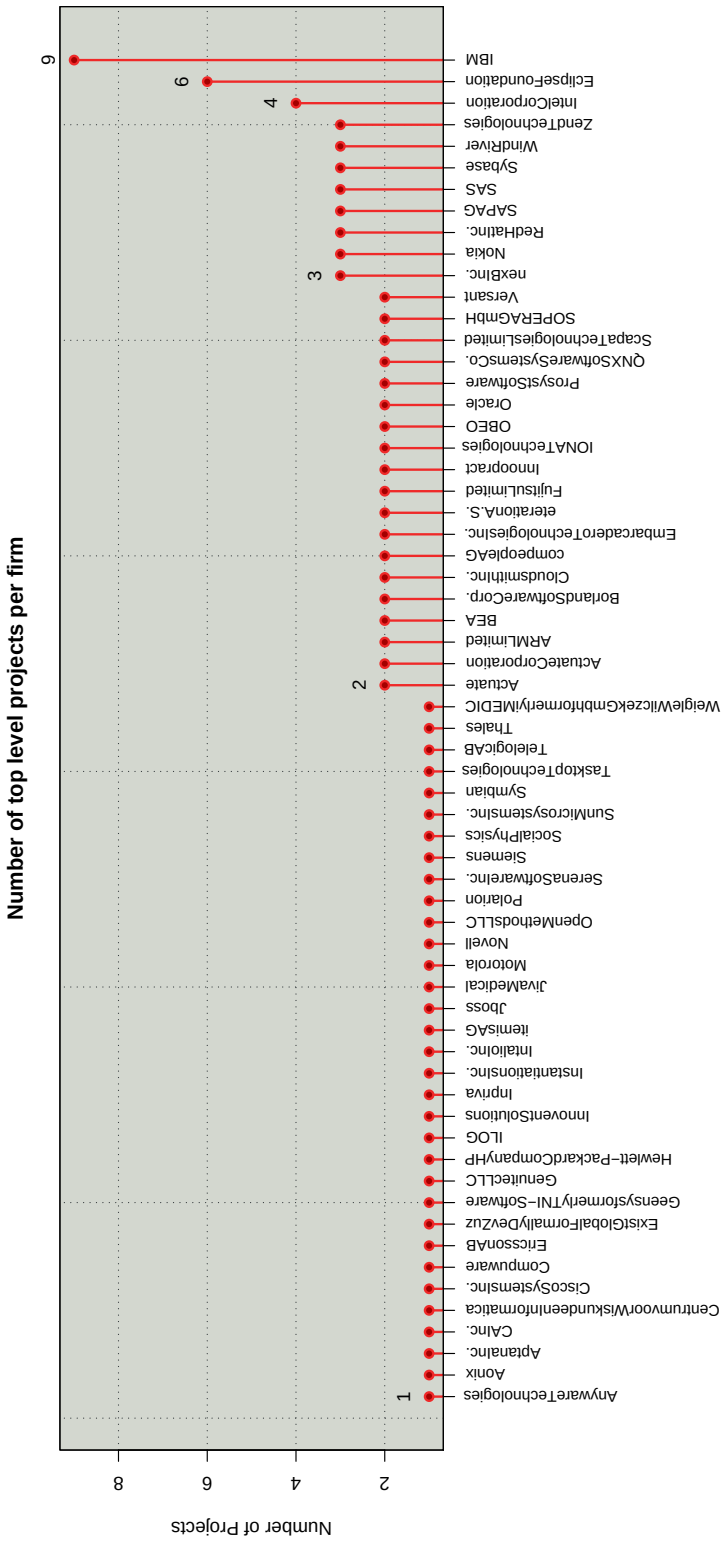

Figure 3.2: Number of top level projects each firm participates on in the Eclipse Ecosystem

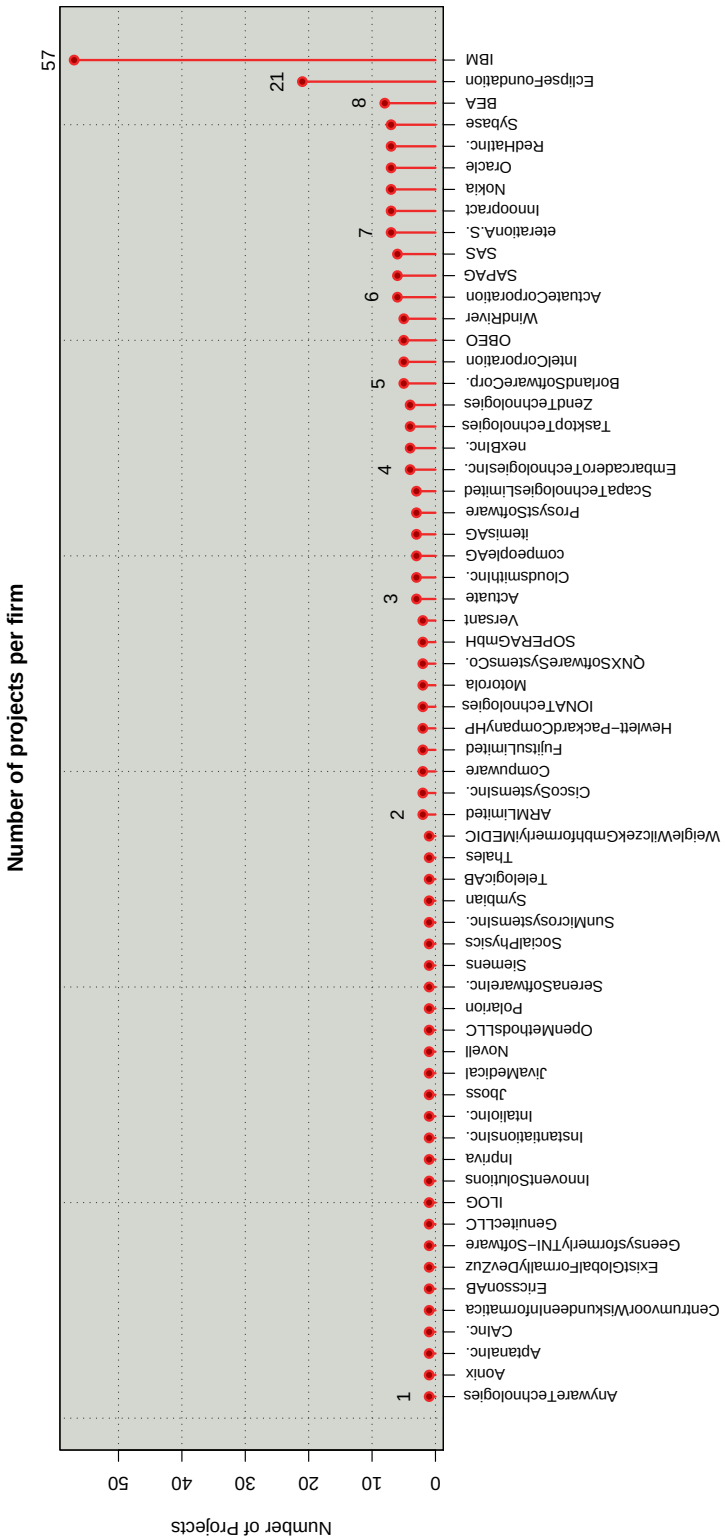

Figure 3.3: Number of sub-projects each firms participates on in the Eclipse Ecosystem

as the primary coordination medium. This network was generated at the top level and sub-project level, and the degree of each firm can be seen in figure 3.4 for top level project and for sub-projects in figure 3.5.

The top level projects is where much of the roadmapping and management of the Eclipse ecosystem takes place. The PMC for each top level project ensures that firms are following the rules governing software development. If two firms were co-present at the same time working in the same top level project there is a reasonable chance that some sort of collaboration was needed (although, it should be noted that figure 3.4 and figure 3.5 do not take into account temporal relationships). Although IBM is still an outlier in the dataset, having collaborated with 58 other firms at the top-level project, this general distribution is much more even, following a near-perfect linear distribution.

Of special note in figure 3.4 is SocialPhysics, which has no collaboration with other firms on top level projects. This is largely because of the nature of the project that Social-Physics works on, a framework called Higgins that seeks to provide a common interface to various sorts of identity management tools both over networks and in physical spaces[115]. When the project was proposed to the Eclipse Foundation, it was unique as it was more of a library than a tool, which is the prior focus of the Eclipse Foundation. At numerous events the Eclipse Foundation has heralded Higgins as a successful attempt to broaden the community beyond the traditional IDE market. The website for Higgins boasts the involvement of numerous tech giants including IBM, CA, and Google. While these firms are active in developing tools that work with Higgins, at the time of data collection, they had not

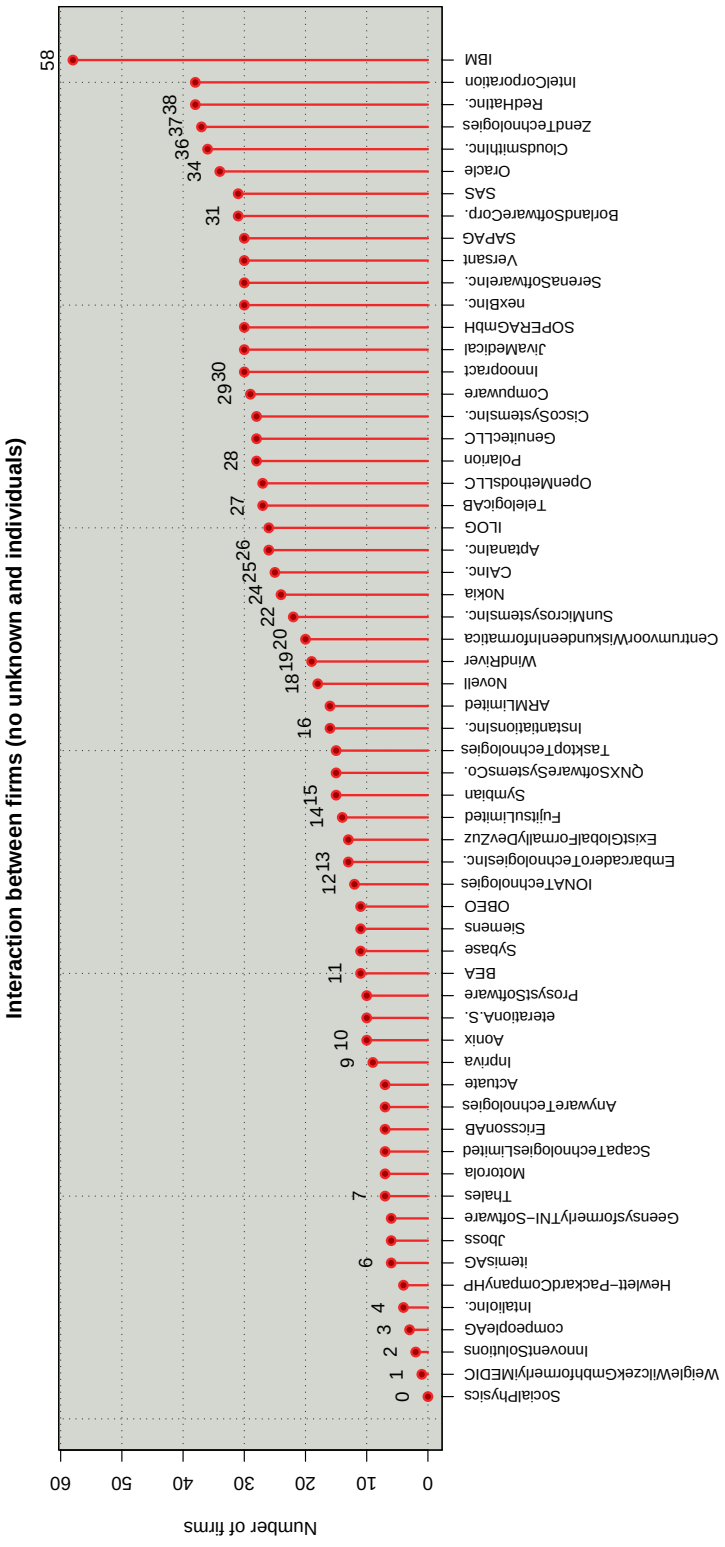

Figure 3.4: Top level project shared participation in Eclipse. A pair of firms are considered to have shared participation if both firms committed code to a subproject under the top level project. For example, IBM is active on top level projects that have contributions from 58 other firms.

yet made contributions to the Higgins source code. While it does appear that Higgins is successful in developing a new product and business model within the Eclipse Ecosystem, there is little evidence of collaboration between firms that has been the hallmark of much of Eclipse's development.

The sub-project level, which provides a more nuanced view of technical collaboration is shown in figure 3.5. There is significantly less collaboration between firms at this level. This is to be expected as there are 89 sub-projects compared to 11 top level projects. More than half of the firms in the community contribute to projects that have three or fewer other firms contributing and eight firms work on no sub-projects that have participation by other firms. From a technical perspective, this allows those firms nearly complete freedom to implement their projects in a way of their choice. It may also prove a temptation for those firms to utilize communication and development processes that are more suited for proprietary development than the more expensive and time consuming Open Source process.

From the perspective of building an ecosystem, these results are both worrisome and encouraging. The degree to which firms are able to operate independently is worrisome and it changes the picture of the community from a group of firms working together toward a shared set of goals to a collection of firms working independently that, within a defined set of constraints, may each seek to maximize their own benefit to the detriment of other portions of the project. However, it also works well for building an ecosystem because firms can clearly succeed and need to master only a small niche of the greater ecosystem in order to extract benefits from a substantial portion of the community.

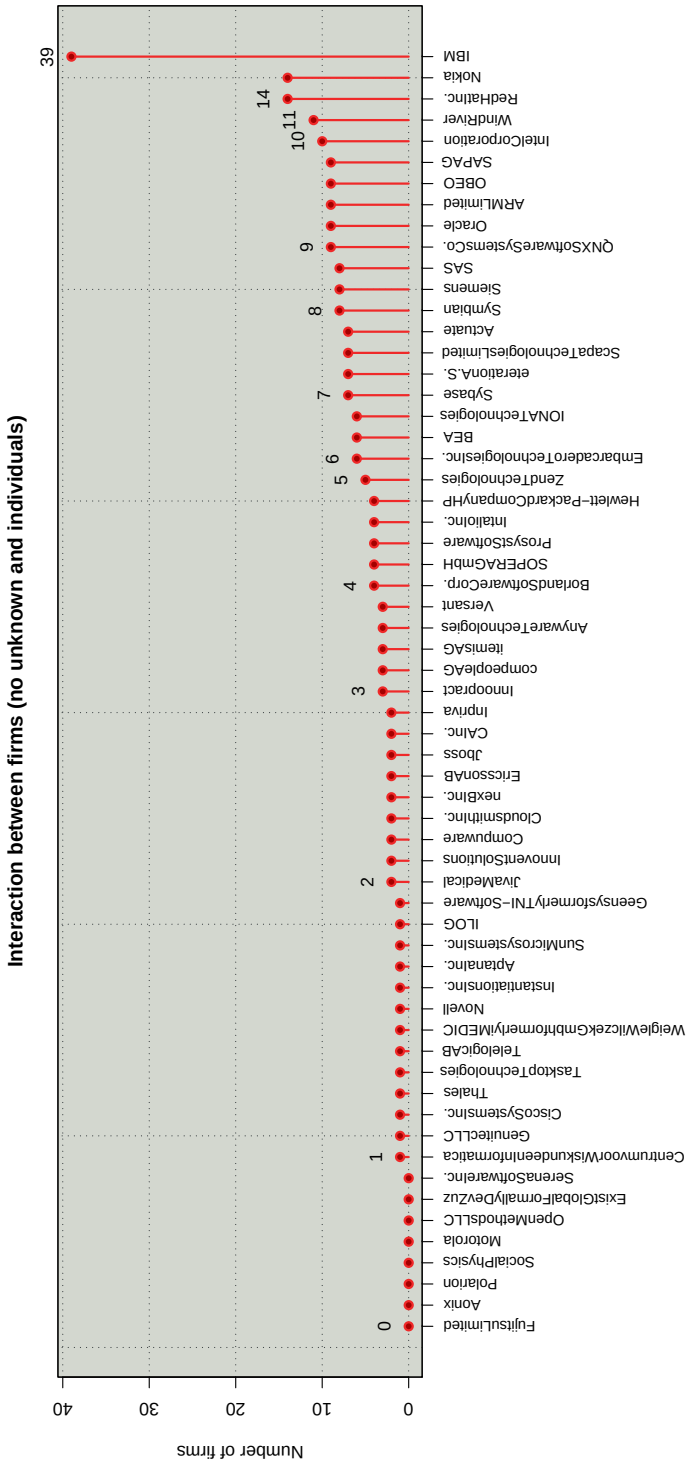

Figure 3.5: Sub-project level shared participation in Eclipse. Two firms are considered to have participated in the same sub-project if they both committed code to that sub project. For example, the sub-projects that IBM has committed code to have contributions from 39 other commercial firms.

3.3.1 Firm Participation on Projects

Projects in the Eclipse ecosystem are encouraged to have contributions from developers employed by multiple firms. This is believed to help guard against a single firm leaving the ecosystem and causing a variety of possibly critical projects from faltering. Using the same data it was possible to generate the number of firms working on each of the top level projects and sub-projects, in essence while the previous section showed the breadth of interest of firms in the community, this section shows the breadth of appeal of projects to the community. The results can be seen for top level projects in figure 3.6 and for sub-projects in figure 3.7.

At the sub-project level a handful of projects show that they have no commercial participation, this result may be artifact of the data as commits by developers who were classified as "unknown" for their corporate involvement, typically individuals active in the early days of Eclipse, were not allocated to a commercial firm. Therefore, while these projects may have commercial interest, it is not possible to ascertain to what degree they appeal to a commercial market.

The sub-project which has gathered the most widespread interest is the tools.CDT project, which is commonly called CDT in the community. The CDT is an effort to create a development environment for the C programming language on top of the Eclipse framework. It is one of the oldest sub-projects in Eclipse dating back to the days of the Eclipse Consortium and continues to be developed today. It has also found widespread acceptance,

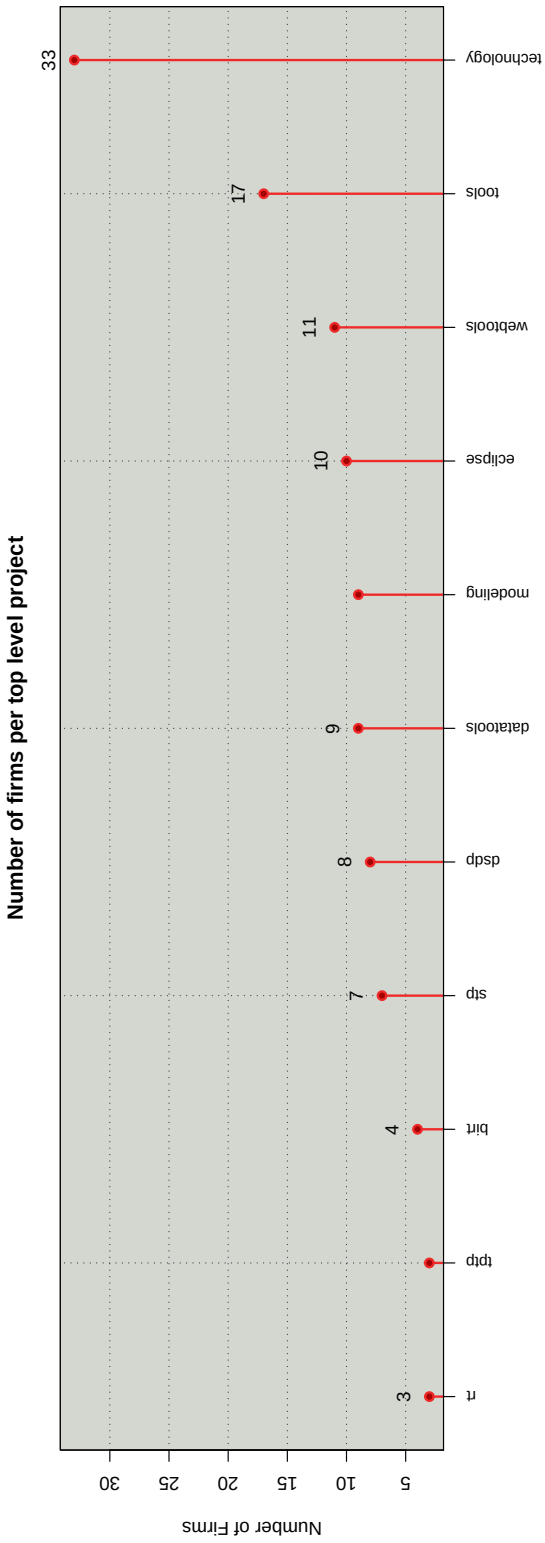

Figure 3.6: Number of firms contributing code to each top level project in the Eclipse ecosystem. This is the total number of distinct firms that have contributed code to the sub-projects belonging to that project. Participation is shown for all time, thus older projects such as technology and tools exhibit much higher levels of participation than the newer runtime project (rt).

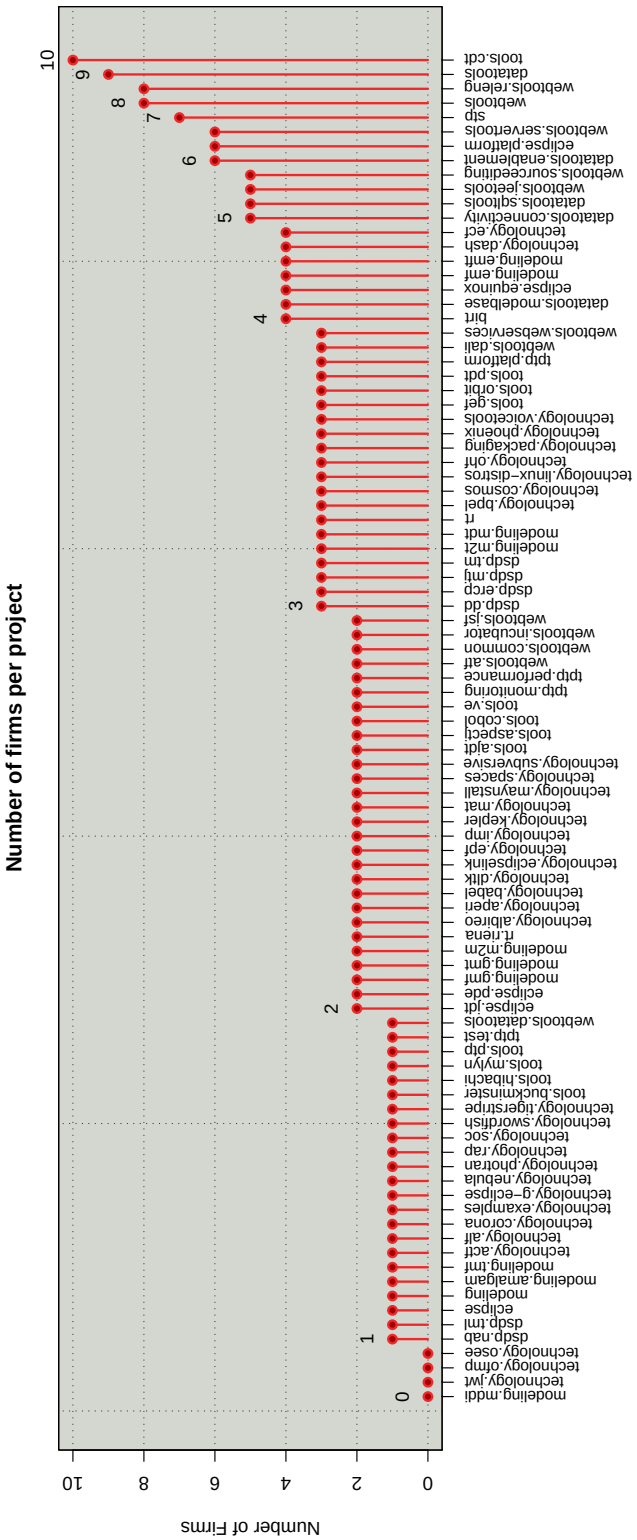

Figure 3.7: Number of firms contributing code to each sub-project in the Eclipse ecosystem. Contributions from the Eclipse Foundation (mainly housekeeping chores), developers with "unknown" affiliations, and "independent" developers have been removed, thus resulting in several projects with no apparent commercial contributions.

especially in the community of embedded systems developers. However, while many firms have been active on the project, the actual pattern of participation paints a very murky picture of the project history.

In figure 3.8 the history of the CDT is broken up into month long time periods. At each month the proportion of commits made by each firm is shown, along with a black line that indicates the volume of commits relative to the busiest time period on the project, in this case period 50. This very clearly shows a tumultuous history for the project, with multiple firms taking the lead on the project at different periods of time.

The genesis of the project was from IBM and group of developers who were categorized as "individual"[4] or working for IBM. Shortly into the life of the project QNX Software, the developer of the real-time operating system of the same name, took the reins of the project, eventually seeing IBM's contributions to the project drop away completely for a period of almost a year starting in period 16. This marks the first dramatic change in project leadership within CDT. It is also within this period that ARM Limited (now owned by Intel), the developers of a highly efficient microprocessor suitable for embedded environments became very involved in the project.

In period 20 Wind River, one of the dominant market players in the embedded systems market makes its first contributions to the project and in period 24 IBM again returns to the project. These firms represent the bulk of the activity for the next two years, with

[4]Within the data set, appearances of "individual" and "unknown" for commercial affiliation are much more common in the early time periods before the Eclipse Foundation had more rigid intellectual property procedures in place.

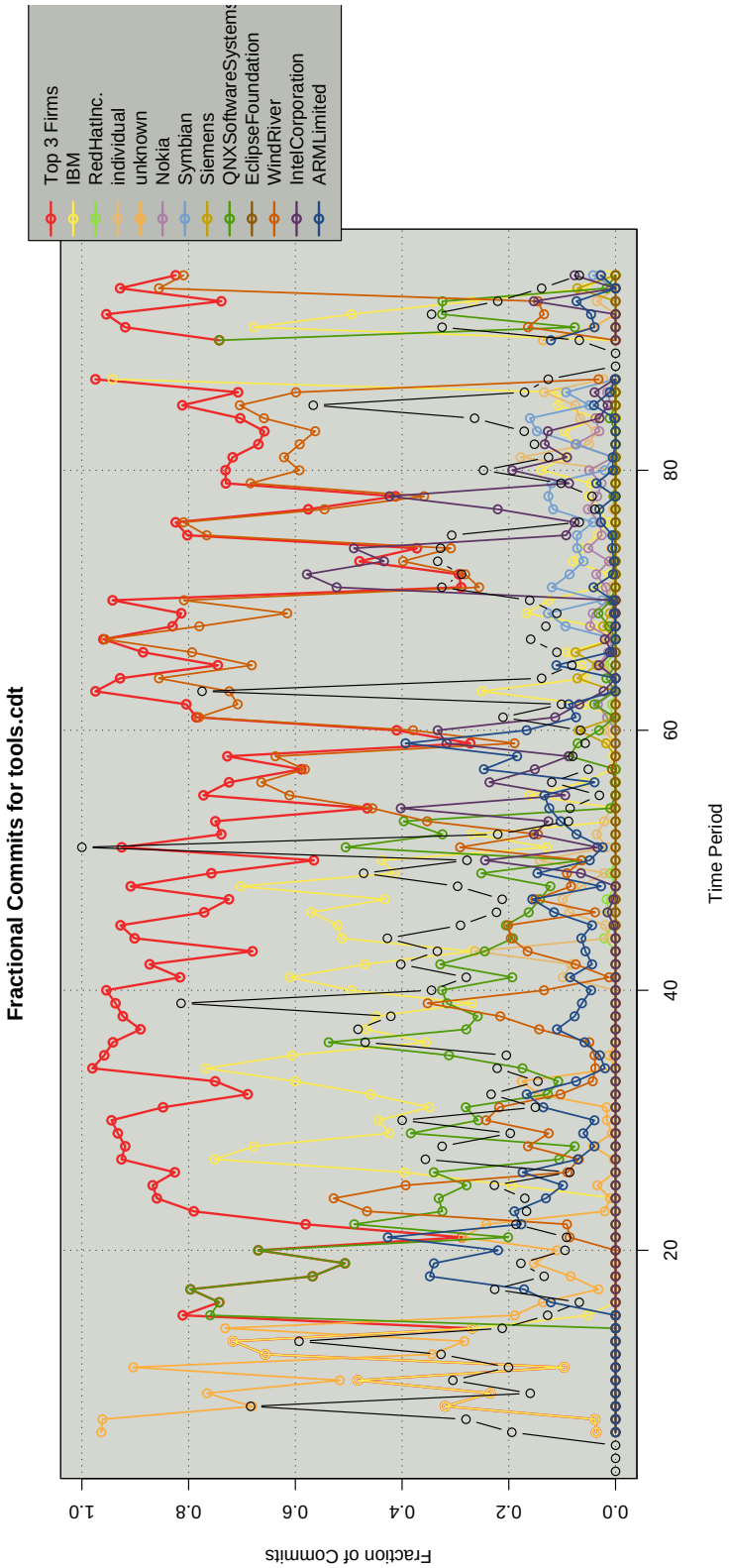

Figure 3.8: Fractional contributions to the `tools.cdt` project by firm by month. The red line at the top represents the sum contribution of the top three firms in the project (IBM, WindRiver and QNX Software Systems). Time is measured in months since the first available data, April 2004. The black line with the non-connected circles represents the volume of commits relative to the most active month in the project lifecycle, period 51. Near the end of the data, there is a two month span for which there was no data about firm level contributions.

IBM typically leading the way, followed by QNX, Wind River, and ARM limited. Time period 50 represents a remarkable change for the project, during this time QNX made a huge number of commits to the project, while IBM backed off slightly. According to one interviewee, this was the point where QNX had finished enough of the work on the CDT that they had a working project for their needs. After the major involvement by QNX most of the development on the project has been managed by Wind River who have successfully marketed the Wind River Workbench as the primary IDE for developing a wide variety of embedded systems. Another member of the CDT community indicated that these changes of leadership weren't always viewed as a success however. It was perceived that the handoffs of leadership in the community occurred frequently because one firm was changing focus or leaving the project, and without someone else stepping up the project would die. He described many of the transitions as "reluctant" on the part of the firm that took leadership.

This pattern of leadership change and multiple firms with substantial involvement is in contrast to the patterns seen around the Eclipse platform, as seen in figure 3.9 which shows contributions to the platform project in the `eclipse` top level project. Once again, noise in the data at the beginning of the Eclipse project yields a number of active developers with "unknown" affiliations at the genesis of the project. After this fact, however, the bulk of the code has been written by a single company, IBM.

From a long term ecosystem stability perspective, this is a challenge for the Eclipse community; if IBM ever chooses to change focus away from Eclipse, many of the core

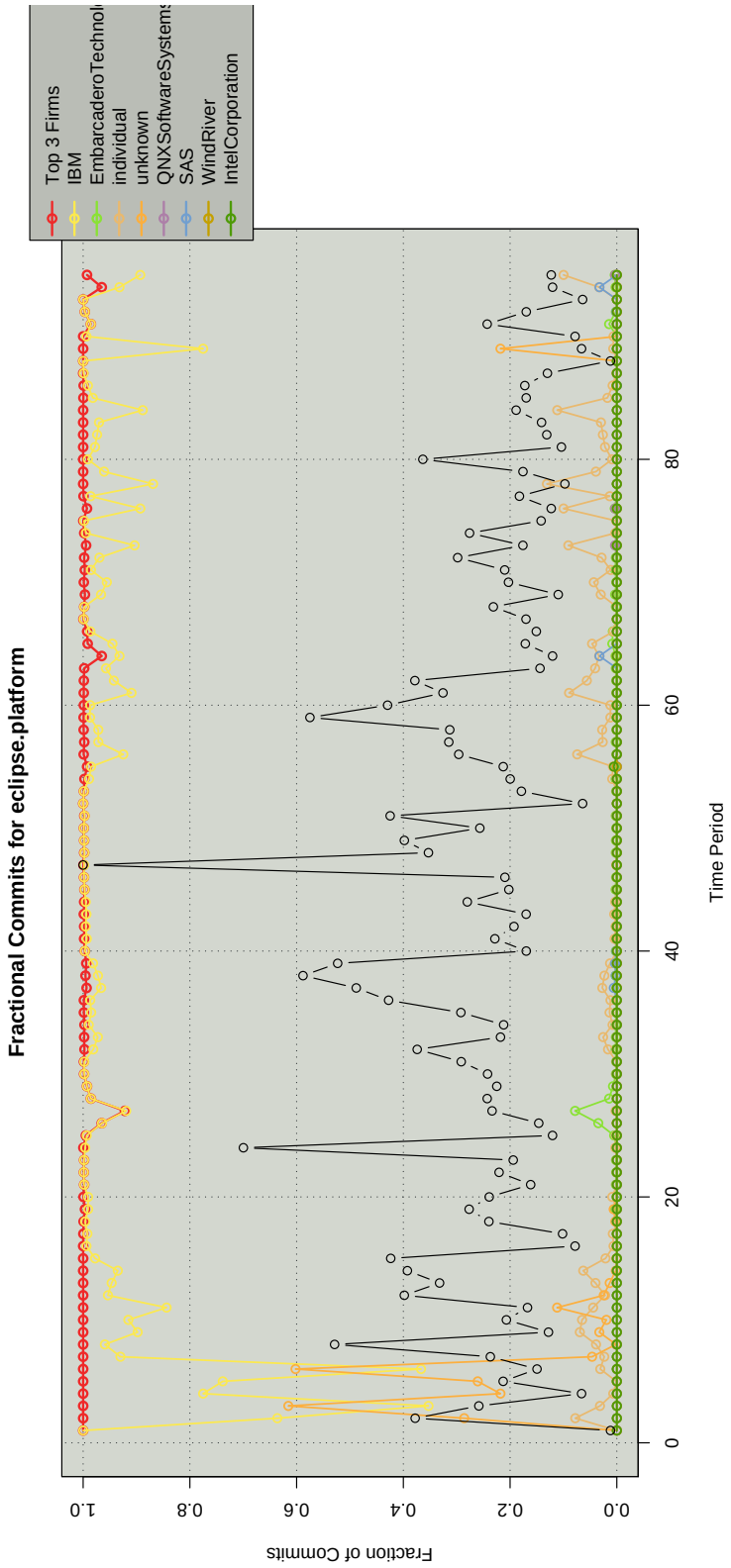

Figure 3.9: Fractional commits to the `eclipse.platform` project by firm by time. The red line at the top represents the sum contribution of the top three firm classifications in the project (IBM, individual, and unknown). The black line with the non-connected circles represents the volume of commits relative to the most active month in the project lifecycle, period 47.

components of the community, including the object model, graphical widgets, and plug-in management would be unmaintained. However, such a setup may be beneficial for IBM as they have a greater ability to direct the development of the project and create features they desire – which is exactly what IBM has done. Recent versions of many software packages from IBM, including Lotus Notes, Lotus Sametime, and Lotus Symphony are built using the framework provided by the platform project. IBM's great investment in the core technologies across a wide variety of projects provides a degree of certainty for community members. One interviewee described his firms use of SWT, the widget toolkit for Eclipse, and said "IBM takes care of SWT, they need to. We just build on it and make SWT a stronger market force."

Indeed, some of this dramatic difference in contribution levels may be due to the difficulty of monetizing standard components, such as graphical user interface widgets. Most popular widget toolkits, including the standard widget toolkits on Windows and Mac are free with the development environment. One of the most prominent commercial widget providers, TrollTech, who produce the QT multi-platform widget toolkit, was recently purchased by Nokia and announced in early 2009 that it was relicensing the entirety of the project under the terms of the LGPL license[89]. In effect, this made the QT toolkit non-commercial. This lack of marketability makes it difficult for a single firm to devote large resources to the project

3.4 Comparison of Eclipse with GNOME

To better understand the implications for the Eclipse community of their work distribution between firms, and assess if this distribution is common within Open Source communities, a comparison analysis was performed with the GNOME community. GNOME, which was founded in 1997, has achieved moderate success as a desktop environment for Open Source operating systems such as Linux. It is primarily volunteer driven, although a significant number of commercial firms, primarily Linux distributors, pay developers to contribute full time to the project. The entire CVS history of the GNOME project was collected until the point where the version control system was migrated to Subversion on January 1, 2007.

Employees of firms were identified through a combination of email address analysis (e.g. someone with an @redhat.com email address most likely worked for Red Hat), reading through mailing lists, and in some cases questioning individual community members directly about their professional involvement. In total there were 16 companies who made significant contributions to GNOME[5]. These companies employed 259 developers. There were commits from 832 developers for which they were either verified as volunteers or for which there was no sign of commercial employment.

The governance style of GNOME is very different from Eclipse, as each project is allowed to manage itself, decide on its own standards and issue releases as it sees fit. However, as the community uses time based releases, their are general periods when all

[5]One artifact of this data is that Helix Code was later renamed Ximian. Ximian was purchased by Novell near the end of the data set. In addition, SuSE was also purchased by Novell around the same time. The contributions are sorted out by firm name at the time the commits were made to CVS.

developers are rushing to complete code for the next release. The release cycle along with annual conferences provides for some degree of shared vision and planning for members of the community. Most telling about the community is that anyone with commit access can contribute to any module of the project source code. While there is a social norm discouraging developers from committing code to projects without first checking with project maintainers, this norm does not apply to translators who are able to take advantage of the open nature of the product source code repositories and quickly translate the software into a variety of different languages by lowering the amount of bureaucratic work necessary to create a translation.

As with Eclipse, the first step was to analyze the number of firms that each firm had shared interest in code. Once again, the standard used was that if the two firms both contributed code to the same module, then they would be connected. The degrees of each of the firms is shown in figure 3.10. Unlike Eclipse, GNOME has no major commercial benefactor, and therefore, there is no outlier like was found in Eclipse. Aside from that, however, both communities display a relatively similar pattern of commercial involvement.

The next step was to evaluate the number of commercial firms working on each project in the community. Unlike Eclipse, GNOME has no concept of top level projects, so it was only possible to evaluate co-participation as the individual project level. Another artifact of GNOME is that because almost anyone with CVS access can create a project, this leads to numerous projects that die out or get folded into other projects. In the Eclipse Ecosystem the Eclipse Foundation prunes such projects from the CVS tree, but this is not present in

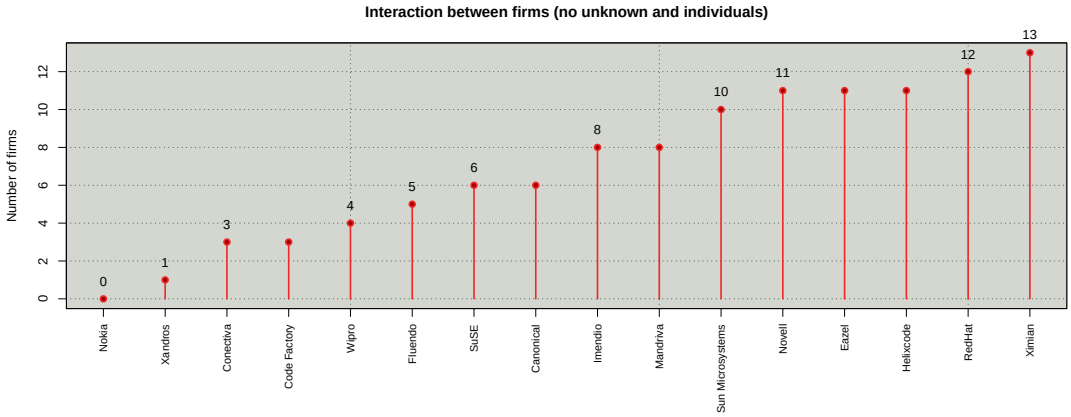

Figure 3.10: Number of firms with shared project contributions in the GNOME ecosystem. For example, for all projects across the ecosystem that Red Hat made contributions two, there were contributions from twelve other distinct firms. The distinction between HelixCode and Ximian, Ximian and Novell, and SuSE and Novell has been maintained in this data even though HelixCode later became Ximian and Ximian and SuSE were later purchased by Novell.

GNOME. The number of commercial firms with code in each of the projects in GNOME is shown in figure 3.11.

Once again, a similar pattern of involvement is seen as in Eclipse, with many projects garnering commercial interest from only a few commercial firms. The major difference, however can be seen in the GTK project. GTK is the standard widget toolkit that all end-user applications for GNOME are built upon. As the library has matured, it has began to include more utility code making it serve as the primary module for all projects in the community. In contrast to the Eclipse community, in which one firm, IBM, made all the commits to the platform, in GNOME, fifteen out of the sixteen firms have contributed code to GTK. The exception to this is Nokia, which did later contribute to GTK as it forms the basis for their Maemo platform and n7xx/n8xx line of internet tablets, although this was after the data for the project was collected.

As a final point of comparison between the two communities, a social network was generated that linked contributions by firms to projects within the communities over the course of a one month period. For Eclipse, as shown in figure 3.12 the month selected was May 2008, about a month before the Eclipse community ships its annual "release train". During this period there was large amount of bug fixes while simultaneously developers were planning out new features in new branches of the software. In particular, many of the ideas from EclipseCon 2008, which took place in March 2008, were beginning to see their initial experimental implementations. This is contrasted with figure 3.13, which shows a one month snapshot of the community around GNOME. Taken from May 2005, the GNOME commu-

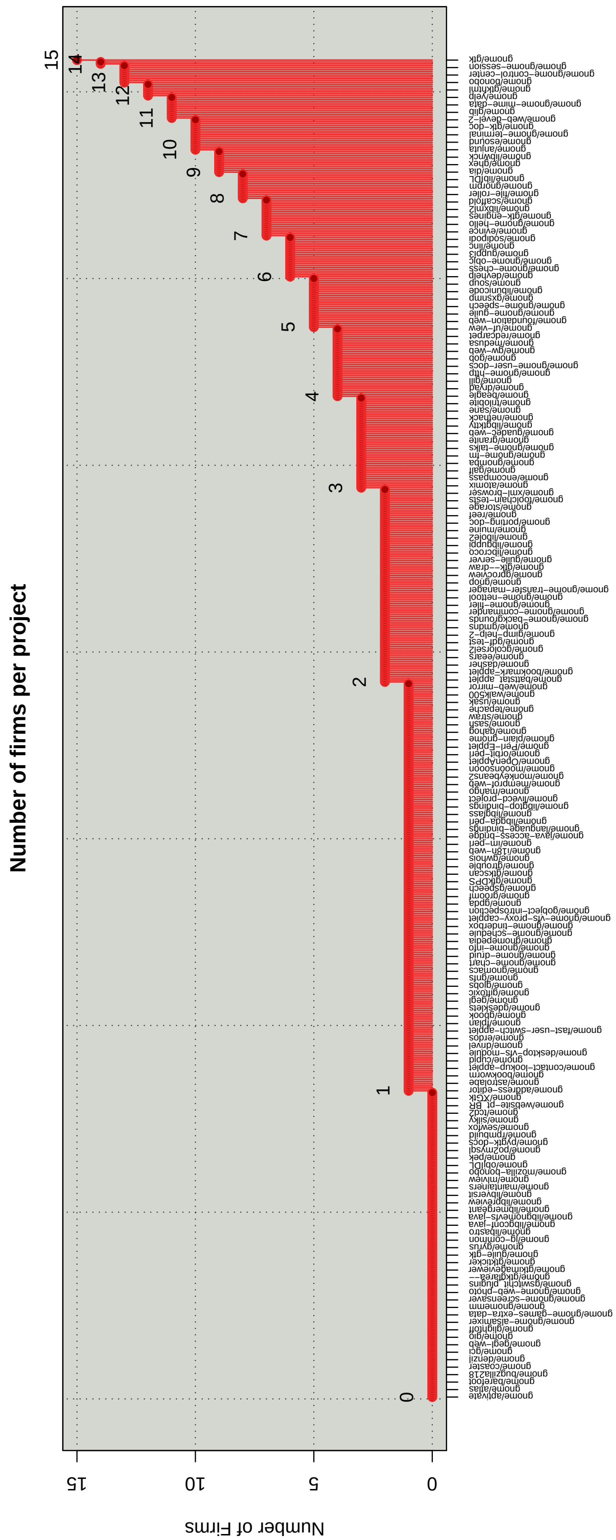

Figure 3.11: Number of firms contributing code to each project in GNOME. Due to space constraints only 25% of the project labels are shown in the figure. The project with the most commercial contributions is `gnome/gtk`, which forms the basis of the GNOME ecosystem and has contributions from fifteen different commercial firms.

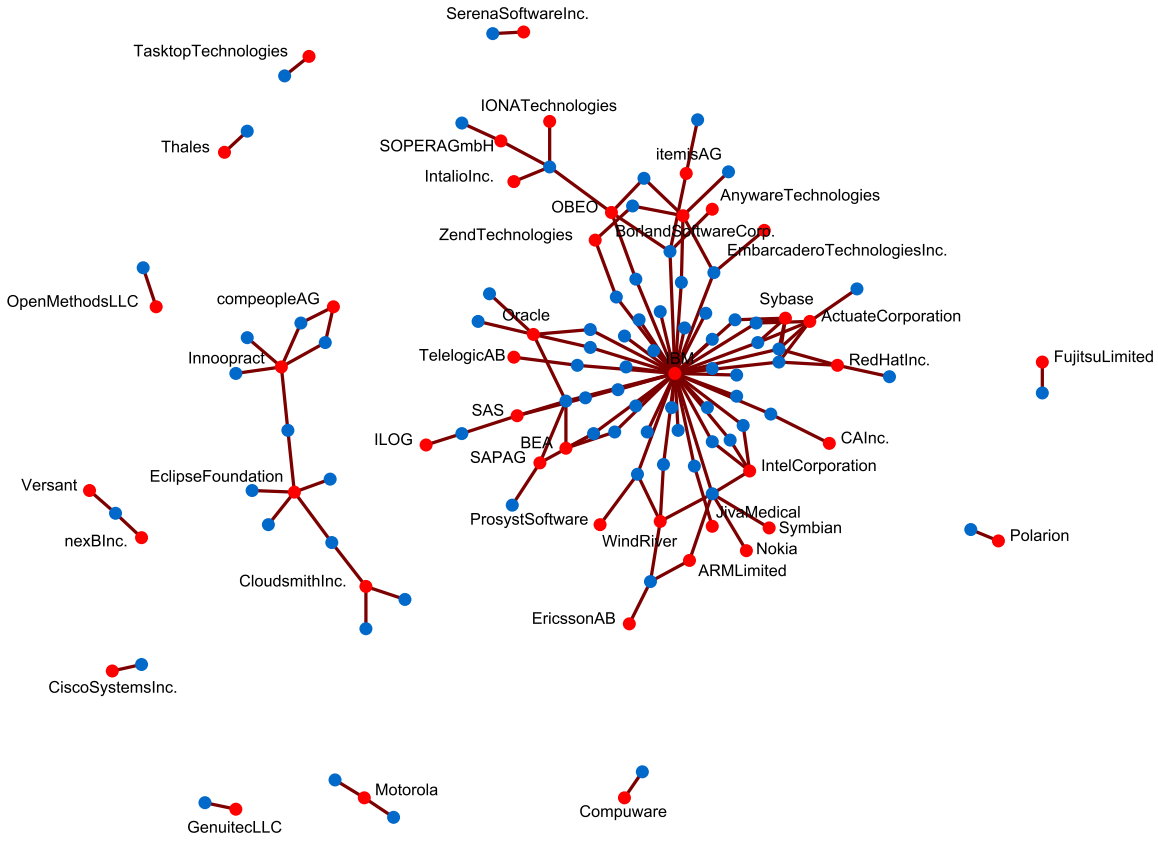

Figure 3.12: Participation by firms (red) in projects (blue) within the Eclipse ecosystem during May 2008.

nity was approximately the same age as the Eclipse community in the previous figure. This was two months after the most recent release; most developers were patching bugs in the software and implementing new features in anticipation of their upcoming annual conference, GUADEC. At this point Novell had already purchased Ximian and SuSE, however the data keeps developers with their original affiliations in the case of an acquisition.

Several things stand out between these two networks. The network around the Eclipse ecosystem is a disconnected network, while the network around GNOME is nearly fully connected, with the exception of Nokia. Many firms in Eclipse are active on only a handful

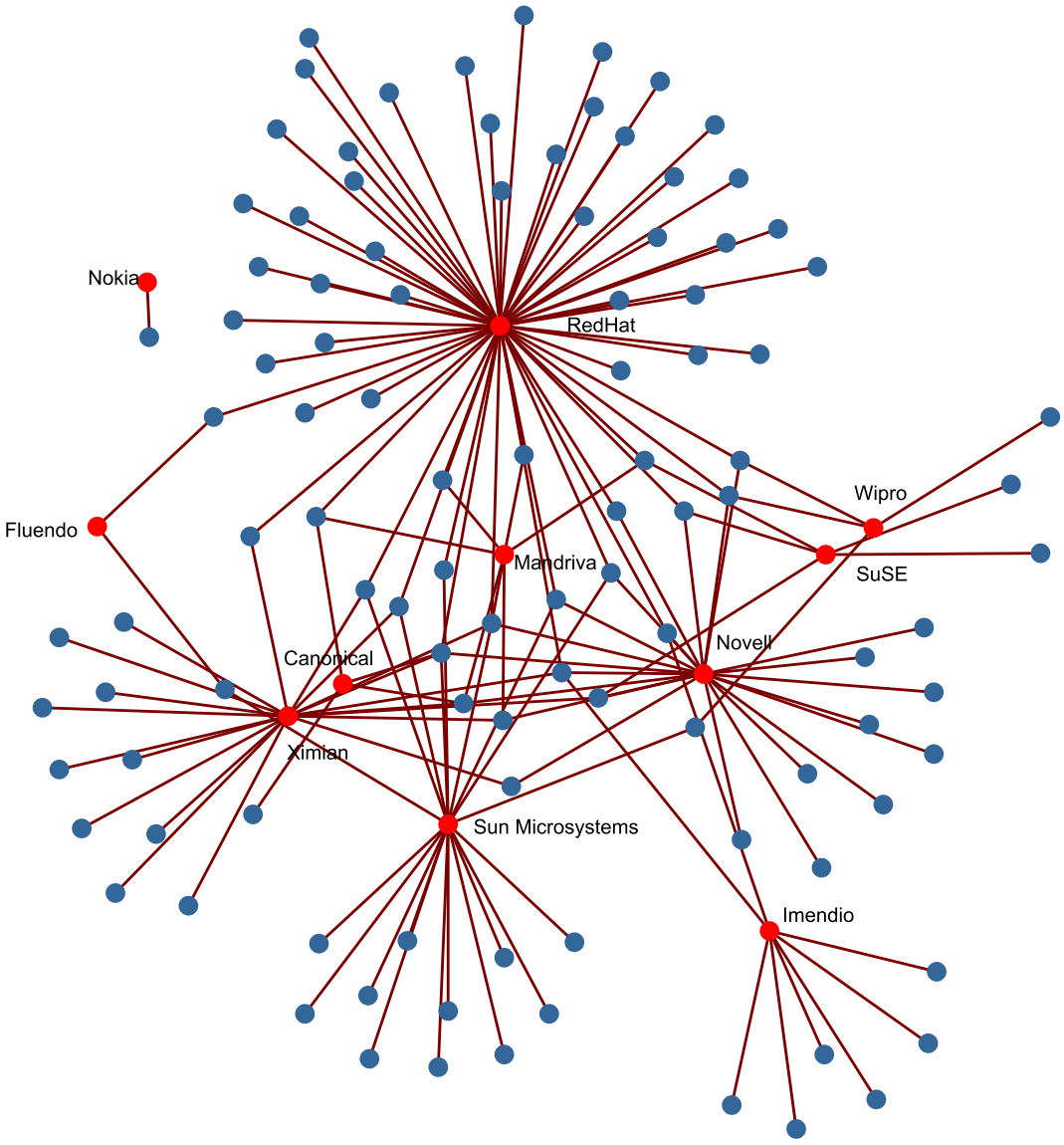

Figure 3.13: Participation by firms (red) in projects (blue) within the GNOME ecosystem during May 2005.

of projects, and these projects often occupy areas where the largest players, attached the giant component in the center of the graph, typically are not involved. Contrast this with the GNOME ecosystem, which sees many firms heavily involved in all aspects of the infrastructure, often sharing numerous projects between each other. Only a single firm was active on just one project during the period, Nokia.

Without a major benefactor, the firms involved in the GNOME ecosystem all work on key components of the infrastructure together, forcing a larger amount of collaboration between firms. In this snapshot, GTK+ and GLIB, the primary libraries for building GNOME applications each have five firms contributing to their code during the month. In the Eclipse ecosystem, however, during the month shown only IBM contributed code to the Eclipse platform, despite the fact that many firms rely on the platform for their own application development.

However, this is not necessarily a good or a bad aspect of the Eclipse ecosystem. The large amounts of centralization around IBM may be a concern for many firms, indicating a heavy reliance on IBM's continued participation, but the disconnected nature of the network makes it clear that many firms are able to participate and create value without needing to be tightly tied to the core of the ecosystem. In essence, these firms on the periphery are taking advantage of opportunities for additional value creation within the ecosystem without the need to have expert knowledge of the internals of the ecosystem. This is in marked contrast to GNOME, where most firms are close to the internals of the community and must retain developers with intimate knowledge should the need for platform modifications arise.

There exist some differences between the communities that may be able to explain some of the variation between the two. There are far fewer projects in the Eclipse ecosystem than in the GNOME ecosystem, despite the fact that Eclipse has many more commercial firms contributing. This is largely a result of the need for projects in Eclipse to be approved by a council and have a formal incubation period before being accepted into the main Eclipse CVS repository. In contrast, GNOME freely allows the creation of new modules in CVS with no formal review process. Rather, the review happens when a module is proposed to be included with the full GNOME desktop distribution. Beyond sheer numbers, the scopes of the projects vary between the communities. Within Eclipse many of the core components are coalesced into the Eclipse platform project, while in GNOME these are separated into many different modules, primarily GTK+ and GLIB. This, while it is possible to compare these networks based on overall connectedness, caution should be exercised when inferring larger trends based on degree.

Finally, part of the difference and the increased connectedness in GNOME over Eclipse may be a result of the programming languages for each ecosystem. Eclipse is written in Java, which is designed to be object oriented and foster information hiding, while GNOME is written primarily in C, which is typically non-modular.

3.5 Conclusions

This chapter has made several contributions toward understanding how firms actually collaborate in an open source ecosystem. Within the Eclipse ecosystem there is very little collaboration between different firms – many firms work on only a handful of projects that are shared with other firms. Numerous companies are able to successfully compete and innovate in the Eclipse ecosystem by specializing in only a single component. This highlights one of the major strengths of the Eclipse Ecosystem – the ability for firms to make money by specializing in a small component.

However, most notable about the Eclipse community is the degree of centralization around Eclipse by IBM. This core component of the ecosystem is almost exclusively maintained and developed by IBM with little contributions from other other firms. Participants in the community generally believed that the heavy participation by IBM in the platform was a boon, which is most likely true given IBM's heavy use of platform technologies in other projects. However, not every project that IBM leads is guaranteed to survive and this creates a potential vulnerability for Eclipse participants. Most recently the Aperi project, which was an ambitious effort to provide a unified interface for large scale disk storage system management was forced to close after IBM withdrew it's support for the project[74].

Although there were no shipping commercial products based on Aperi, the presence of commercial products based on an Eclipse project is not enough for the community to keep a project alive without significant developer support. This was shown with the closure of

the Application Lifecycle Framework (ALF) and several sub-project streams in the SOA Tools Platform project within Eclipse. In these the Eclipse Foundation chose to archive the projects after there was a lack of developer interest in the projects – in spite of the fact that Serena software the primary developer of ALF was shipping a commercial product based on it[69]. This strategy on behalf of the Eclipse Foundation almost certainly helps to deter free-riding as it closes down projects that are populated exclusively by free riders.

In a broader sense, such a strategy by the Eclipse Foundation also empowers firms that maintain projects and may afford them an additional level of bargaining. The license for Eclipse does little to prevent a firm from taking the code of a project and creating a proprietary fork of the source code, where additions and changes are not licensed under the Eclipse Public license or shared with the upstream Eclipse Foundation[108]. Although it is unclear if a firm has ever explicitly used the threat of taking their work out of the ecosystem (interviewees indicated it would be heavily frowned upon within the community), members companies did perceive the risk inherent in having a single firm control most of the commits behind a project.

The comparison with GNOME showed that this distribution of commercial interest in Eclipse was similar to that of another mature Open Source community. However, within Eclipse the great centralization around IBM in the main `eclipse` project and the platform sub-project is not replicated in GNOME – instead showing the complete opposite result with all nearly all firms contributing to the core GTK project. The Eclipse Foundation frequently needs to counter the misconception that IBM is the only company behind

Eclipse and still owns the intellectual property, a statement that has been wholly false since the creation of the foundation, but their heavy reliance on IBM for work on the platform may serve to continue to support such perceptions. For the long term health of the community the Eclipse Foundation needs to find ways to incentivize other commercial firms to participate in core portions of the project that are not easily monetizable.

Chapter 4

Firms and Individuals: The Impact of Commercial Participation on Volunteer Participation[1]

4.1 Introduction

Early Open Source software projects, those that originated before the dot-com boom of the late 1990's and early 2000's, were typically developed by a core of distributed volunteers who freely exchanged ideas and code to create software for the common good of those contributing individuals[85, 127]. Over time, many of these projects became robust enough to

[1]This chapter is substantially based on a paper in progress with Jim Herbsleb and Robert Kraut.

attract a wide variety of contributors and end users resulting in the creation of a community of both developers and users. These communities were held together by a common set of norms and expectations. Central to most communities were the norms of sharing modifications to the software with the greater community and governance by a meritocracy – a system that gave those who made the greatest contributions to the community the ability to directly modify the project source code and control the overall direction of the software[26, 93].

Today, Open Source projects have evolved and many projects have a variety of commercial firms with full-time developers contributing to the project. In the Firefox web browser, Linux operating system, and OpenOffice suite of office programs, volunteer and paid developers from numerous firms collaborate to plan and develop key features of the software[46, 47, 64]. Before entering these communities, firms and their associated developers may have different goals from those of volunteers and in some cases may not be familiar with, or implement properly, the Open Source development process and community norms[125]. Their presence in Open Source projects could either foster or disrupt the original volunteer communities. While previous research has addressed the motivations and actions of individual commercial developers in an Open Source software environment, there has not been any analysis of the overall community impact of commercial participation[96].

The primary goal of this chapter is to determine how volunteer developers react to commercial participation in Open Source communities and to better understand what attributes

of commercial firms lead to successful commercial/volunteer partnerships. The following sections describe four hypotheses regarding how commercial firms could influence volunteer participation in Open Source projects. These hypotheses are based on the issues of having a commercial firm in a volunteer project. They are further refined by dividing the firms into two categories based on their broad interests within the community. The analysis of these hypotheses utilizes a multi-method approach consisting of qualitative interviews and quantitative analysis of archival data. Together this allows better understanding of how commercial firms affect volunteer participation in Open Source communities.

4.1.1 Commercial Participation and Positive Project Momentum

Commercial participation in Open Source communities often brings increased overall visibility to the community, increasing the value for participants, especially those who wish to use their participation to signal potential employers[64]. Commercial firms often provide wider distribution by adding the project to their existing offerings, garner media attention for the project by issuing press releases about the software, present information about the project and community at trade shows, and encourage their employees to become active within the community through the use of community run mailing lists and websites such as wikis, bug trackers, and weblogs. These community tools serve a dual purpose, in addition to providing a forum for developers to discuss, plan, and share information about their current tasks and ideas, these websites provide easy access for individuals outside the community to see and learn about what is going on in the community, including new

releases, new developers, and new corporations involved in the community[44]. For example, the GNOME project, a successful desktop environment for Linux and Unix systems, has a website called Planet GNOME[2] that aggregates the weblogs of most of the developers in the community. As community members write articles on their personal weblogs, the articles are automatically added the Planet GNOME where they appear next to a picture of the developer. Using this website individuals can visit a single website and get up to date information about development in the project and the lives of the primary developers. When an individual is only marginally involved with the community and looking for ways to get involved he may see the participation of commercial firms in these tools as validation of the project and seek to participate and contribute to such a project because of the possibility for future rewards, such as increased technical know-how or the possibility of career advancement.

If commercial developer participation validates the importance of the project and increases the momentum, then an influx of commercial firms and paid developers should attract volunteer developers and increase their participation in the community. The number of changes that full time commercial developers can make, and their high level of skill may speed up the development process, increasing the utility of the project to community members and contribute to volunteers identification and attachment to a successful project. Such attachment with projects, communities, and movements is an important factor in volunteers remaining active in a community and overall community success both in conventional volunteer organizations and Open Source communities [56, 62].

[2]Planet GNOME can be found at `http://planet.gnome.org/`

Hypothesis 1 *Participation of commercial developers on an Open Source project is associated with an increase in volunteer participation in the project.*

4.1.2 Negative Impacts of Heterogeneity

However, just as participation by commercial firms can provide resources to a community, they also introduce heterogeneity into the pool of developers. Whereas initially all of the volunteer developers may have been able to rally around the primary focus of the community, developers employed by commercial firms may be working just for a pay check, with little concern for overall community health and well-being. In the long run, such heterogeneity in workgroups decreases overall productivity and increases tension within teams [90, 129]. Open Source communities have additional issues of heterogeneity which result in decreased performance, such as personal ideology for community participation [106].

Hypothesis 2 *The participation of commercial developers on an Open Source project is associated with a decrease in volunteer participation in the project.*

4.1.3 Business Models and Community Norms

In the late 1990's when Open Source was first attracting interest from commercial firms, most had similar business models. These firms followed the model of Linux distributors, such as Red Hat, that took the output of the community as a whole, packaged it with documentation and additional software to make it easier to use, and then sold the complete

collection of software with enhanced support[116, 134]. These companies tied their financial success to the success of the Open Source community as a whole. More recently as the market has matured, additional business models have arisen that allow firms to isolate and derive revenue from a single component that is part of a larger community or start their own communities around small niche products[60].

Commercial firms were separated into two broad classes based on their business model and interactions in the community: community focused firms that package the entire output of a community, such as Linux distributors, and product focused firms that utilize only a portion of the output from the community for their products. Within the context of GNOME, most of the community focused firms are Linux distributors that have a vested interest in shipping a complete and usable desktop environment with their distributions of Linux. Product focused firms typically enhance particular components from a community, such as a component library, or focus on a particular application in their business model. Many small consultancies fit in the category of product focused firms – for example when a major electronics manufacturer was developing a way to stream media via the Internet, they contracted a firm that specialized in the multimedia framework that GNOME uses to extend the framework and develop substantial portions of the product.

As the community around GNOME was founded on the principle of creating a Free Desktop Environment, rather than a collection of individual projects, this may foster a communal spirit amongst volunteers. When combined with the nature of many tools, such as Planet GNOME, that provide an overview of the whole community and the fact that anyone

can easily participate anywhere in the community, it is likely that volunteers will identify more closely with community focused firms leading to an increased power in attracting new volunteer developers over that of product focused firms.

Hypothesis 3 *Community focused firms will have a more positive relation to the change in the number of volunteer developers than product focused firms.*

4.1.4 Cognitive Complexity at the Module Level

At the heart of Open Source projects is source code – files written in various programming languages that embody the primary functionality of the project. The code for complex projects may consist of hundreds or thousands different files, each performing a specific task. To assist in developer logistics and comprehension, within large projects code and responsibilities are typically broken up into smaller components, called modules[87]. For example, a simple email client may have three modules: receiving mail, sending mail, and graphical user interface. All work within a module must typically be carefully coordinated, since all parts of the module tend to be closely coupled. Work in different modules tends to be much more loosely-coupled, and typically requires much less coordination. Each module may have a set of developers who are responsible for maintaining the module and overseeing development. Organizationally, modules often replicate the structure of the larger project – complete with their own mailing lists, bug tracking, and social norms.

Because of the distributed nature of most Open Source software development, projects

have adopted strong norms of open communication and decision making. For example, the Apache project requires that all decisions reach consensus on publicly accessible mailing lists. However, collocated developers employed by a commercial firm who work closely together have decreased incentive to post to the project mailing lists and maintain the transparent decision and documentation process. Such a process increases the cognitive complexity of code and prevents volunteers from fully understanding the logic of the new code. The loss of open discussion allows collocated developers to create code that is less modular making future changes more difficult further decreasing participation [66]. Because commercial developers work full time, they change project code much faster than volunteer developers. A survey of volunteer Open Source developers found volunteers average 14 hours a week on Open Source projects only a third of a standard 40 hour work week for commercial developers[62]. These issues posit a real danger that as developers from commercial firms modify code within a module of a module of a project, it will become increasingly difficult to for volunteer developers to comprehend the set of changes, forcing the volunteers previously working on the module to migrate to alternate modules within the project or leave the project completely.

Hypothesis 4 *The participation of commercial firms in modules of an Open Source project is associated with reduced volunteer participation in those modules.*

4.2 Research Method

Open Source software projects have rich historical archives of communication. For many projects, every communication, debate, and decision is automatically recorded by project support tools. While it is possible to gain useful insights into a community using just the archival data, understanding the context and ensuring correct interpretations of the data require qualitative as well as quantitative analysis. This combination of qualitative and quantitative research techniques allows us to understand the nuances of how communities and commercial firms interact and is particularly helpful in the cases where commercial firms make decisions regarding project participation outside the framework of the project.

Two studies were conducted focusing on a single large Open Source community. The first study was a qualitative study to identify the views of developers toward commercial participation and to provide additional background context about the community. The second study analyzed quantitative data obtained from the community in order to evaluate hypotheses regarding commercial participation in Open Source suggested both by previous research and the results of the qualitative study.

4.2.1 Community Background

Our research focuses on the GNOME project, a large and successful Open Source desktop environment started by volunteer developers in 1997 as a response to the lack of a completely free and Open Source desktop environment for Linux and other free computer

operating systems. By many metrics, this is a highly successful project: more than 10 years of history, stable releases every six months, and a continually growing user base[39, 57].

GNOME is the desktop environment for computers from Sun Microsystems, software from the project is in use in a myriad of devices like the One Laptop Per Child and Nokia n800 series of Internet tablets, numerous startup firms have created solid businesses around the project, and it recently was made available direct from Dell computers as part of their option to provide Linux on new computers. In our period of analysis, which goes from the origins of the project in 1997 to late 2006, there were over 1200 individuals who had "commit" status – the ability to directly modify the project source code without needing to go through an intermediary and almost 1000 different components in the shared source code repository. The community coordinates most of their activity through Internet enabled tools such as a shared bug tracker, mailing lists, and real time chats. The community, although originally comprised only of volunteers, has adopted modern software engineering practices such as release reviews, formal bug tracking, and project roadmapping [38] and faces many of the coordination and collaboration issues found in most distributed teams from cultural differences that frequently arise between Americans, Europeans, Australians, and Asians to the need to schedule board conference calls in such a way that only a single member has to take the call in the middle of the night[57].

One of the key elements of the GNOME project is that it is composed of many smaller projects of varying size, complexity, and maturity. For our purposes, when I refer to the "GNOME project" I mean this larger community, and a "project" is one of these smaller

projects in the community. The community operates as a federated system, giving each project with the opportunity to control their own outcomes and chart their own roadmap subject to some broader constraints and goals developed by the community. When a project has reached sufficient maturity, the developers may apply to have that project included as part of the main community software distribution, greatly increasing the probability that the project will be included as a default component with new installations of Linux and granting the project a large userbase. Most projects in the community have their own mailing lists and bug trackers and are generally managed by individuals working on those projects. Most community participants are active on multiple projects, but because there are hundreds of projects within the community, there are no developers who are active in, or able to monitor all the projects.

The community has a track record of commercial investment. During the dot-com boom of the late 1990's several firms were created to customize the project, develop components, and provide support for users of the software. However when the bubble burst in 2000-2001, many of these firms went bankrupt or left the market, leaving critical components largely unmaintained. The community slowly built up commercial support again and now has significant corporate investment from firms that distribute software as a component of the Linux operating system, and from other firms that utilize the software as a base toolkit that can be used for the design and manufacture of embedded devices such as PDAs and mobile phones.

4.3 Study 1: Developer Interviews

The first study was a set of interviews designed to better understand the community and the role of commercial firms within the community. The first author attended one of the two major annual face-to-face meetings for both volunteer and commercial community participants. These events are generally considered to be one of the highlights of the year for the project and take place shortly after the major releases of the software, approximately every six months. To encourage participation by volunteers in the conferences, the GNOME Foundation provides travel stipends to volunteers in the community to attend the conferences. While this helps volunteers attend the conference, because of issues with getting time away from work or school and the limited number of stipends, the population at conference typically under-represents volunteers relative to their contribution to the community.

Before attending the conference, key individuals were identified and contacted to schedule the interviews, and the most active firms in the community were researched and classified according to their business model within the community. During the conference, a total of eighteen individuals were interviewed over the course of three days. Interviews were semi-structured, lasted twenty to forty minutes, and were conducted during breaks in the schedule. Each interviewee was asked for the relevant professional background, how they got involved with the community, where they currently participate in the community, how they relate to commercial developers in the community, and if they believed our divi-

Table 4.1: General Description of Interviewees

Total Interviewees	18
Commercial Developers	50%
Volunteer Developers	50%
Student Volunteer Developers	17%
Commit Access	78%
Self-Described as Developer	89%
Self-Described as Community Support	11%
Longest Participation	10 years
Shortest Participation	11 months
Median Participation	3 years

sion of firms into community focused and product focused classifications was accurate.

General descriptive statistics about the interviewees can be found in Table 4.1. Of interest is that only fourteen of the eighteen individuals could directly commit to the project source code – two of the newer developers, one commercial and one volunteer, still needed to contribute through intermediaries and neither of the individuals who self-described their role as community support could make changes directly to the source code. As previous studies had shown that Open Source communities were typically only 1.5% female[40], it was not unusual that all interviewees were male.

The nine volunteer participants had varied backgrounds. Three of the volunteers identified themselves as students who primarily participated during their free time. The other six volunteers indicated their use and participation in the project was at least marginally related to their roles at work. For example, two of the volunteers were IT support staff in environments where GNOME was used as the desktop environment. All six of these

non-student individuals admitted to writing code and contributing to GNOME while they were "on the clock", even though their jobs had no role that involved GNOME. These participants believed their participation was relevant to their jobs and participation improved their performance at work.

The participants came to the project through a variety of routes. Most first became interested in the community because of their general interest in Linux and technology, but their reasons for changing from a passive community member who only uses the software to an active, contributing, member varied. Three of nine volunteer developers indicated that another individual working in the community had played a very large role in bringing them in to work on the community. Two of the developers indicated they started submitting changes to a project in the community and were later offered the chance to become maintainers of the project. The remaining four volunteer developers could not identify a specific reason they became more active in the community. Five of the commercial developers were active as volunteers in the community before they were hired. The remaining four commercial developers were hired by the firm for other projects and later shifted to projects in the GNOME community.

> "If it weren't for [commercial developer name], I wouldnt be involved in the community. He saw my postings on the mailing list and encouraged me to get more involved. About a month later he asked if I would like to maintain the project."
>
> –Volunteer developer speaking about how he became involved

4.3.1 Views of Commercial Participation

Both commercial and volunteer developers thought commercial developers provided manpower and the focus necessary to accomplish tasks that volunteer developers lacked the skill or motivation to accomplish. Additionally, the commercial developers believed their firms provided a marketing force for the community, increasing the appeal and bringing in more individuals to work on and participate in the community.

Volunteers generally welcomed the expertise and effort that commercial developers provided. One volunteer explicitly stated he hoped that his participation would be noticed by commercial developers and they would offer him a job, as they had for one of his friends. Three of the volunteer developers believed there were times when the heterogeneity introduced by commercial developers was beneficial – in particular the skills of commercial developers were sought for highly technical areas such as system performance and low-level libraries that volunteers often could not develop. None of the developers, volunteer or commercial, ever mentioned intentionally treating another individual differently because they worked for a different firm or were a volunteer, although a commercial developer did indicate that he believed code written by volunteers wasn't always as useful or reliable as code written by professional software engineering employed by his firm. At a modular level such a comment highlights the differing directions and goals of commercial firms and volunteer developers.

Two of the commercial developers who began working in the community as volunteers

expressed a small amount of frustration in aligning the goals of their firm and the community – possibly alienating volunteers in the community, but generally thought their firms had found ways to succeed. In one case the firm adopted a dual process model for participation in Open Source, where developers had individual responsibility for ensuring their participation was congruent with the values and norms of both their firm and the project. Internal to their firm they had to follow the roadmap and processes for the firm's product, while at the same time they needed to follow and work within the roadmap created by the community. This model caused many problems for the firm because the roadmaps diverged as the project progressed. In the end, the commercial developers de-emphasized the roadmap of their company, working harder to fit their development into the process of the community. The developers perceived that this led to a slow down in production and persisted until they were successful in convincing their managers to adopt and internal process that was much closer to the communal development processes. Although this caused contention within the firm, it was thought to be best for the community.

> "I certainly would not want to see commercial participation go away. But I think there are things that some companies should be more careful of when working in the community. At [firm name], weve been very careful how we work with the community."
>
> –Developer at community focused firm

The need to be careful when choosing how to participate was echoed by a commercial developer who had been active in the community for more than five years and had worked with multiple firms. He was currently employed by a product focused firm and was critical

of his firms participation; believing that his current firm had little respect for the community norms. Rather he believed his firm was involved only the sake of exploiting the community for their own products, and had little interest in the health and values of the overall community. This view was in sharp contrast to his previous experience at a community focused firm that he described as fostering involvement within the community. This developer left the product focused firm and the entire community shortly after the conference and his departure stirred up debate within the community about how firms should interact with each other and volunteers.

4.3.2 Classification of Firms

The interviewees were asked about their views of the nine largest firms (as measured by the number of changes made to the community source code repository). As researchers, we had previously classified the firms according to business model within the community. Five of the firms were product focused firms, which worked primarily within smaller areas of the community code, and four were community focused with contributions to many projects within the community. A brief description of each of these firms can be seen in Table 4.2. Each of the interviewees was provided a description of our classification scheme and asked to classify each of the nine firms. Out of the 162 classification tasks across interviewees, only two were not in agreement with our classification (Fleiss $\kappa = 0.953$). The two points of disagreement were both employees of a firm classified as product focused who believed their firm was better classified as a community focused firm.

Table 4.2: Major firms participating in the community as measured by the number of commits to the community source code repository.

Product Focused Firms	
Firm A	A large IT firm that became involved in the last five years through the purchase of Firm B. Migrating from a community focused to product focused firm.
Firm B	A medium firm that developed enterprise class software and provided services for the community. Purchased by Firm A.
Firm C	A small firm that assists in application development for the embedded market.
Firm D	A small firm that produced enterprise class applications for the community. Ceased operations in 2002.
Firm E	A small venture capital funded firm that developed software and sold integrated services for the community. Ceased operations in 2001.
Community Focused Firms	
Firm F	A large Linux distributor and long time supporter of community.
Firm G	A large IT firm that uses the community software to compliment hardware offerings.
Firm H	A Linux distributor that historically shipped a desktop environment from a competing Open Source community and had small participation in the community.
Firm I	A medium Linux distributor that historically supported the community and that of its competitors.

However, when asked about perceptions of specific firms the views of interviewees varied. In particular, most of the volunteer developers believed that product focused firms had more difficulty working with the community than community focused firms. Attitudes were almost universally favorable toward community focused firms. In contrast, develers had mixed perceptions of product focused firms. In the words of one volunteer these firms, were viewed as being guilty of "not caring about volunteers." Another volunteer, who maintained a project within the community that had contributions from about ten developers, was extremely skeptical about participation by a major product focused firm, despite being good friends with many of their developers and contributing to other projects maintained and stewarded by the firm. He expressed concern about the method of participation by the firm and the fact that they didn't require everyone to go through the same community socialization process before gaining committer status. This led him to be wary of contributions to his component from the commercial firm. Later in the interview process, when five of the other volunteer developers were asked specifically about this firm, they all echoed similar concerns about the firm's participation.

> "I dont think the commercial firms have the same interests as volunteers. If they submitted code to my project, I'd accept it, but if they started to submit lots of code, I'd start to look a lot more at where the project was going."
>
> –Volunteer developer and project maintainer

These interviews paint a mixed picture with regards to commercial development. While most developers indicated that they appreciated commercial development in the commu-

nity, a substantial portion of the developers were skeptical about the behavior of these commercial firms. The views expressed toward commercial firms by the developers indicate that there may be a relation between business model and perceived attractiveness of commercial firms in the community. In particular, there was some preliminary support for commercial firms attracting volunteers (hypothesis 1), but it was not universal. There was also a difference in perception between the firms classified as community focused and those classified as product focused, with the product focused generally slightly more negative, lending support for hypothesis 3 and hypothesis 4.

4.4 Study 2: Quantitative Analysis

Theory surrounding the issue of commercial participation in Open Source communities and the interviews conducted in the first study provide a foundation for the second study, a longitudinal analysis of historical data obtained from the community. I begin by further validating the classifications by business model proposed to, and validated by, the interviewees through an analysis of three kinds of behavior of commercial and volunteer developers: 1) open and potentially non-technical interactions in a forum with little learning curve (mailing lists), 2) a focused technical forum open to anyone with a moderate learning curve (bug tracking system), and 3) the highly technical interactions that build the software and require significant dedication and skill to understand and participate in (project source code). The data are then used to develop profiles of developers and projects and test for the effects of

commercial participation on volunteer participation at the project and module levels.

4.4.1 Data Collection and Analysis

Most Open Source projects follow a set of norms that are generally referred to as "the Open Source process." A key component of this process is the collection and archival of nearly all communication data as a form of organizational memory and as a tool for developers and users to later reference. In most communities public mailing lists are archived where they can be easily indexed and searched, bug tracking systems provide a complete audit history of every change made to each bug report, and a version control system manages and records all changes made to the software. When using a version control system, each developer downloads a complete copy of the code for the project, makes and tests their modifications, and then sends information about the files that were modified back to a main server in a single action called a commit. Each change is tracked in the system, allowing developers to revert to a previous point in the development process or "roll back" changes that may have been detrimental to overall development while providing a method of providence for all code modifications [30].

Working with the system administrators in the community, archival copies of mailing lists, bug databases, and the community's version control system were obtained. During the period of study, the community utilized concurrent version system, CVS, as their version control system. It was set up in such a way that any developer with a CVS account could

commit directly to the repository for any project. To control the community and protect the project source code, CVS access was only granted to developers after a request was made by another developer to create the account, limiting the number of individuals who could contribute to those who demonstrated significant dedication to the project. Bugs were managed and tracked using the Bugzilla software package, allowing anyone with an account, available instantaneously through a web form, to submit and comment on issues related to a project. The mailing lists were managed using the Mailman software, and most lists were open to anyone with an email address.

Each of the tools utilized different account and identity management solutions. All accounts belonging to each developer were manually unified and linked together within the data set. This allowed us to simply and directly obtain all contributions for developers across different projects and mediums. Information about developers was augmented with employment information gathered from examining developer email addresses, signatures at the end of messages, blog postings, project web pages, and interviews with community developers. This provided the necessary information to classify a developers participation as volunteer, product focused commercial, or community focused commercial allowing the analysis of firm level behaviors in the community.

4.4.2 Product Focused vs. Community Focused Developers

The interviewees supported our idea that there were two different types of firms contributing to the GNOME project – community and product focused. Initially, based on some of the comments of the interviewees, it was believed that community focused developers might have more experience and thus be seen as experts in the community. This seemed reasonable, as the first firms to invest in the community were community focused firms. However, there was no statistical difference between the tenure in the community for product focused (mean of 5.24 years) and community focused (5.51 years) developers. Both had more experience in the community than did volunteers (3.93 years).

Several interviewees also believed that there were observable behaviorial differences between community and product focused developers. Based on interview responses and personal experiences in the community I identified and analyzed a set of behaviors that could be seen as pro-social and community building. These behaviors have the primary characteristics of showing an interest in the community beyond the narrow focus of products the developer is paid to work on or are behaviors which have a high probability of interacting with individuals in the community who are not already developers. As developers were active in the community for widely varying amounts of time, each activity was normalized by the number of years the developer was active in the community.

The first way that individuals outside the community are likely to interact with commercial developers is through project mailing lists. Individuals that start many new discussions

and reply to a variety of messages are likely to interact with a variety of volunteers and address issues raised by the community. Beyond being highly active, the number of mailing lists a developer posts to also increases the sense that the developer is building community. The community building effect is magnified it the developer is active on mailing lists serving projects that they have never committed code into.

I examined the mailing lists from the community and for each developer counted the number of messages posted, new discussion threads started, projects mailing lists they were active on, and the number of projects they posted messages to for which they had never contributed code. Each of these values was normalized by the number of years the developer had been in the community, as measured by the duration from their first observable contribution in any project to their last observable contribution (or the end of the data set if still active). I then took the mean across each of the classes of developer; volunteer, product focused, and community focused; and performed an ANOVA to compare the three means. The results as shown in table 4.3 indicate that there is a significant difference in participation patterns between the three classes of developers. In particular, commercial developers were found to be much more active on mailing lists. However, when tests were performed analyzing just the difference between product and community focused commercial developers; a difference was found only in the number of messages posted to project mailing lists.

A semantic content analysis of all email messages sent to public mailing lists was then performed. This identified elements that support information seeking behavior in the com-

Table 4.3: Mean Activity per Year on Mailing Lists by Class of Developer (superscripts indicate statistically different groups of means in each row)

Variable	Volunteer	Product Focused	Community Focused	P-value
Messages	39.04^A	87.10^B	135.80^C	< 0.001
Threads Started	13.85^A	34.42^B	56.37^B	< 0.001
Mailing Lists	0.37^A	0.68^B	0.53^B	< 0.001
Extra Mailing Lists	0.20	0.23	0.20	0.400

munity, such as posting email addresses of contacts and providing pointers to web pages. The results were then aggregated by whether the author of the message was employed by a community or product focused firm. This analysis found that community focused developers included 85% more references to email addresses, and 140% more references web addresses than developers at product focused firms (as measured by the proportion of words that were email addresses and web addresses). Both of these behaviors are pro-social and may help new community members become acquainted with the project and eventually contribute as developers.

Mid-level technical interactions on the Bugzilla bug tracking system may have similar affects in building community as posting to message to a mailing list. In particular, users are encouraged to post any bugs encountered to Bugzilla. These bugs are periodically triaged by a group of community members who then assign the bugs to project developers. At the simplest level, each bug is given a small message form that allows developers to post messages which are sent back to the original submitter and any other individual with interest in the bug. Often times, developers post messages indicating that a bug has been verified as present, asking for more information, or provide a workaround for the user

experiencing the bug. Individuals also may submit patches to bugs that address the bug, a behavior that could be seen by a volunteer as taking a significant interest in their issue. When working with Bugzilla, developers can mark bugs as fixed, indicating that the patch has been submitted and accepted. Finally, we can count the number of distinct projects a developer was active on within the Bugzilla system to get an idea of overall activity and also cross-reference this against activity in the source code repository to see where developers contribute to bug management but do not write code. We take this to be an indication that a developer is taking concrete action showing a broader concern for the project, beyond the local areas that are the focus of the developer's interest.

Each of the previously described metrics was collected for every developer and used the same method that was used for the mailing lists to normalize for the length of time the developer was active in the community. The metrics were aggregated by class of developer, and summarized in table 4.4. Surprisingly, while commercial developers had a greater frequency of activity, as measured by the number of comments, patches, and bugs fixed, they had the same relative amount of breadth in the system as volunteer developers. Contrary to our initial belief, when accounting for tenure in the project, product focused developers were active on project bug trackers for significantly more projects and projects for which they had written no code than community focused developers (as measured by the Extra Projects variable). However, as a whole, the ANOVA for these values were not significant.

Certain actions by commercial developers in the CVS code repository may also be construed as pro-social community oriented behavior. Working on a variety of projects within

Table 4.4: Mean Activity per Year in Bug Reporting Database by Class of Developer (superscripts indicate statistically different groups of means in each row)

Variable	Volunteer	Product Focused	Community Focused	P-Value
Comments	74.92^A	156.50^B	133.40^B	0.010
Patches	4.93^A	9.60^B	6.14^A	0.042
Bugs Fixed	1.44^A	3.80^B	7.30^B	< 0.001
Projects	2.60	3.54	2.43	0.141
Extra Projects	1.56	2.09	1.11	0.556

Table 4.5: Mean Activity per Year in CVS Repository by Class of Developer (superscripts indicate statistically different groups of means in each row)

Variable	Volunteer	Product Focused	Community Focused	P-Value
CVS Projects	13.72^A	5.42^B	15.29^A	0.002

the community probably shows that the developer has a greater interest in the overall health and well-being of the community. Analysis of the logs indicates that developers employed by community focused firms contribute are active on significantly more projects, as shown in Table 4.5. The increased participation across a wider number of projects confirms the responses by many of the volunteer interviewees who believed the product focused firms worked only in narrow niches within the community and that community focused firms spread their effort across multiple projects.

This analysis shows that there are sometimes dramatic differences in the patterns of participation between volunteer, product focused, and community focused developers. In general, all commercial developers are more active in the community than volunteer developers, community focused developers are much more active and visible on commu-

nity mailing lists and within the project source code. Furthermore, when analyzed using ANOVA, there is statistically little difference between developers at community focused firms and those at product focused firms.

4.4.3 Quantifying the Impact of Commercial Developers on Volunteer Participation

The community makes it very easy for developers to start a new project, leading to a variety of projects that contain only small amounts of code, or represent the efforts of only a single developer working on a very specific tool. To select projects that had a substantial community around them, I filtered the data selecting only projects with more than 20 developers, more than 100 bugs filed in the Bugzilla bug tracking system, at least one community hosted mailing list associated with the project, and more than a year of overlap between the source code history, mailing list archives, and bug tracker data. These requirements yielded fourteen projects from the community.

As the data was presented as a continual time series, there was a need to aggregate the data into longer time periods to facilitate the analysis. Time periods ranging from one week to six months were explored. At the shorter end, the data exhibited great variability from one period to another, especially with respect to participation by volunteer developers who often disappeared for weeks at a time due to commitments outside of the project. Longer time periods faced the opposite problem, the release cycle for GNOME is six months long

and longer time periods would fail to capture the different stages of development and would lead to substantial delay between effects. In addition, while some projects had almost ten years of history, other projects only had approximately fifteen months of history. Eight weeks was chosen as a compromise length of time to aggregate into time periods. Each release cycle for GNOME then contained three distinct time periods, enough time to show changes in participation without being subject to the noise of shorter time periods.

The number and identity of volunteer and commercial developers committing code during the period and the number of commits to the project during the period were recorded for each eight week time period. The distribution of the number of commercial and volunteer developers is highly skewed toward the lower end of the range and can be approximated with a log-normal distribution. To a lesser degree the distributions of the number of community focused developers, product focused developers, and commits are also skewed. To accommodate for this in the models, the logarithm (base 2) of these variables is used. Summary statistics and correlations can be seen in Table 4.6 and Table 4.7 below. Of note is the high correlation between the number of volunteer developers at time t and time $t-1$. This is a sign of a broader problem of autocorrelation in the number of volunteer developers across many time periods, which can be seen in figure 4.1. This high level of autocorrelation can be compensated for by examining the difference in the number of volunteer developers between different time periods. The autocorrelation of this new variable are seen in figure 4.2.

The correlations between time periods are now significantly less, with the maximum

Table 4.6: Summary Statistics of Data Collected from 14 projects at 8 week intervals (601 total observations)

Variable	Mean	Median	Skewness	Kurtosis	Std Dev	Max	Min
$VolDevs_{i,t}$	4.01	3	1.24	0.96	3.94	18	0
Number of volunteer developers contributing code to project i at time t							
$ComDevs_{i,t}$	3.57	2	2.19	4.70	4.87	26	0
Number of commercial developers contributing code to project i at time t							
$ComDevs_{CF,i,t}$	1.12	1	2.16	4.77	1.66	9	0
Number of commercial developers from community focused firms contributing code to project i at time t							
$ComDevs_{PF,i,t}$	2.65	1	2.84	8.22	4.49	26	0
Number of commercial developers from product focused firms contributing code to project i at time t							
$Commits_{i,t}$	114.97	46	2.98	11.54	175.64	1407	0
Number of commits made by all developers to project i at time t							
Observations, N	42.92	49	-1.66	1.74	12.72	53	14

Table 4.7: Correlations of Data Collected at Project Level after log transformations.

	$VolDevs_t$	$VolDevs_{t-1}$	$ComDevs_{t-1}$	$ComDevs_{CF,t-1}$	$ComDevs_{PF,t-1}$	$Commits_{t-1}$
$VolDevs_t$	1.0000					
$VolDevs_{t-1}$	0.8263	1.0000				
$ComDevs_{t-1}$	0.3755	0.3921	1.0000			
$ComDevs_{CF,t-1}$	0.4346	0.4258	0.6272	1.0000		
$ComDevs_{PF,t-1}$	0.2733	0.2773	0.9272	0.3555	1.0000	
$Commits_{t-1}$	0.6655	0.7331	0.6400	0.4208	0.5504	1.000

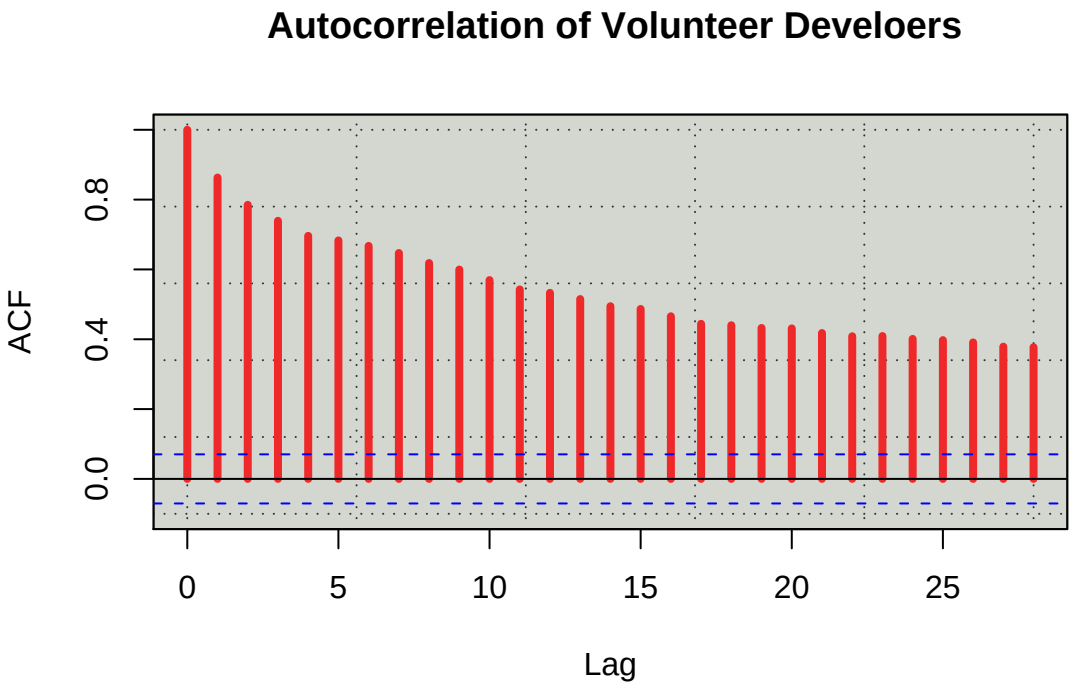

Figure 4.1: Autocorrelation of the number of volunteer developers between time periods

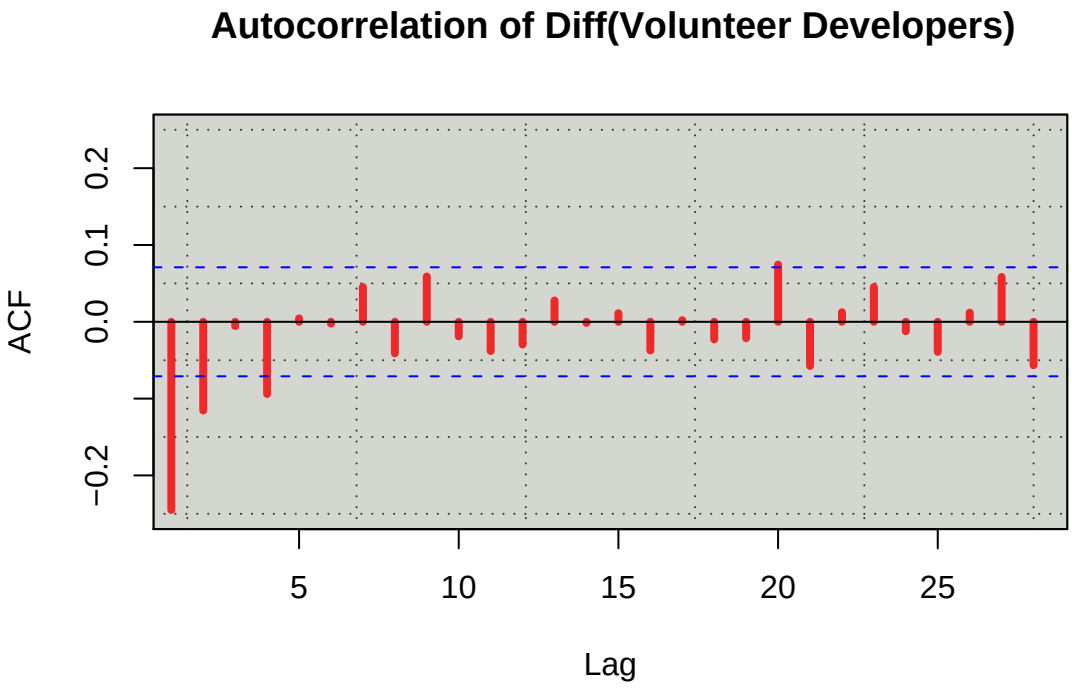

Figure 4.2: Autocorrelation of the diff'd number of volunteer developers between time periods

correlation ocurring at time lag 1. Figure 4.2 suggests that the difference in the number of volunteer developers may take on an AR(2) model, which has interesting implications for predicting the future number of volunteer developers, but is beyond the scope of this work. This reformulation of the response variable as a diff continues to satisfy the original hypotheses.

The next step in building a model with time lagged elements is to evaluate the cross correlation between the response variable and possible predictor variables, as seen in figure 4.3. In all cases the cross correlations are below 0.3. Most interesting is the result shown at the top of the figure, illustrating the dramatic change in sign between a lag of zero and a lag of one time period. This indicates that periods of highest volunteer activity, as measured by the number of volunteer developers, often attract additional developers, but the next time period many of these developers leave the project.

A variety of different control variables were explored for the projects, including number of email messages, total commits to project source code, and number of files active during the time period. These variables were consistently highly correlated (> 0.92) with one another, so the number of total commits was selected as a control for the general level of project activity. An additional control variable of the time period of the observation was also included to account for a general observation that projects frequently loose developers over time. As some projects had significantly longer history than others, this variable also had a log transformation applied.

I begin with a regression model that predicts the change in the logarithm of the of the

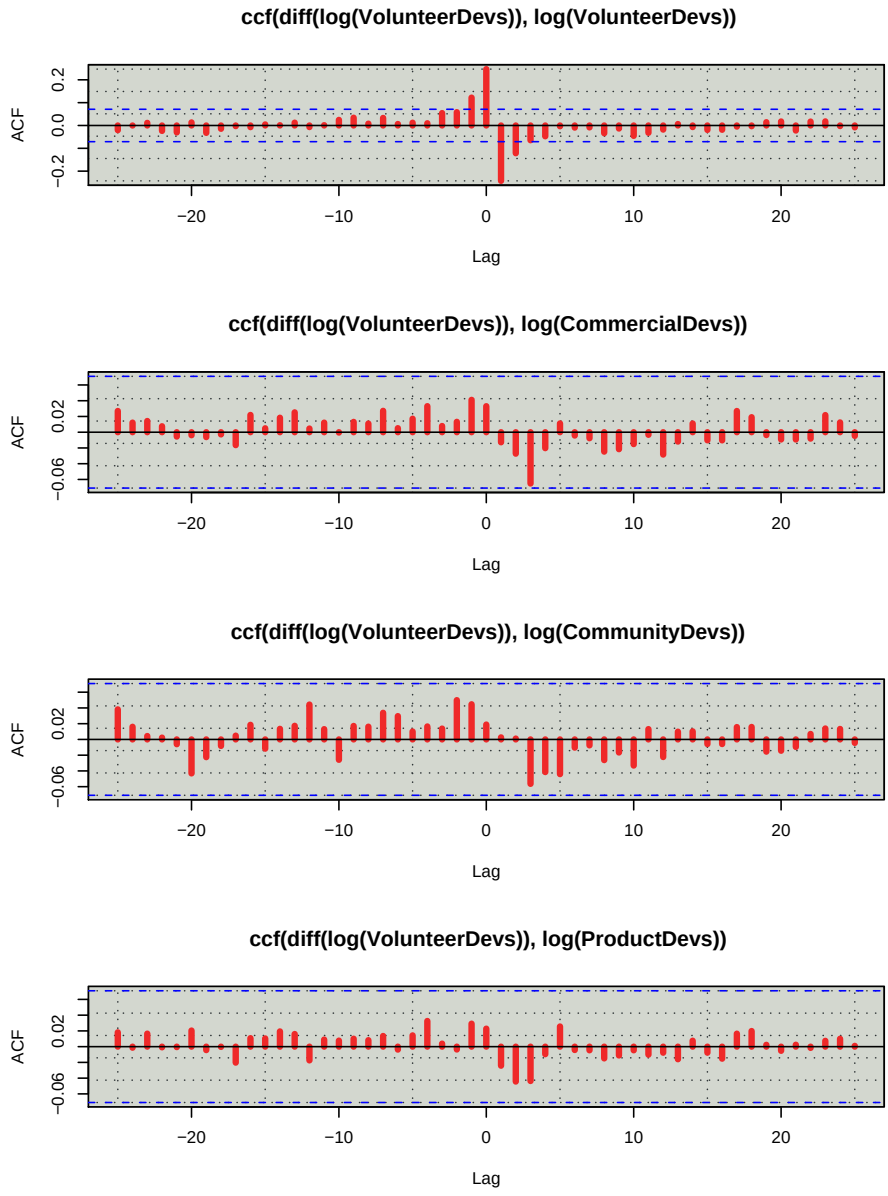

Figure 4.3: Cross correlation of the diff'd number of volunteer developers with predictor variables. The top figure shows the cross correlation with the number of volunteer developers. The second figure shows cross correlation with the total number of commercial developers. The third and fourth figures show the cross correlation with the number of community and product focused developers respectively.

volunteer developers contributing source code to project at time t as a function of the log transformed number of volunteer developers, commercial developers, and commits at time $t-1$. To accomidate for the varying level of inherent attractiveness for different projects in the ecosystem, each project had a dummy variable applied to determine the intercept.. The regression model is shown in equation 4.1. Within this model, the response varible is the difference in the log of the number of volunteer developers for project i from time period $t-1$ to t, $\text{diff}\left(\log\left(VolDevs_{i,t}\right)\right)$ and the predictor variables are the intercept for the project, α_i, the log of the number of volunteer developers for project i the previous time period, $\log\left(VolDevs_{i,t}\right)$, the log of the number of commercial developers for project i for the previous time period, $\log(ComDevs_{i,t-1})$, the number of commits for project i for the previous time period, $\log(Commits_{i,t-1})$, and an identifier for the current time period, $\log\left(t\right)$.

$$\text{diff}\left(\log\left(VolDevs_{i,t}\right)\right) = \alpha_i + \beta_0 \log(VolDevs_{i,t-1}) + \beta_1 \log(ComDevs_{i,t-1}) + \beta_2 \log(Commits_{i,t-1}) + \beta_3 \log\left(t_i\right) + \epsilon_{i,t} \quad (4.1)$$

The results of the model, reported in Table 4.8 indicate that an increase in the number commercial software developers working on a project has no effect on attracting additional volunteer developers to the project. However, general activity in a project, as measured by the number of commits, is related to an increase in the number of volunteer developers in

Table 4.8: Hypothesis 1 and 2 – Regression coefficients predicting change in number of volunteer developers by project (equation 4.1)

Variable	Estimate	Std Error	P-Value
$\log(VolDevs_{i,t-1})$	-0.4779	0.0356	$< .001$
$\log(ComDevs_{i,t-1})$	0.0411	0.0311	0.187
$\log(Commits_{i,t-1})$	0.0819	0.0199	$< .001$
$\log(t_i)$	-0.0471	0.0156	0.003
$R^2 = 0.211$, $AdjR^2 = 0.193$, $DF = 746$, $p < 0.0001$			

the next time period. This effect is tempered by the general trend of projects attracting fewer new volunteer developers as they grow older. Coefficients for project dummy variables, not all shown, ranged from 0.09 to 0.89 and were significant at the $p < 0.001$ level for 11 of the 13 projects. The model meets the requirements for a linear regression and the distribution of the residuals is rougly linear on a QQ-Plot, as shown in figure 4.4. Due to the lack of significance of $\log(VolDevs_{t-1})$ it is not possible to reject or support hypothesis 1 or hypothesis 2 with this model.

As I have shown there is a marked difference between the methods and magnitudes of participation of the two types of commercial developers: product focused and community focused. In Equation 4.2 I expand on the model to differentiate between participation of developers for community focused firms, $ComDevs_{CF_{t-1}}$, and product focused firms, $ComDevs_{PF_{t-1}}$. The same data are used with a regression model, with the regression coefficients presented in Table 4.9.

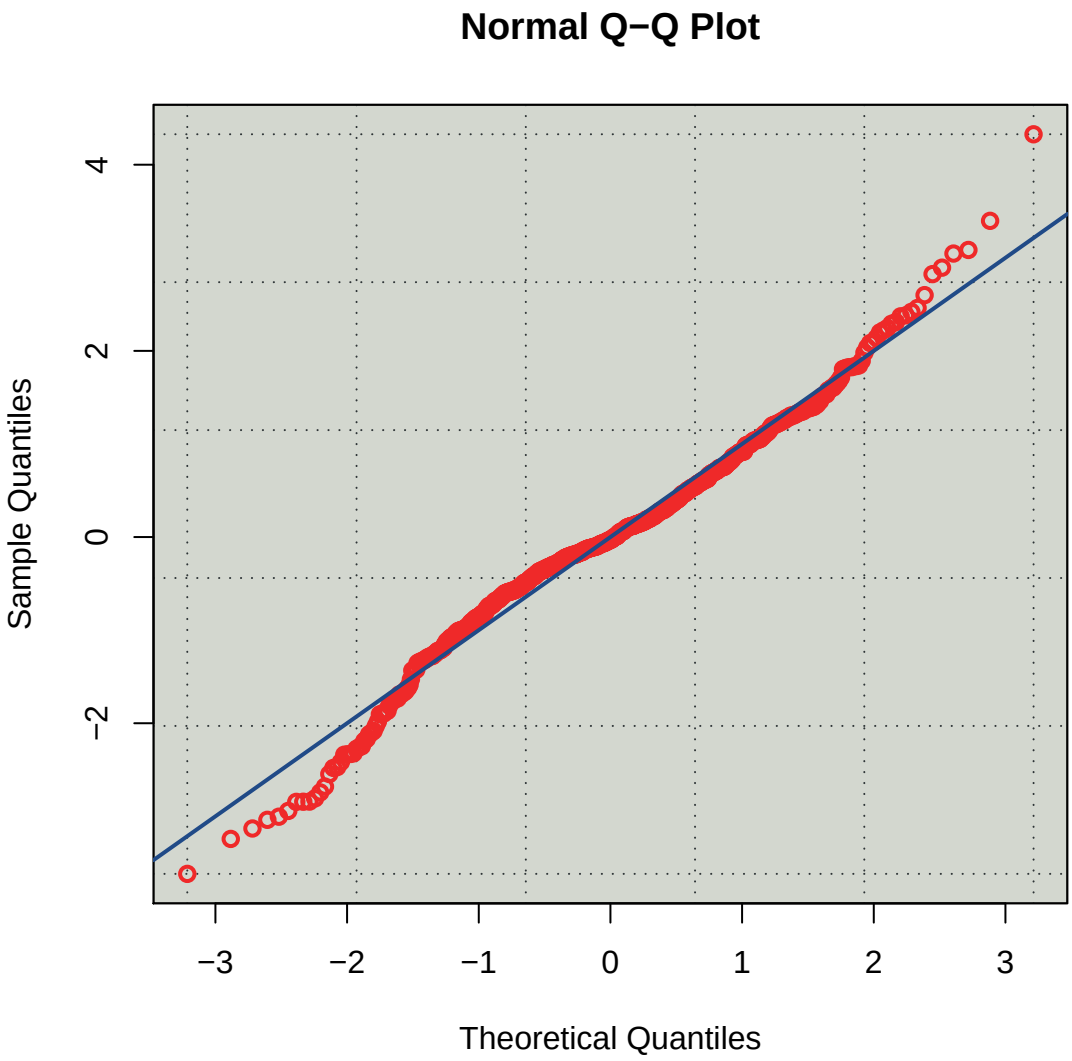

Figure 4.4: QQ-Plot of the residuals from fitting equation 4.1

Table 4.9: Hypothesis 3 – Regression coefficients predicting change in number of volunteer developers by project broken up by firm model (equation 4.2)

Variable	Estimate	Std Err	P-Value
$\log(VolDevs_{i,t-1})$	-0.4871	0.0356	$< .001$
$\log(ComDevs_{CF_{i,t-1}})$	0.0918	0.0350	0.008
$\log(ComDevs_{PF_{i,t-1}})$	-0.0212	0.0299	0.479
$\log(Commits_{i,t-1})$	0.0843	0.0194	$< .001$
$\log(t_i)$	-0.0401	0.0159	0.011
$R^2 = 0.217$, $AdjR^2 = 0.198$, $DF = 745$, $p < 0.0001$			

$$\text{diff}(\log(VolDevs_{i,t})) = \alpha_i + \beta_0 \log(VolDevs_{i,t-1}) + \beta_1 \log(ComDevs_{CF_{i,t-1}}) + \beta_2 \log(ComDevs_{PF_{i,t-1}}) + \beta_3 \log(Commits_{i,t-1}) + \beta_4 \log(t_i) + \epsilon_{i,t} \quad (4.2)$$

In contrast to the original model where there was not a significant relationship between firm participation and the change in the number of volunteers, when the firms are broken up by business model, we see a significant difference. Participation by developers from community focused firms has a significant and positive relationship to the change in the number of volunteer users, while participation by developers for product focused firms has no statistically significant impact. The other coeffecients in the model remain similiar to those shown in table 4.8 and the explanatory power of the model has increased slightly. Once again, the residuals of the model are close to normally distributed, as shown in the q-q plot in figure 4.5. This difference between developers at community and product focused firms lends support for hypothesis 3.

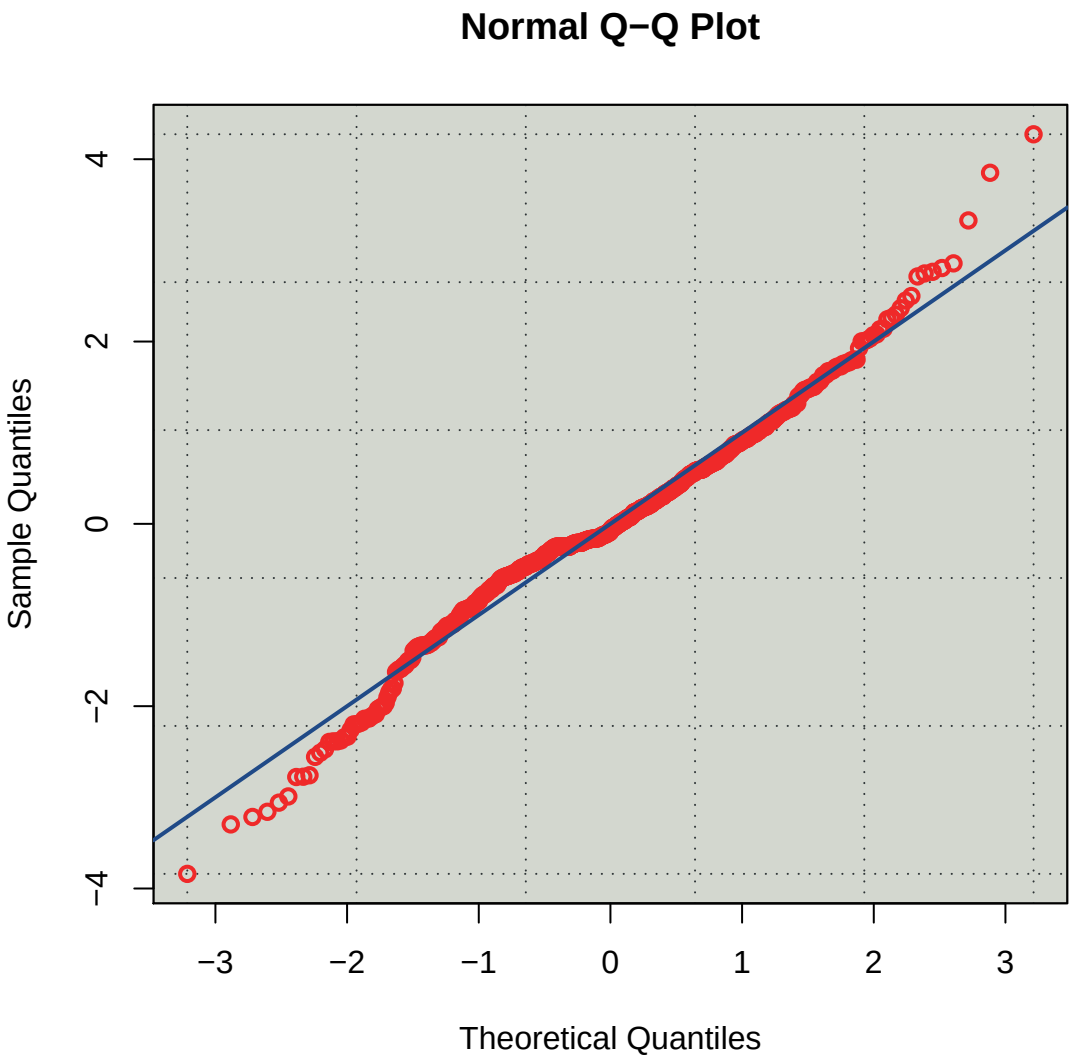

Figure 4.5: Q-Q Plot of the residuals from fitting equation 4.2

In order to evaluate Hypothesis 4, projects were subdivided into their constituent modules, in order to use modules, rather than projects, as the unit of analysis. While language specific methods exist to specify explicit module structures and infer implicit structure through static source code analysis, the code in the GNOME project is written in a variety of different languages and programming language specific methods are inconsistent and impractical. As an alternative, social network analytic clustering methods were used on the network of source code to approximate modules within the project. The CONCOR algorithm was used to produce eight clusters per project as it requires no additional information beyond link information when generating the groupings. Functionally, the computation views the network of files as a matrix and then attempts to rearrange the rows and columns of the matrix so entities that are structurally equivalent, meaning they link to the same set of other files, are grouped together [4]. While a variety of unsupervised clustering algorithms exist that determine an optimal number of clusters (e.g. Newman's algorithm [79]) these methods often produce more clusters than practical, leaving many clusters with only a single active developer.

My method of inferring code modules within a project utilizes a network structure where nodes in the network are files and edges are added between nodes if they were committed back to the central repository in a single commit. This approach is commonly used in software engineering research, and such links are often called "logical" dependencies[36] and has been shown to be particularly appropriate for measuring coordination requirements[14, 15]. This approach is based on the observation that files are

generally changed at the same time by the same person because there are important dependencies between them. In this way, network is generated where highly related files are densely clustered together. The CONCOR algorithm was run on this network for each project and configured to generate eight clusters, each approximating a module within the project. The number eight was selected as a compromise value that typically yielded multiple developers in each cluster without having clusters that contained all developers. The same summary statistics shown in Table 4.6 were generated for each module in each time period, yielding a total of 6360 observations.

The analysis at the project level was then replicated with the new data based on the clusters within each project. For this analysis, a new subscript, j, is added to the model indicating the cluster within project i. Intercepts are calculated for each of the clusters in the data, $\alpha_{i,j}$, and time is the number of periods since the start of the project. The complete equation is shown in equation 4.3.

$$\begin{aligned} \text{diff}\left(\log\left(VolDevs_{i,j,t}\right)\right) &= \alpha_{i,j} + \beta_0 \log(VolDevs_{i,j,t-1}) + \\ &\quad \beta_1 \log(ComDevs_{i,j,t-1}) + \beta_2 \log(Commits_{i,j,t-1}) + \\ &\quad \beta_3 \log\left(t_i\right) + \epsilon_{i,j,t} \end{aligned} \tag{4.3}$$

The new model testing for cognitive complexity issues was analyzed and the results can be seen in Table 4.10 and a Q-Q plot of the residuals can be seen in figure 4.6. The

Table 4.10: Hypothesis 4 – Testing for issues of cognitive complexity through the analysis of effect of commercial developers at the module level with pooled commercial participation

Variable	Estimate	Std Err	P-Value
$\log(VolDevs_{i,j,t-1})$	-0.5405	0.0142	$< .001$
$\log(ComDevs_{i,j,t-1})$	0.0038	0.0132	0.774
$\log(Commits_{i,j,t-1})$	0.0992	0.0079	$< .001$
$\log(t_i)$	-0.0022	0.0055	0.693
$R^2 = 0.226$, $AdjR^2 = 0.213$, $DF = 5996$, $p < 0.0001$			

residuals within the model deviate slightly from a normal distribution, however, this is not considered to be sufficient to jeopardize the results of the model. The effects of the model largely mirror the results found when analyzing the project level (see table 4.8. The lack of significance for the coeffecient of $\log(ComDevs_{i,j,t-1})$ does not allow either confirmation or rejection of Hypothesis 4. The developers from commercial firms were once again segregated by whether or not the developer worked for a community or product focused firm, resulting in equation 4.4.

$$\begin{aligned}\text{diff}(\log(VolDevs_{i,j,t})) &= \alpha_{i,j} + \beta_0 \log(VolDevs_{i,j,t-1}) + \\ &\quad \beta_1 \log(ComDevs_{CF_{i,j,t-1}}) + \beta_2 \log(ComDevs_{PF_{i,j,t-1}}) + \\ &\quad \beta_3 \log(Commits_{i,j,t-1}) + \beta_4 \log(t_i) + \epsilon_{i,j,t} \end{aligned} \tag{4.4}$$

After fitting the model using the availble data, shown in table 4.10, the familiar pattern of developers from community focused firms having a postive relation to the number of

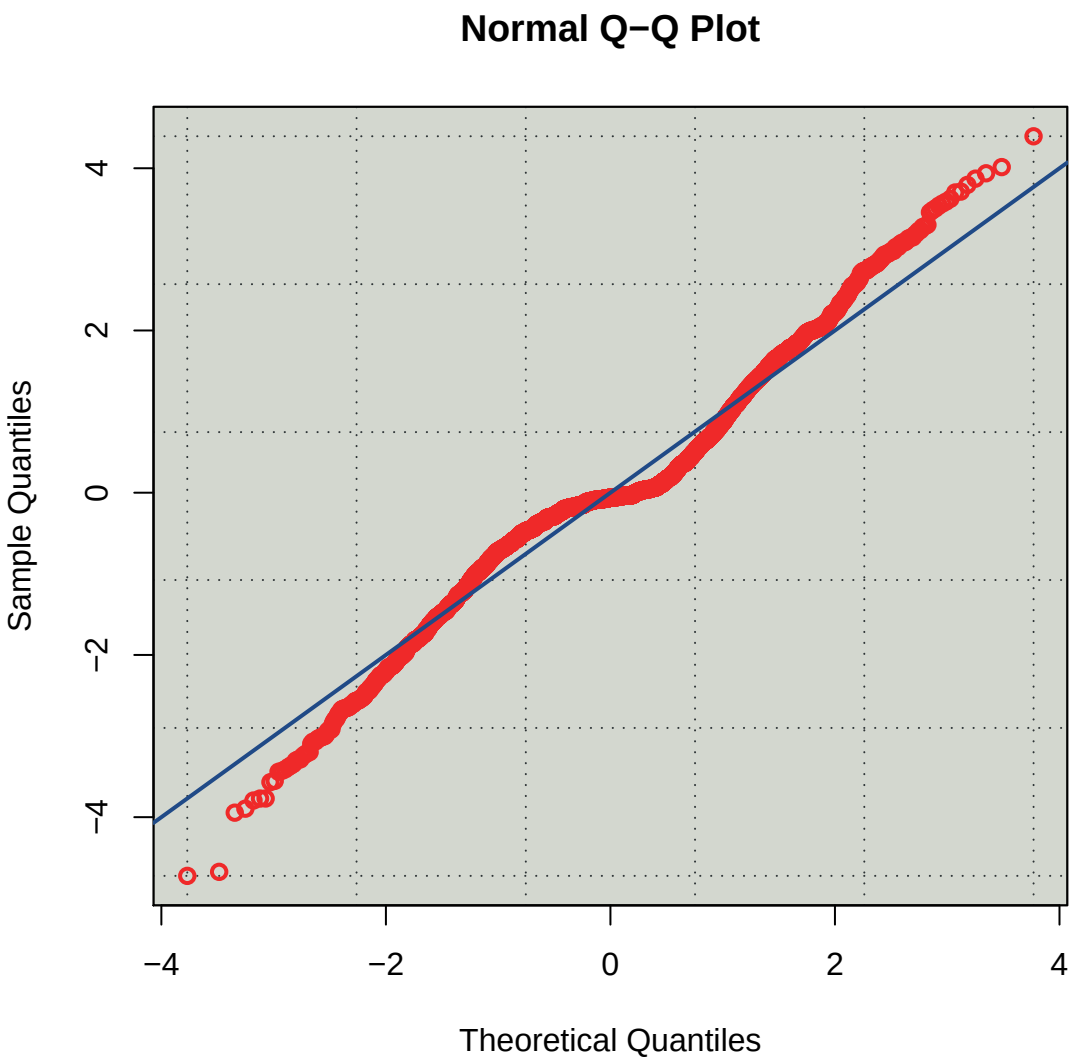

Figure 4.6: Q-Q Plot of the residuals from fitting equation 4.3

Table 4.11: Hypothesis 4 – Testing for issues of cognitive complexity through the analysis of effect of commercial developers at the module level

Variable	Estimate	Std Err	P-Value
$\log(VolDevs_{i,j,t-1})$	-0.5464	0.0142	$< .001$
$\log\left(ComDevs_{CF_{i,j,t-1}}\right)$	0.0616	0.0156	$< .001$
$\log\left(ComDevs_{PF_{i,j,t-1}}\right)$	-0.0284	0.0134	0.034
$\log(Commits_{i,j,t-1})$	0.0996	0.0075	$< .001$
$\log(t_i)$	0.0004	0.0055	0.934
$R^2 = 0.229$, $AdjR^2 = 0.215$, $DF = 5995$, $p < 0.0001$			

new volunteer developers and developers from product focused firms having a negative relation once again emerges. Therefore, this results in conflicting evidence about whether or not commercial developers increase the cognitive complexity at the module level and force volunteer users to leave. Therefore, it is not possible to reject Hypothesis 4, nor is it possible to support it.

4.5 Discussion

The results presented in this capter show that participation by commercial firms can have a positive impact on the participation of volunteer developers. At an overall level, it was found that at a macro level, the participation of commercial developers in an Open Source project did not have a statistically significant relationship to a change in the number of volunteers working on the project, preventing rejection of Hypothesis 1 and Hypothesis 2. On the positive side, for project maintainers, there was not significant evidence that at the macro level commercial participation caused volunteers to leave the community, suggest-

ing that project managers should not exert effort to actively deter commercial investment in their projects. Likewise, however, this research also shows that commercial firms should not expect to be greeted with entirely open arms in a purley volunteer community as their contributions may not be attractive to the community and may not attract additional volunteers to the project.

Another contribution is the identification and validation of two distinct classes of business models for firms participating in Open Source communities and the different impact they have on volunteer participation. These models closely aligned themselves either with broad community success or with a single project in the community. As expected, community focused firms were found to be highly visible on project mailing lists and wrote code for more projects than developers from product focused firms. The different business models were found to have very different impacts on volunteer developers with community focused developers attracting volunteers while product focused developers had no statistically significant effect. Given the predominant view of the interviewees that community focused firms were more aligned with the values and norms of the community, this supports the notion that the communities are sensitive to the values and norms of commercial particpations and inidicates that rather than valuing the views of a wide variety of participants with differing knowledge and goals, the community tends to shy away from heterogenity. This result should urge caution on firms wishing to participate in Open Source projects, and suggests that behaving in a way that supports the community may actually strengthen and enlarge it.

Finally, I analyzed whether or not the collocation of commercial developers and the possible increase in cognitive complexity had an effect on volunteer participation. Contrary to the hypothesis, it was found that this was not the case when commercial participation was pooled. There are multiple possible reasons for this, all of which may be subjects of future research. Developers for commercial firms may take extra care to ensure that they communicate the changes they make on public mailing lists or other forums. They may be faster at giving responses back on personal emails about projects. Perhaps the self-selection process in Open Source, which requires developers be able to figure the project source code with little assistance, draws developers who are able to compensate for such situations. A final possible explanation is that the norms of writing clean and modular code force all developers to write code in such a way that the advantages obtained through co-location are lessened to the point where they no longer impact the cognitive complexity of the code. When participation was segmented by the business model of each firm, it was once again found that participation by developers of community focused firms was associated with a subsequent increase in the number of volunteer developers, while developers of product focused firms were associated with a decrease in the number of volunteer developers. Further work should be conducted to identify why this effect persists at the micro level.

This research suggests both caution and some reassurance for firms considering a product focused relationship to an Open Source community. Our qualitative results show that volunteer developers frequently made negative comments about product focused firms, which is quite worrisome. On the other hand, increased participation of developers from

product-oriented firms did not drive volunteers away. It may be that the increased visibility that commercial participation lends a project offsets any negative effects from its perceived failure to uphold community norms.

No matter what the reasons for the increased success of community focused firms in attracting and retaining volunteer developers, firms continue to release projects both large and small as Open Source and they continue to take advantage of Open Source based technologies. We have seen that in this case, the dual worlds of volunteer and commercial can co-exist in an Open Source project with little danger of the commercial firm dramatically damaging the incumbent volunteers. Going forward, understanding the methods by which these firms attract and retain volunteer developers is an open research question that will yield great benefits for firms seeking to utilize this revolutionary software development model.

Chapter 5

Individuals and Individuals: Evolution of the Socio-Technical Congruence Metric

Early pieces of computer software were frequently written by a single individual. The bulk of VisiCalc, the first spreadsheet and the computer program that is frequently cited as the turning point for home computers from hobbyist toys to serious business tools was largely written by a single developer, Dan Bricklin, and later refined by another developer Bob Frankston[52]. In the context of such small teams of engineers, the need to manage information flow is small and can be managed by face-to-face meetings or emails between developers. As teams grow, however, the dependencies become more difficult to manage,

and work typically must be broken up into small components and handed off to individuals in a less collaborative fashion[70]. As a project progresses teams tend to naturally develop informal patterns of communication that address dependencies between these components and foster progress[35].

With large scale complex tasks, or the type now addressed by software engineering, even small changes in the system, either in the informal communication patterns or task dependencies, may have cascading effects throughout the project. The changes affect the task dependency structure and results in a misalignment of the informal communication patterns with the actual dependencies needed for the work, decreasing overall productivity[48]. In this chapter I expand upon the socio-technical congruence (STC) metric that is used to understand how team communications, both formal and informal, align with dependencies between tasks[15]. I approach this problem through an empirical study of the GNOME project. I begin by first reproducing a portion of the results of Cataldo et. al. This is notable because it is a replication within an Open Source community, which tend to be far more organic, relying on ad-hoc teams and informal communication processes for team coordination[132] and because of difficulty of collecting the requisite data for STC in less controlled software engineering environments.

I then propose several modifications to the metric that provide better and easier insight into team coordination by separating out the effects of increases in coordination requirements and the communication that addresses those requirements. Next, I address the changing nature of task dependencies in the organization. This is particularly important

for organizations with long running development cycles that wish to calculate STC on a rolling basis, as previously the task dependency network was assumed to be fixed. Finally, I address one of the major concerns with the overall validity of the metric, that of noise in the collected data and the possibility of falsely inferring or omitting data.

5.1 Organizational Congruence

In 1968 Melvin Conway proposed a concept which has since come to be called "Conway's Law". Briefly stated, he noted that organizations tend to mirror the products they design. For example, if a firm had three teams working together to create a compiler, the resulting compiler would likely be a three pass compiler[17]. In such a scenario the technical dependencies and organizational structure are in alignment and therefore the when technical issues arise they are largely contained to a single coherent team. While most organizations break tasks into smaller components for ease of project management[70], software engineering is one of the few fields that is explicit about this structure due to the concept of modules within modern software engineering[87]. Such a modular structure assists in understanding tasks and assignments of individuals to tasks within an organization.

Beyond merely structuring work, organizations serve as information processing units and dynamically adapt their social structure to create information conduits between different segments of the organization[21, 34]. As the tasks that each organizational segment performs become more intertwined, the amount of information exchanged between these

segments increases in response[22]. This relationship between the inter-related tasks of an organization, different segments of the organization, and communication between the segments is the heart of socio-technical congruence (STC), a metric designed to provide a quantifiable value for the degree that organizational communication matches coordination requirements[15].

The calculation of STC is formulated in matrix notation, although additional work by Valetto et. al. has formulated the problem in graph theoretic notation[119]. In both the matrix and graph formulation, three pieces of information are required. The first is a network of task assignments, $\mathbf{T_A}$. This binary matrix maps an individual member of the organization to tasks within the organization. In the context of a software engineering organization this may map individuals to modules of the project they have modified. The second component is the task dependency network, $\mathbf{T_D}$, which identifies the ways in which tasks have interrelated dependencies. Within software engineering this may show logical dependencies between files[36]. The final network needed is the network of actual coordination, $\mathbf{C_A}$. In the original work various different networks were tested for $\mathbf{C_A}$, including organizational structure, geographical proximity, and recorded communication between individuals[15].

Under a matrix notation, a network of coordination requirements, $\mathbf{C_R}$, is calculated by multiplying the task assignment network, $\mathbf{T_A}$, by the task dependency network, $\mathbf{T_D}$, as seen in equation 5.1. After computation, $\mathbf{C_R}$ is transformed to a binary matrix such that any non-zero cell in $\mathbf{C_R}$ is set to 1.

$$\mathbf{C_R} = \mathbf{T_A} \times \mathbf{T_D} \times \mathbf{T_A}' \tag{5.1}$$

The overall congruence for the organization is then the fit of the coordination requirements, $\mathbf{C_R}$, to the actual coordination, $\mathbf{C_A}$. This is based on the concept of organizational fit within organizations that relates the ability of a particular organizational design to carry out a task[8]. The calculation yields the proportion of links in $\mathbf{C_R}$ that are also present in a network of actual coordination within the team, $\mathbf{C_A}$. This calculation can be mathematically represented as the logical conjunction between corresponding cells in the $\mathbf{C_R}$ and $\mathbf{C_A}$ matrices, as shown in equation 5.2.

$$\frac{\sum (\mathbf{C_A} \wedge \mathbf{C_R})}{\sum \mathbf{C_R}} \tag{5.2}$$

One intriguing potential use of the STC metric is in the creation of tools to assist software developers. For example, a team with a tool that automatically calculates STC can quickly see where technical dependencies exist for which there is no corresponding communication to resolve the dependencies, also known as gaps. By directing communication to fill these coordination it is possible to reduce overall development time[14]. In a distributed team, such as an Open Source project, a tool that provides this direction is even more important as individuals have fewer chances for ad-hoc opportunistic collaboration and what communication is possible is typically over very lean media[44, 132].

Such a tool can also make a developer aware of new colleagues to consult with, a useful feature when someone first joins a project or returns from an extended hiatus. While many commercial firms may pair up junior developers with more accomplished senior developers, who already understand the structure of the code and the social network in the organization, this is rarely the case in Open Source projects. Providing a tool that takes advantage of STC to a new individual who seeks to contribute to an Open Source project could prove to be very beneficial to the new developer and the the project as a whole, as the new developer would have some context of whom they must coordinate with to accomplish their task[99].

5.2 Problems with Socio-Technical Congruence

While STC has shown to be a useful metric, there are several issues associated with the metric that have yet to be addressed. One is the lack of replication of the metrics. The data used in the original study required a significant amount of manipulation and "clean up" before valid results were obtained[15]. To the best of my knowledge, this result has not been duplicated on any other "real world" data set. Therefore, one priority was to run the metric on a less processed data set from a similar environment using primarily data that could be collected automatically from pre-existing tools such as version control systems, bug trackers, and project mailing lists.

Beyond the practical analysis needed to validate the metric, there are several issues that

reviewers raised regarding STC. Notably among these are the fact that STC is a network level metric, which provides a single numeric value for the overall organization. From a high level management perspective, this may be beneficial, but when designing tools for individual users, the slight change in a STC provided by a new communication link may seem diffuse and difficult to understand. I propose further formalizing how STC affects individuals and develop a method for generating STC scores for individuals in an organization.

Changes in social and technical architecture also pose problems to STC. A large scale re-factoring of a project will make many previous dependencies no longer relevant. The introduction of a decay factor, where the networks from the previous time periods are scaled by a factor < 1 before adding in the data for the current time period can help address these changes. I apply this decay not only to the task dependency network, T_D, but also to the actual communication network C_A to reflect loss of knowledge over time and the need to periodically refresh communication links in an organization.

There is also some debate about the structure of the task dependency networks in the metric. As it relates to software development, the task dependency network represents the logical dependencies between files in the project. So if file A and B were ever modified and committed back to the version control system during the same transaction, then a link will appear in the task dependency network. In the original work, this network was generated once, at the end of the observational period, representing the complete task dependency network for the entire history of the project. When performing a retrospective analysis

of a project, it is possible to generate and utilize such a network structure, however when designing tools for real-time use of the project, this is not possible. Furthermore, such a structure assumes that logical dependencies that occurred at the very early stages of the project never "time-out", and have a continual and lasting effect. This is also unlikely as dependencies that were handled long ago are unlikely to require frequent addressing by developers, or changes in project source may have made them irrelevant. For this reason, we compare the results between metrics where T_D is generated once for all time periods in the project, and where T_D is generated for each time period by using the sum of the logical dependencies before that time period. These models are then integrated with the decay parameters to obtain a robust model of task dependencies.

Finally, all of the data used in the calculation of STC are inherently noisy. This is particularly important when we consider the actual coordination network, C_A, which may be obtained from automated tools. Use of automatically collected networks from mediums such as email and real time chat are subject to both high levels of errors of omission – a failure to infer a link between two individuals where there should be a link, and errors of commission – incorrectly linking together two individuals on the basis of a communication that was not relevant to satisfying any coordination dependencies. There may be an opportunity to augment automatically collected information with data from surveys, which provide more information, but such surveys are time consuming to create and manage, and often indicate that individuals have problems remember to whom they spoke.

5.3 Replication of Original Results in Open Source

Calculation of STC requires several different networks from a project, many of which may not be easily available. For this study, I used data from the GNOME project, previously discussed in detail in chapter 4. Detailed data from the origin of the project in 1997 until the the end of 2006 were collected including a copy of the version control system archive, a complete copy of the bug tracking database, and messages from project mailing lists[1]. Identities were unified across data sources by examination of common email addresses, name recognition, and manual analysis. All 1218 individuals who contributed code to the project were manually verified and checked for accuracy by the author. Where uncertainty about the identity of an individual was found, the identity of that individual was verified with members of the community.

5.3.1 Selection of Projects

The GNOME project is a large and diverse community that maintains an open policy that allows developers to easily create new projects in the community. Often times these projects are "one-off" demonstration projects or simple toys that a developer was working on in their spare time and never gather any real traction. Other times the project may be a valuable component of the GNOME desktop environment, but maintained by a single developer. For

[1]I wish to thank the system administrators and my liaisons in the GNOME project for their help with this data collection. Specifically, thanks to Luis Villa for providing access to the bug database, Olav Vitters for providing copies of the source code repositories, and Jeff Waugh for lubricating the whole process.

this reason, the selection of projects was pared down using the same criteria as described in section 4.4.

5.3.2 Generation of Networks

As volunteer Open Source projects typically lack a formal hierarchy, and most members of the project are geographically distributed, it was not possible to test organizational hierarchy and geographic congruence. The actual coordination network, C_A, was generated by examining the bug trackers and project email lists for each month.

For email messages, networks were created by examining the message headers to identify message threads. A link was then created between all individuals participating in a particular message thread. This effectively generates a symmetric network that is a clique between all participants in the thread. A message was determined to be in a thread through one of two different methods:

- The `In-Reply-To` header in the message that indicates the unique ID of the message that is the parent in the thread. This header is automatically appended by most desktop mail clients and nearly all of the current webmail offerings. However, for extremely old archives, in particular messages dating from before 2000 this method is not always practically because the header is frequently absent.
- A heuristic analysis of the message subject, header, and content of the message to see if it is related to other messages. In particular, examining the `to` header of the

message in conjunction with the subject and examination of quoted text tends to accurately identify which messages are in the same thread. This method was used only in the case that the former method could not be used. It is based on the routines found in GNU Mailman[33], one of the standard tools for managing mailing lists and creating archives of mailing lists. This method was rarely needed for messages newer than 2000 as most mail clients now support `In-Reply-To` headers.

These communication data were augmented with data from the Bugzilla bug tracker database for the community. Within the Bugzilla data, two individuals were assumed to have communicated during a period if they both commented on the same bug. Thus, once again, a clique was effectively formed between all participants on a bug discussion.

The data from project mailing lists and bug tracker data were aggregated together and dichotomized to create a binary network representing actual coordination, C_A between individuals in the community.

The task assignment, T_A, and task dependency, T_D, networks in the community were generated through an examination of the CVS source code archives for the community. In this network structure, tasks were mapped to individual files and filtered to include only source code related files, eliminating documentation, project build files, images, and other non-technical elements. An individual was mapped to a task if the individual modified and committed the file back to the repository during that time period. Files were mapped as having a dependency if they were modified together in the same logical commit[36]. Commits were further filtered such that commits with more than 20 files were removed as

they most likely are part of large scale non-code changes such as changing licenses and updating version numbers.

5.3.3 Selection of Control Variables

Using data from an Open Source setting presents a variety of challenges that were not present in the study by Cataldo et. al.[15]. In particular, as this is a volunteer community with no central arbiter of qualifications or requirements, there is no way except through a survey to get typical control variables commonly used in software engineering regression analysis, such as education level and tenure. However, for the results to be viable a survey would need a response rate high enough to provide complete coverage for the projects of interest. This is simply not possible in an Open Source context.

However, it was possible to account for general properties of the bug in question. In particular the number of individuals involved with a bug and the number of changes made to the status of the bug have previously been shown to be highly related to an increase in time to resolve bugs[50]. The control variables proposed for the regression are:

- $numDevs$ – the number of developers who were active on that bug, either by commenting on the bug in the bug tracker, or committing code and tagging the commit with the bug number. Most of this comes from activity on the bug tracker as in practice, only 12% of bugs in the data set had commits that were easily tracked back to the bug. Contrary to the concept of the "developer-user" espoused in many pieces of

Open Source literature, most of the comments on bugs are not written by developers, and for that reason their participation is broken out here.

- $deltas$ – the number of changes made to the bug status. This number was incremented whenever the status of a bug was changed, such as going from "NEW" to "ASSIGNED", "NEEDINFO", or "RESOLVED". It also was incremented when the person the bug assigned to was changed.
- $deltasPeople$ – the total number of people who made delta increments to the bug. In practice only a handful of individuals ever changed the status of the bug thanks to some strong community norms. A high number of $deltasPeople$ typically indicates that the bug has some confusion about ownership or that it has been handed off between developers.
- $comments$ – the total number of comments left on the bug. Bugs that attract more comments are typically either high visibility bugs or controversial issues.
- $commentsPeople$ – the total number of people who have posted comments on the bug.

Another common variable which is often used in such regressions is the assigned priority of the bug. However, in practice most bugs, 69.2% in the case of GNOME, are never changed from the "Normal" status. Further, in most projects anyone can change the status of the bug, even if they are not affiliated with the project. This often leads to individuals that are new to the community finding a small bug that affects them and immediately marking it as a "blocker" because it impairs their use, when the "blocker" status is reserved for

Table 5.1: Correlations between control variables for regression in Open Source

	$deltas$	$deltaPeople$	$numDevs$	$comments$	$commentsPeople$
$deltas$	1.0000				
$deltaPeople$	0.7701	1.0000			
$numDevs$	0.4456	0.5186	1.0000		
$comments$	0.5241	0.4660	0.3281	1.0000	
$commentsPeople$	0.5838	0.6738	0.4983	0.62278	1.0000

bugs that absolutely must be fixed the next release of the software.

5.3.4 Results in Open Source

Analysis of the control variables yielded high levels of correlation between the variables as shown in table 5.1. This high correlation along with some preliminary analysis indicates that the combinations of $(deltas, deltasPeople)$ and $(comments, commentsPeople)$ are problematic for a linear regression model. I decided that $deltas$ and $commentsPeople$ should be dropped from the model to assist in producing a statistically valid result. Although high correlations between control variables still exists, there was little problem found with variable inflation and multicollinearity in the final model.

The regression model was then to predict the time to resolve a software defect, as measured in the $\log_2$ of days based on the congruence of the organization at the time and the control variables around the defect. The regression shown in equation 5.3 uses the above mentioned control variables and the overall STC as the independent variables. The network was broken up into eight week long periods and all software defects opened and

resolved in a period were given the same value of STC. Defects that spanned multiple periods were given a value of STC that was the average project STC across those periods.

$$LogResolutionTime = \beta + \beta_1 numDevs + \beta_2 deltaPeople + \beta_3 comments + \beta_4 STC + \epsilon \tag{5.3}$$

The regression was run with 26512 non-enhancment related bug reports from projects that were part of the GNOME ecosystem. The results are shown in table 5.2. As has been shown in previous research, the more developers that are active on a bug, the longer it will take to resolve[49]. In addition, the more people that have changed the status of the bug, indicating possible changes in ownership, the longer it will take to resolve the bug, although this effect is smaller than the increase from the number of developers. More communication, as measured by the number of comments on the software defect, reduces the overall time to resolve the bug. Finally, teams with high socio-technical congruence experience shorter times to resolve bugs. It is important to note that in this regression, the value of STC is calculated once per project, per time period, and not for each individual software defect. This shows that teams with high STC will perform better across the entire project.

Table 5.2: Regression Analysis of STC in Open Source

Variable	Estimate	Std Error	P-Value
Intercept	1.4297	0.0548	$< .0001$
$numDevs$	0.3202	0.0302	$< .0001$
$deltaPeople$	0.0794	0.0177	$< .0001$
$comments$	-0.0144	0.0036	$< .0001$
STC	-0.2804	0.1236	0.0233
R^2 = 0.126, $DF = 26507$, $p < 0.0001$			

5.4 Individualized Congruence

Within a small organization it may be easy to interpret how the communication patterns of an individual developer affect overall STC, but as the size of the network, both individuals and dependencies, increases the connection between individual actions and overall STC becomes more diffuse. Work was undertaken to address the disconnect between an individual actions and network level congruence. I begin by formally describing the individualized congruence, UIC of an individual, i, as the congruence of only those edges that are incident upon i in the $\mathbf{C_R}$ and $\mathbf{C_A}$ matrices, as shown in equation 5.4. In this notation, $\mathbf{C_R}[\mathbf{i},]$ is used to indicate the entirety of the column (or row) i in the coordination requirement matrix. All matrices in the calculation of UIC should be binary matrices to ensure the metric is in the range $[0, 1]$.

$$UIC_i = \frac{\sum(\mathbf{C_R}[\mathbf{i},] \wedge \mathbf{C_A}[\mathbf{i},]) + \sum(\mathbf{C_R}[,\mathbf{i}] \wedge \mathbf{C_A}[,\mathbf{i}])}{\sum \mathbf{C_R}[\mathbf{i},] + \sum \mathbf{C_R}[,\mathbf{i}]} \tag{5.4}$$

Likewise the concept of individualized socio-technical congruence is easily expandable to utilize weights of edges as proposed by Helander[119]. This new formulation, termed weighted individualized congruence, WIC, is shown in equation 5.5. In contrast to UIC, WIC utilizes networks that need not be binary. Therefore, the degree of coupling between tasks $\mathbf{T_D}$, and the frequency of an individual working on a task, $\mathbf{T_A}$, both play relevant roles in this calculation. In this equation we define $d(M)$ to be a function that takes a matrix M and dichotimizes it such that all cells greater than 0 are set to 1 and all cells less than or equal to 0 are set to 0.

$$WIC_i = \frac{\sum \left(\mathbf{C_R}\left[\mathbf{i},\right] \wedge d\left(\mathbf{C_A}\left[\mathbf{i},\right]\right)\right) + \sum \left(\mathbf{C_R}\left[,\mathbf{i}\right] \wedge d\left(\mathbf{C_A}\left[,\mathbf{i}\right]\right)\right)}{\sum d\left(\mathbf{C_R}\left[\mathbf{i},\right]\right) + \sum d\left(\mathbf{C_R}\left[,\mathbf{i}\right]\right)} \tag{5.5}$$

This formulation retains the same lower bound on individualized congruence of 0, but there is no upper bound of WIC. Logically, this formulation should reward individuals who not only communicate across links, but also pick those links where the most coordination is necessary. The $\mathbf{C_A}$ matrix is dichotomized to prevent easy tampering with the metric and also because of the inherent uncertainty already present in collecting information about the actual coordination in the network.

5.4.1 Distribution of Metrics

First, to understand the distribution of the metrics and how they may impact the regressions, a histogram of the distributions of both UIC and WIC for each developer at each time

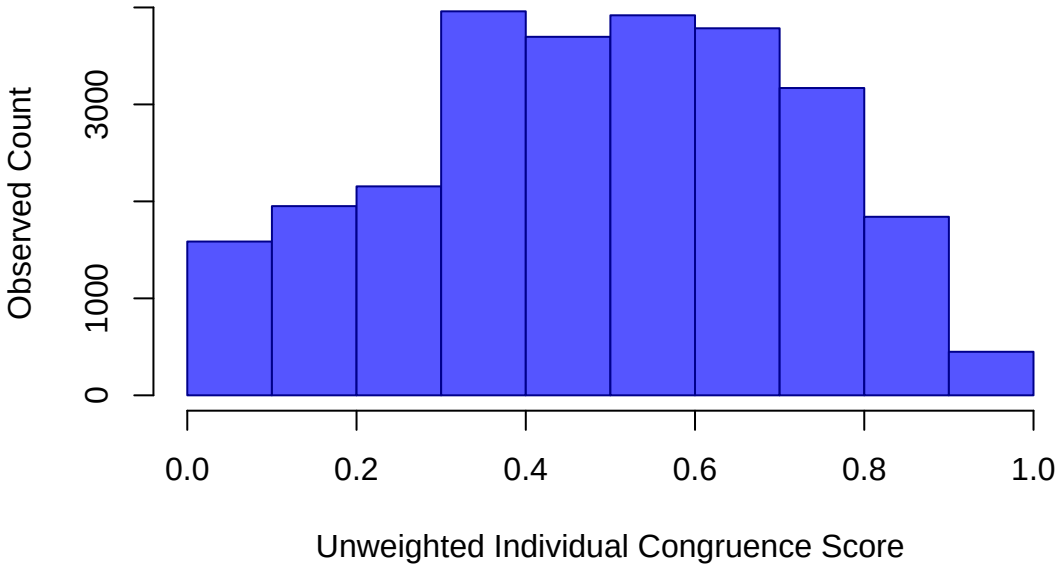

Figure 5.1: Distribution of the Unweighted Individualized Congruence metric, UIC, across selected projects in the GNOME ecosystem

period they were active on a bug was created. The results can be seen in figure 5.1 and figure 5.2.

The UIC results in figure 5.1 follow roughly a normal distribution, although with slightly heavier weighting toward individuals with very low congruences, most likely because of missed communication between individuals in the ecosystem, the previous establishment of development mechanisms that serve as proxies for coordination, such as documentation, or attrition of members from the community.

The results for WIC in figure 5.2 are not nearly as clear. Without the exception of

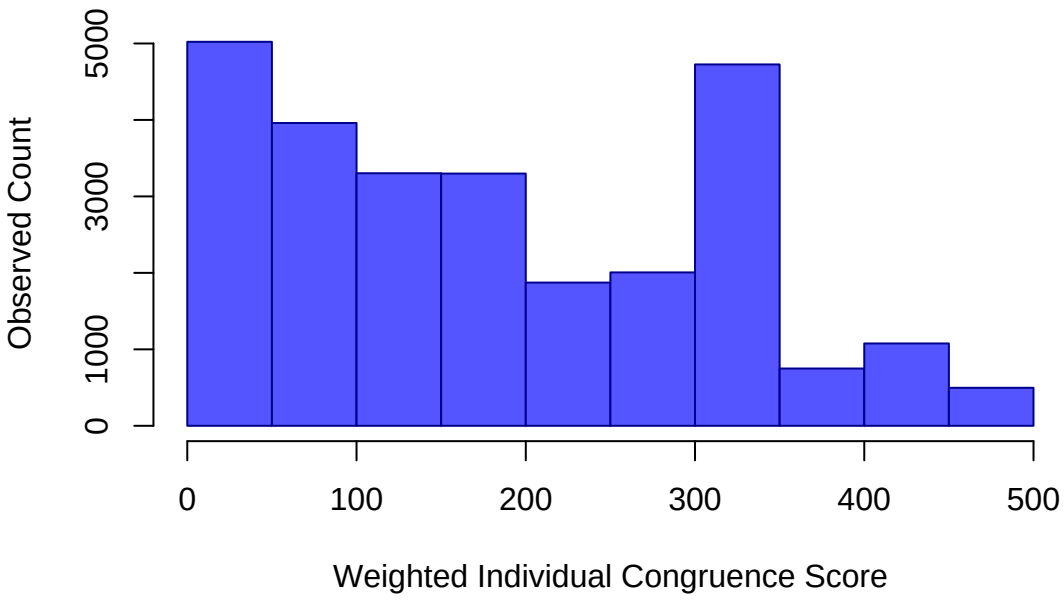

Figure 5.2: Distribution of the Weighted Individualized Congruence metric, WIC, across selected projects in the GNOME ecosystem

a spike in the $300 - 350$ range, the results roughly follow an exponential decay model. Analysis of the abnormal spike shows that it comes primarily from developers working on the Evolution email client, which was primarily developed by a single company with many co-located developers in a manner very similar to a proprietary closed source application. Another interesting aspect of this figure is that not all projects had the same range of scores. Some smaller projects, such as Beagle, never had the coupling between modules that would allow such high scores for WIC. Thus, while the distribution of the values is interesting, it is not comparable across projects in the same that UIC can be compared.

5.4.2 Regression Analysis

To understand the relationship between individual congruence and the time to resolve a bug, a regression model was created. The dependent variable is $\log_2$ of the time to resolve defects in the software as measured in days. Independent variables were the previously described control variables and the mean of the individualized congruence for the developers active on that bug. In practice most bugs had only a single developer working on them (median=1, mean=1.41, max=7), so taking the mean of all developers on a bug should not have a significant impact on the results. The results, as shown in table 5.3 and table 5.4, indicate that the number of developers contributing to the bug and number of people changing the status of the bug both increase the amount of time to resolve software defects, while an increase in the number of comments made on the bug and an increase in the individualized congruence of the developers working on the bug both serve to decrease the amount of

Table 5.3: Simple Regression Using Unweighted Individualized Congruence

Variable	Estimate	Std Error	P-Value
Intercept	1.9707	0.0581	$< .0001$
$numDevs$	0.2846	0.0301	$< .0001$
$deltaPeople$	0.8074	0.0176	$< .0001$
$comments$	-0.0142	0.0036	$< .0001$
UIC	-1.2140	0.0770	$< .0001$
R^2 = 0.134, $DF = 26507$, $p < 0.0001$			

Table 5.4: Simple Regression Using Weighted Individualized Congruence

Variable	Estimate	Std Error	P-Value
Intercept	1.7509	0.0508	$< .0001$
$numDevs$	0.3048	0.0301	$< .0001$
$deltaPeople$	0.7882	0.0175	$< .0001$
$comments$	-0.0142	0.0036	$< .0001$
WIC	-0.0020	1.38×10^{-4}	$< .0001$
R^2 = 0.132, $DF = 26507$, $p < 0.0001$			

time necessary to resolve a bug. In the case of UIC, a developer with high UIC may easily take less than half the time to solve the defect as a developer with very low congruence. Disappointingly, the overall explanatory power for the model is quite low, explaining only approximately 13% of the overall variance. Similar to the original work, the addition of the congruence metric adds about 2% to the R^2 value over a model without congruence and approximately 1% against the model previously shown in table 5.2.

The next step was to break apart the fractional portions of the individualized congruence metric to independently evaluate the relationship between actual coordination and coordination dependencies. This new regression replaces the independent variables of UIC and WIC with a $matchComm_{UIC}$ and $matchComm_{WIC}$, a variable that reflects the numer-

Table 5.5: Regression using unweighted individualized congruence with numerator and denominator separated

Variable	Estimate	Std Error	P-Value
Intercept	1.3944	0.0537	$< .0001$
$numDevs$	0.2639	0.0304	$< .0001$
$deltaPeople$	0.8021	0.1772	$< .0001$
$comments$	-0.0126	0.0036	0.0005
$matchComm_{UIC}$	-0.0634	0.0046	$< .0001$
$coordReq$	0.0331	0.0032	$< .0001$
R^2 = 0.132, $DF = 26506$, $p < 0.0001$			

Table 5.6: Regression using weighted individualized congruence with numerator and denominator separated

Variable	Estimate	Std Error	P-Value
Intercept	1.377	0.0536	$< .0001$
$numDevs$	0.3043	0.0302	$< .0001$
$deltaPeople$	0.7715	0.1775	$< .0001$
$comments$	-0.0123	0.0036	0.0007
$matchComm_{WIC}$	-1.006×10^4	7.960×10^{-6}	$< .0001$
$coordReq$	0.0220	0.0027	$< .0001$
R^2 = 0.131, $DF = 26506$, $p < 0.0001$			

ator of equation 5.4 and equation 5.5. A new variable, $coordReq$ is added that is the total number of coordination requirements, the denominator of the STC ratio. The results of these new regressions can be seen in in table 5.5 and table 5.6.

In this enhanced model it is possible to break congruence apart into the constituent parts of the ratio and that their independent results are still significant. Furthermore, the results are consistent with previous theories and results that propose that defects in highly coupled modules, as shown by high values of $coordReq$, will take a longer time to resolve than

Table 5.7: Regression using unweighted individualized congruence with numerator and denominator separated and extra communication included

Variable	Estimate	Std β	Std Error	P-Value
Intercept	1.4590		0.0568	$< .0001$
$numDevs$	0.2500	0.0560	0.0306	$< .0001$
$deltaPeople$	0.8020	0.3289	0.0177	$< .0001$
$comments$	-0.0125	-0.0224	0.0036	0.0006
$matchComm_{UIC}$	-0.0210	-0.0392	0.0056	$< .0001$
$unmatchComm_{UIC}$	0.0314	0.0572	$< .0001$	
$extraComm$	-0.0119	-0.0264	0.0035	0.0006
R^2 = 0.132, $DF = 26505$, $p < 0.0001$				

defects in less coupled modules. Furthermore, the more communication that an individual has that resolves coordination dependencies the faster the time to resolve the defect. However, of note is that the results do not differ significantly between using the weighted and unweighted models of the metric.

As a final step, we can perform one final regression that takes into account communication by developers that is present in $\mathbf{C_A}$, but has no corresponding edge in $\mathbf{C_R}$, we term these communications as "extra" because they appear to be communications that do not satisfy a coordination requirement. For example, if Alice and Bob communicated during the period of study but they had no coordination requirements then this communication is considered to be extra communication. In table 5.7 the effect of including this extra communication, $extraComm$, is evaluated in a new regression model. Rather than using the amount of coordination requirements in this new model, the number of unmatched communications is included. Functionally, this does little to change the model as $coordReq = matchComm_{UIC} + unmatchComm_{UIC}$.

The addition of the extra communication to the model does little to increase the explanatory power of the model, the R^2 increased by less than 1/1000th of a point. Most of the coefficients for the independent variables remained approximately the same. The most interesting aspect of this regression is that even additional communication that does not directly address coordination requirements has a beneficial impact on the time to resolve bugs, however this effect is much smaller than the effect of matched communication. Table 5.7 also includes the standardized betas for each of the variables in the model. Comparison of the beta values show that not only does matched communication have a stronger impact than extra communication on reducing the time to resolve software defects, it also plays a more significant role in the regression model, although the dominant factors in the time to resolve software defects are the number of people changing the status of the bug and the number of developers working on the bug.

This finding of the differences between matched communication, extra communication, and unmatched communication greatly supports the continued development as the socio-technical congruence family of metrics as it has, for the first time, isolated the differences between communication across coordination dependencies and communication without regard to coordination dependencies.

5.4.3 Utilization of Individualized Congruence

While WIC corresponds to the logical concept of rewarding individuals for communicating across links that satisfy many coordination dependencies, in practice, the differences between UIC and WIC are rather small, and in no place do they exhibit opposite results. This small difference suggests that in most cases it makes little difference from an accuracy and results standpoint whether edges in the congruence networks are weighted or not, therefore for the remainder of this work I utilize only those congruence networks that are unweighted.

5.5 Metric Stability

Although software engineering teams typically stay together for extended periods, turnover of developers and changes in architectures can lead to decay in team structure and task dependencies[25, 72, 81]. To validate the stability of the STC metric and address questions regarding the structure of the task dependency network, $\mathbf{T_D}$, a large scale sensitivity analysis was performed along multiple different variables.

The first way the metric was tested was through the introduction of a decay factor applied to the $\mathbf{C_A}$, $\mathbf{T_A}$, and $\mathbf{T_D}$ networks. This factor, δ was scaled between 0.8 and 1.0, where 1.0 indicates no distortion and can be seen in equation 5.6 for $\mathbf{T_D}$. It was likewise mapped for $\mathbf{C_A}$ and $\mathbf{T_A}$. In this way older dependencies, assignments, and communica-

tions are slowly removed from the network over time.

$$T_{D_i} = \sum_{j=0}^{i} \delta^{i-j} t_{D_j} \tag{5.6}$$

To account for possible errors in obtaining the actual communication networks, $\mathbf{C_A}$, a parameter sweep introducing random errors into the network was performed. Errors were modeled as simple errors of commission and omission. $ErrOm$, which resulted from leaving a communication link out of the network when it should have been present and errors of commission, $ErrCom$, which resulted from inserting a link into the communication network where in fact none existed. Care was taken to ensure that the networks remained symmetric after distortion. As the networks of interest were frequently large and therefore rather sparse, the distortion factor was based on a proportion of the existing links. For example, assume a network with 20 agents (190 possible links in a symmetric network) in which there are currently 40 links between individuals. If the distortion algorithm was given a 10% error of omission and a 20% error of commission then on average 4 links would be added to the network to represent edges there were erroneously omitted from the original network (error of omission) and 8 links would be removed that were wrongly added to the network (error of commission). Edges which are removed and added are chosen randomly and there is no account made for network structure when dealing with errors of omission.

To address issues of task dependency network, $\mathbf{T_D}$, generation and better understand

Table 5.8: Variables Modified to Test Network Stability. 100 simulations were done on each point in a full parameter sweep, resulting in 128,000 simulations per project.

Parameter	Start	Stop	Delta
Decay	0.80	1.00	0.05
$ErrOm$	0.00	0.30	0.02
$ErrCom$	0.00	0.30	0.02

how the metric can be deployed in real world software development tools, two different formulations for $\mathbf{T_D}$ were tested. The first formulation utilized a view that worked on the premise of complete information and provided a static network based on the sum of the task dependency from all periods in the sample, both forward and backward. Such a formulation was used in the original work on Socio-Technical Congruence and is appropriate for retrospective studies. Here, such a formulation of $\mathbf{T_D}$ is called "Complete". The second formulation utilized only task dependencies that occurred before that time, a mechanism that is suitable for development of tools to assist software engineers and managers. Such a formulation is called "Progressive". Both formulations of $\mathbf{T_D}$ were subject to the decay previously discussed.

A full parameter sweep was done over all parameters utilizing the exiting data and networks from the GNOME project. The parameters for the sweep can be seen in table 5.8. On a monthly basis overall congruence and individualized congruence for project participants was calculated. This analysis allows the evaluation of sensitivity of the metric to decay, formulation of the task dependency matrix, and errors in creation of the actual communication network and any combinations thereof.

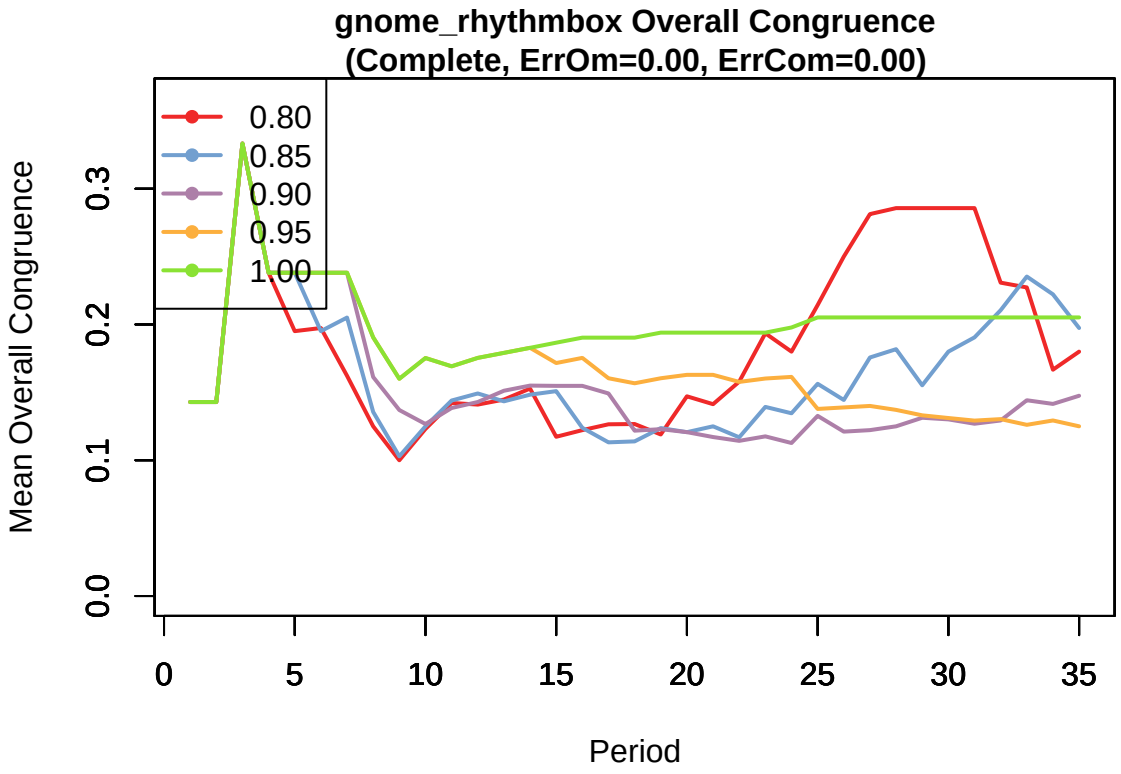

Figure 5.3: Overall network congruence for the project "Rhythmbox" using the complete formulation of $\mathbf{T_D}$ and no errors in the network. Note how additional decay produces higher congruence. Each time period is one month.

5.5.1 Decay In Socio-Technical Congruence

As anticipated, implementing a decay factor in STC brings additional insight into organizational structure and work patters when using STC. In figure 5.3, we see that increasing the decay factor for the project "Rhythmbox" typically increases overall congruence. In contrast, figure 5.4, shows that increasing the decay factor for the project "Beagle" typically decreases the overall congruence for the project.

To understand why these seemingly different results can occur, it is important to un-

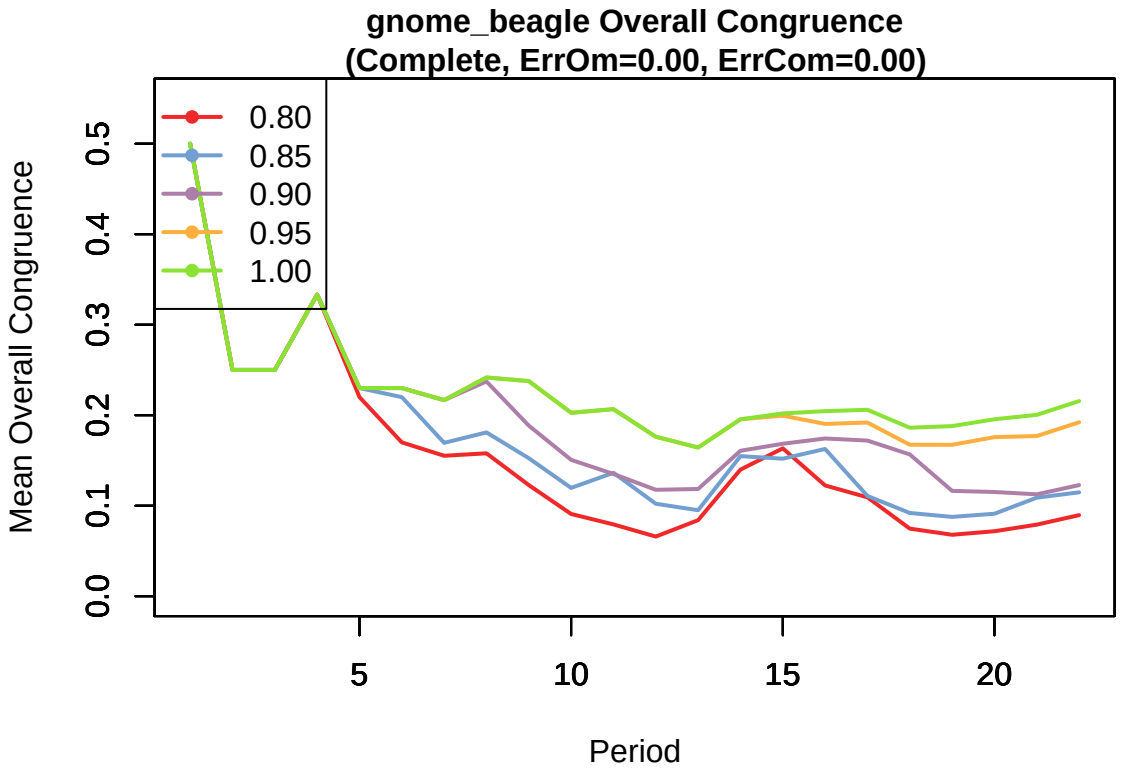

Figure 5.4: Overall network congruence for the project "Beagle" using the complete formulation of $\mathbf{T_D}$ and no errors in the network. Note how additional decay produces lower congruence. Each time period is one month.

derstand what networks the decay function was acting upon. Were decay applied only to the cumulative actual coordination network, $\mathbf{C_A}$, there would be no way that overall congruence could increase as more decay was applied. However, because both the task dependency network, $\mathbf{T_D}$, and task assignment network, $\mathbf{T_A}$ are decayed along with actual coordination, it is possible for congruence to rise because of decay, particularly in the case where either the task dependencies or task assignments experience a heavy amount of churn. Which is the situation that occurs in the Rhythmbox project.

Rhythmbox is an application written in the C programming language for listening to and organizing music. Although there are ways within C to create modular code, most projects, Rhythmbox included, do not take advantage of such methods. For this analysis, I had about four years of project history, during which time the project went from the casual project of a single developer to the standard music player tool for the GNOME desktop. However, during the evolution of Rhythmbox the leadership of the project also changed as the original developer left the project and allowed other developers to take over the project and begin to re-architect the project. This change affected not only the social network of the project, as the original lead programmer was no longer involved, but also the technical networks of the project. A key component of the task assignment network was removed and the re-architecture proposed by the new project leads involved creating a new set of technical dependencies. This occurred around period 15 in the project's life. It is shortly after this period that we see congruence begin to increase in the model with the highest decay, 0.80, precisely because the older dependencies in $\mathbf{T_D}$, which are no longer reflected

in actual code, have since waned and are no longer relevant.

In contrast, the structure of Beagle, a file indexing program written in C#, a programming language that makes it easy to create highly modular programs, shows that applying greater decay to the networks results in lower overall congruence. In contrast to Rhythmbox, Beagle has a a consistent modular structure that was created in the beginning of the project and never changed during the course of the observation period. The primary team of developers also remained constant. In this case, task dependencies were renewed throughout the life of the project, but the coordination around those dependencies, which were largely static, was not renewed because the dependencies had shown little change.

This shows in the case of long running projects where the architecture and project membership may change over time, the use of a decay factor in calculating STC is greatly beneficial for calculating a more realistic measure of how the organization reacts to changes in the code structure. In the case of consistent teams and code structures, however, the addition of a decay function appears to do little other than deflate the score for congruence in a rather predictable given. These results were also seen in several other projects within GNOME that are not detailed here. This is a key finding for the development of interactive tools for managing software development that utilize STC, as larger projects often span multiple years with varying teams and possibly varying architectures, especially between release cycles. This stands in contrast to the original work on STC which looked only at the congruence of the project in shorter periods around release management and did not take into account the full history of the project[15].

5.5.2 Network Formulation

A comparison between the complete and progressive formulations of $\mathbf{T_D}$ found that in most cases there is little difference between the complete and progressive formulations of $\mathbf{T_D}$. In particular, as a project progresses further congruence calculated with both metrics will converge as the progressive formulation of $\mathbf{T_D}$ gets closer to the complete formulation of $\mathbf{T_D}$. Continuing the examination of the Rhythmbox and Beagle projects, the results of these networks using the progressive formulation of $\mathbf{T_D}$ can be seen in figure 5.5 and figure 5.6 respectively.

STC when calculated with the progressive formulation is very similar to the results found when calculating STC with the complete formulation of $\mathbf{T_D}$ in the previous section. In fact, for the situation where no decay is applied to the networks in the model, both formulations of $\mathbf{T_D}$ produce identical results after time period 15. As the amount of decay applied increases, so does the amount by which the progressive formulation exceeds the complete formulation. The differences between the formulations can be seen in figure 5.7 and figure 5.8.

The combination of applying a decay factor and using a progressive formulation of $\mathbf{T_D}$ is most visible in Rhythmbox when there is a high decay factor. The benefits of the progressive formulation also begin to dramatically accelerate around period 15, which, as previously described, is the point in the project history when project management was changed and an effort to re-architect the project source code began. From this point on the

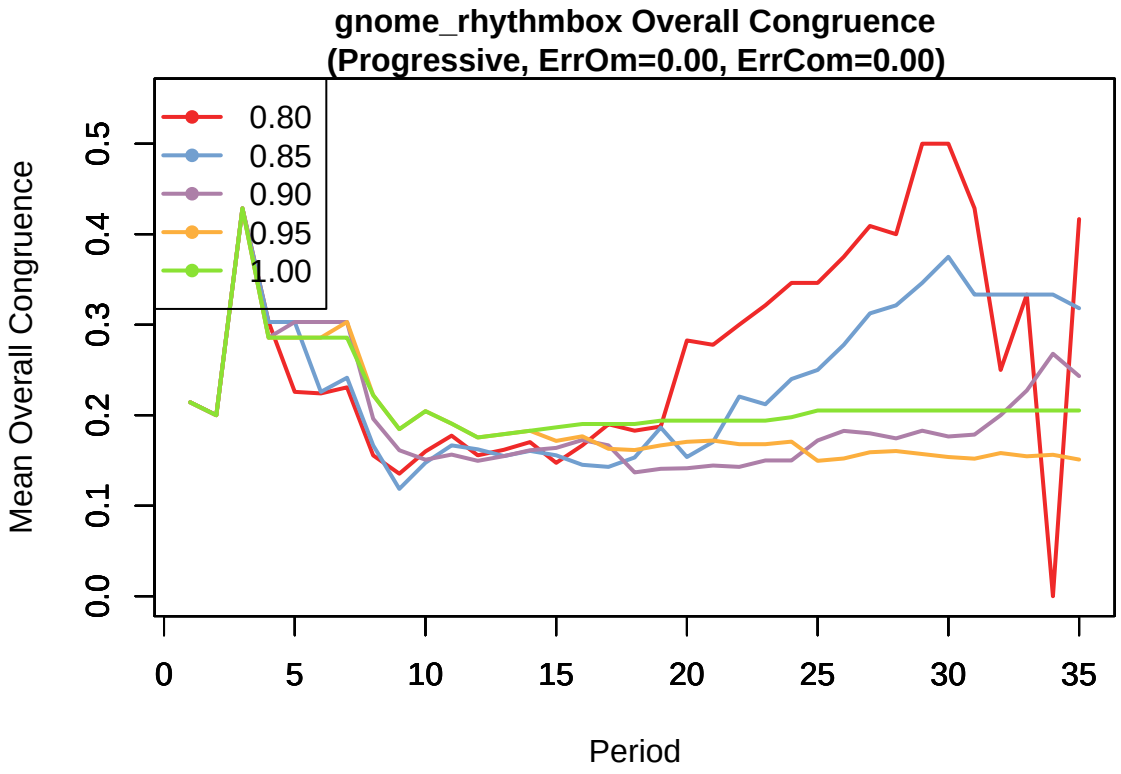

Figure 5.5: Overall network congruence for the project "Rhythmbox" using the progressive formulation of T_D and no errors in the network. Each period is one month.

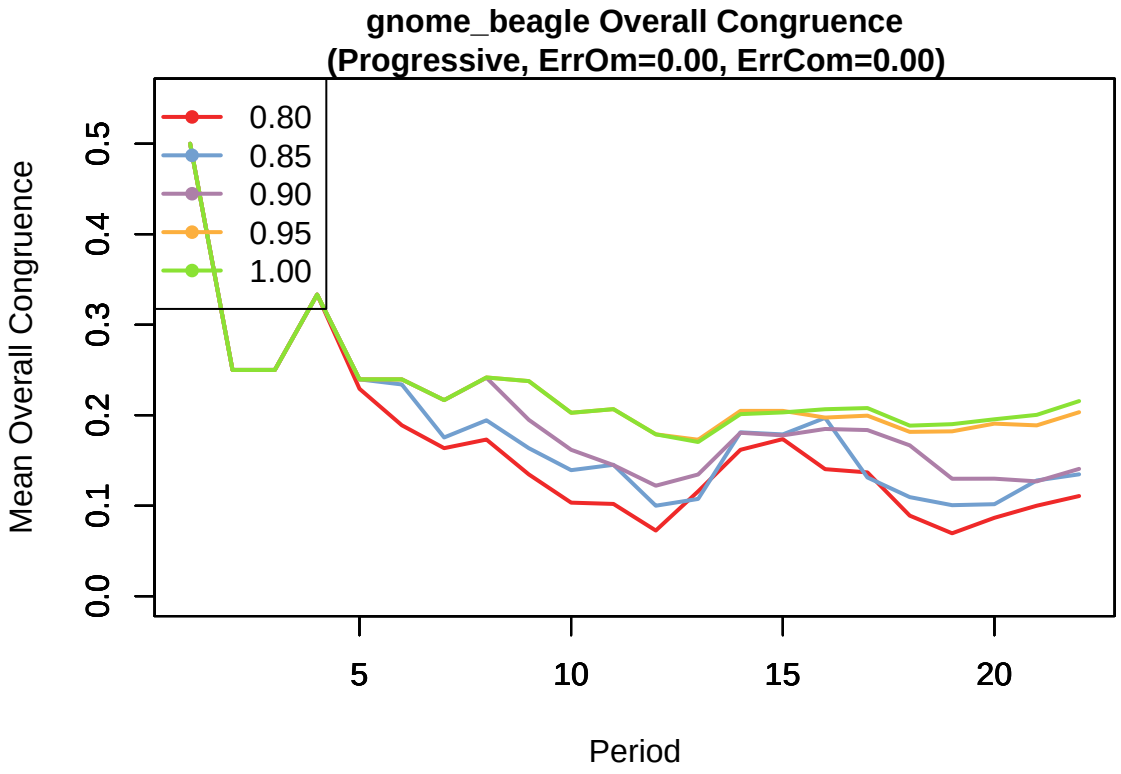

Figure 5.6: Overall network congruence for the project "Beagle" using the progressive formulation of T_D and no errors in the network. Each period is one month.

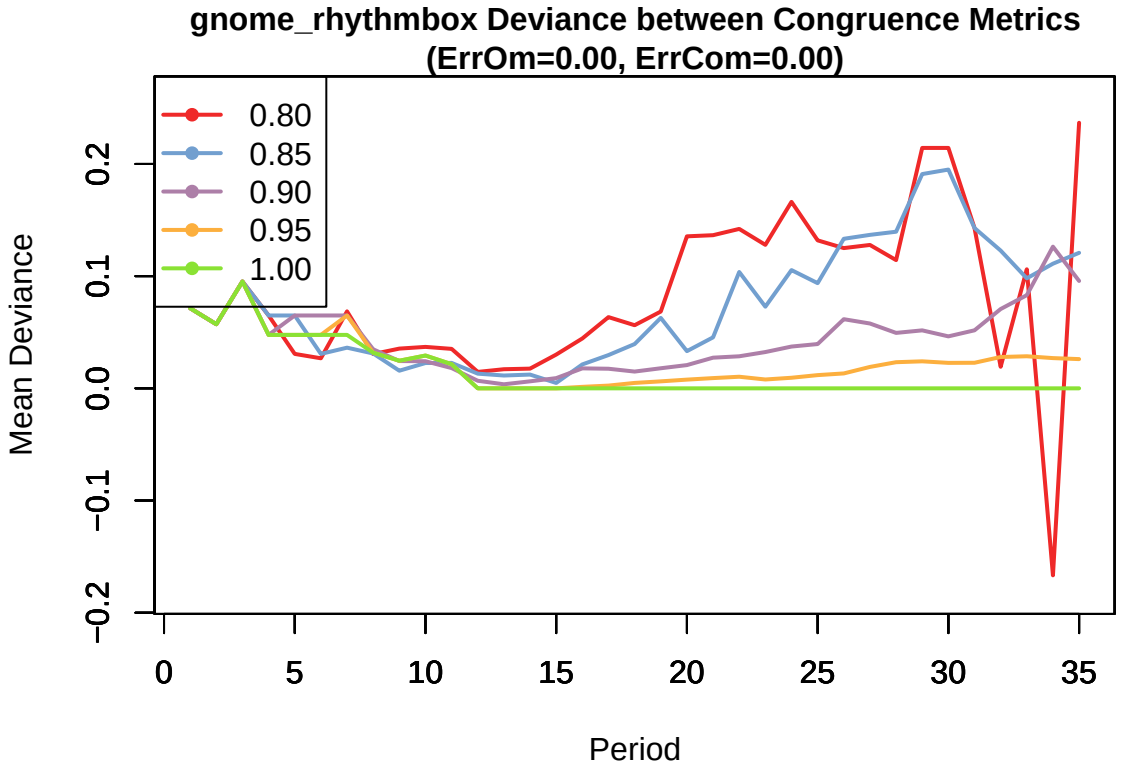

Figure 5.7: Difference between progressive and complete formulations of T_D for "Rhythmbox". Each period is one month.

deviance between the two metrics continues to increase until time period 30, at which point a sharp drop in STC is observed and at time 34 congruence for the progressive formulation of $\mathbf{T_D}$ with a 0.80 decay drops to 0.

Examining the project during this time period yields that month 34 was a time of very little communication for the project. In particular, most of the primary developers of the project were not highly active during that month because of a conference and travel. However, that is not to say that they were not active on the project. Most of the conferences within the GNOME community provide a single room where developers can take advan-

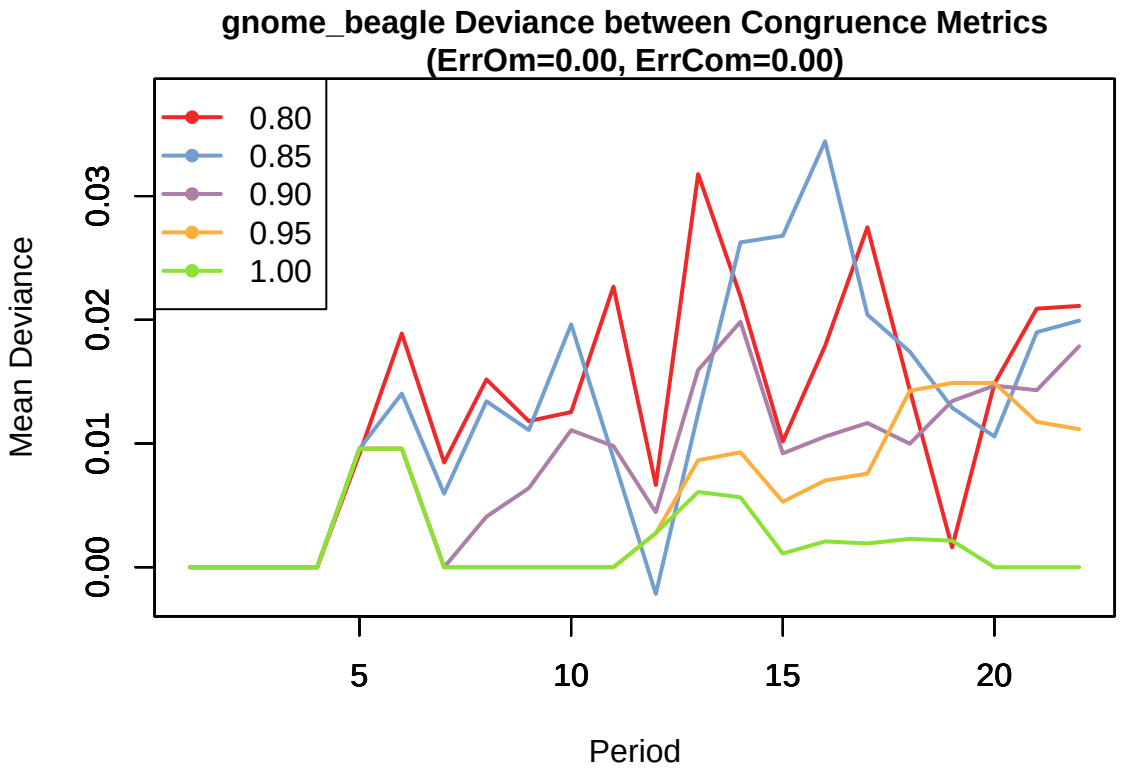

Figure 5.8: Difference between progressive and complete formulations of T_D for "Beagle". Each period is one month.

tage of radical co-location and make rapid progress on the software[107]. This is what happened in this case, the developers were working on new features of the software, which were beginning to be captured in the dependency network at that time and the next time period, but there was little captured actual communication related to it because the primary mode of communication switched from computer mediated tools to face-to-face.

This shows one possible vulnerability of utilizing a progressive formulation of the T_D network, rapid changes that are discussed slightly out of sync with the time window of interest can cause dramatic drops in congruence. However, it appears that this is rather an edge case as it did not exhibit itself on any other projects within the study set.

5.5.3 Errors In Communication Network

The largest part of the simulation pertained to establishing the stability of the congruence metric in the face of noise in the actual coordination network. For each project at each time period a fitness landscape was produced showing the average network level congruence across 100 permutations of the network. Figure 5.9 shows the landscape generated by one of these runs for the Rhythmbox project at period 28, an example which best highlights the differences between the progressive and complete formulations of the task dependency matrix. Please note that no distortion is in the right of each graph, and maximum distortion is on the left of the graph. Moving forward visually increases the rate of errors or commission, while moving to the left increase the rate of errors of omission.

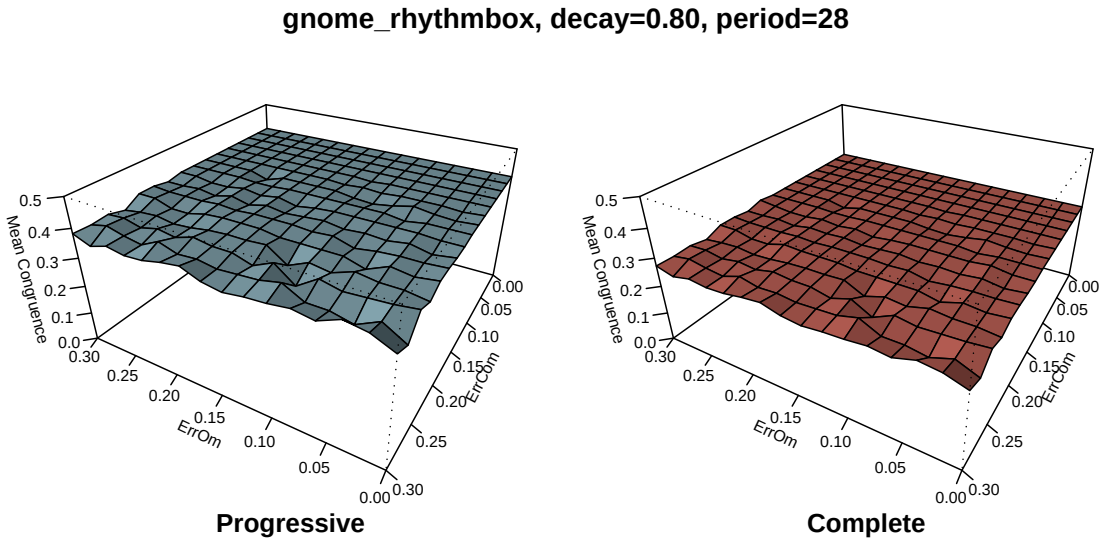

Figure 5.9: Landscape of "Rhythmbox" with 0.80 decay at period 28

The networks were most noisy during the first few periods of the project, often swinging wildly, but almost never going above the value of congruence provided by the network with no noise. Figure 5.10 shows the landscape for the Beagle project in the first period of the project's history. Also, as is typical, an increase in the rate of errors of omission had no effect on the overall network congruence.

A regression model was used to identify the relationship between the error levels and congruence. The dependent variable was the congruence of the perturbed network and the independent variables were the decay factor, $ErrOm$, $ErrCom$, and the congruence of the unperturbed network, as shown in equation 5.7. The results of this regression can be seen in table 5.9 and table 5.10 for the complete and progressive formulations of T_D respectively.

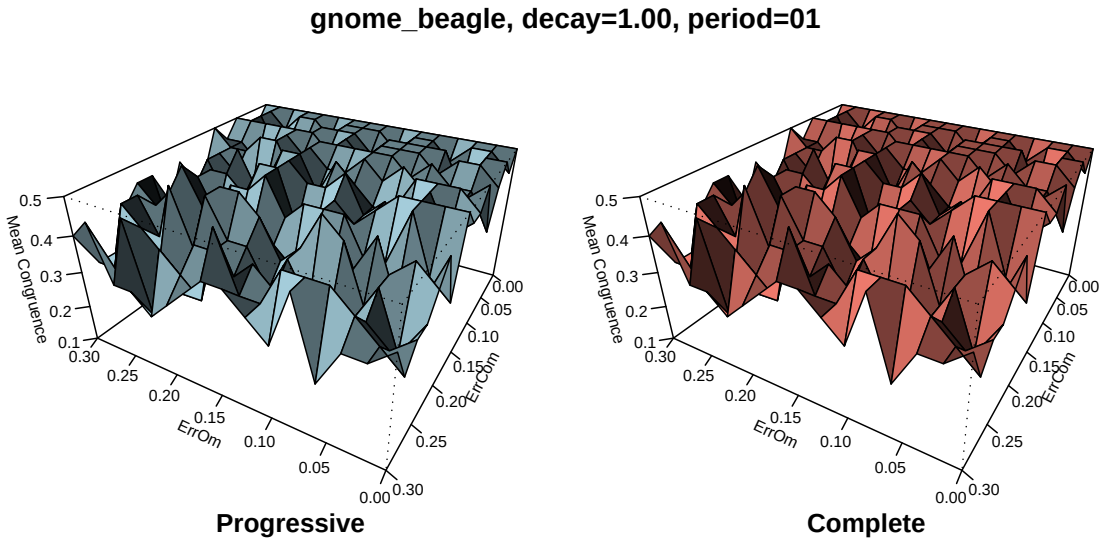

Figure 5.10: Landscape of "Beagle" with no decay at period 1

$$STC_{Perturbed} = \beta_0 + \beta_1 Decay + \beta_2 ErrOm + \beta_3 ErrCom + \beta_4 STC_{Base} + \epsilon \quad (5.7)$$

The results clearly show that the metric is fairly resilient to random errors of omission

Table 5.9: Relation Between Congruence Using Complete T_D Formulation With Error and Unperturbed Network

Variable	Estimate	Std Error	P-Value
Intercept	-0.0190	0.0013	$< .0001$
Decay	0.0541	0.0015	$< .0001$
$ErrOm$	0.0004	0.0011	0.668
$ErrCom$	-0.1440	0.0011	$< .0001$
Congruence	0.8499	0.0012	$< .0001$
R^2 = 0.9553, $DF = 28155$, $p < 0.0001$			

Table 5.10: Relation Between Congruence Using Progressive T_D Formulation With Error and Unperturbed Network

Variable	Estimate	Std Error	P-Value
Intercept	-0.0179	0.0013	$< .0001$
Decay	0.0543	0.0015	$< .0001$
$ErrOm$	0.0004	0.0011	0.742
$ErrCom$	-0.1515	0.0011	$< .0001$
Congruence	0.8502	0.0012	$< .0001$
R^2 = 0.9512, $DF = 28155$, $p < 0.0001$			

within the network. Higher levels of errors of commission tend to decrease the overall congruence of the organization, as this action removes edges from the actual coordination network. Furthermore, systems with less decay tend to exhibit slightly higher congruence, however, this is a very small factor that within the actual data would cause the overall STC of the network to increment only 0.01 from a 0.8 decay to a 1.0 decay (no decay).

5.5.4 Possible Faults

The generation of noise in the networks was done on a purely random basis. While this random effect takes into account the network structure when removing links through errors of commission, errors of omission are treated as a purely random phenomena, with no particular attention paid to the location of the links outside of the random selection. To perform a truly random analysis, the structure of the network would need to be taken into account.

However, this effect is somewhat mitigated by the fact that within the data set, the

chance of entirely missing an individual is low. Of the individuals with coordination requirements, less than 1% of those individuals had no links at all in the actual coordination network. The work process of the GNOME community also helps to reinforce the strength of this claim, by encouraging all developers to collaborate on project mailing lists and discuss bugs on the project bug tracker this leads to fewer than 2% of the individuals with coordination requirements not having communication found in either the bug tracker or project mailing lists. This greatly limits the chance that an individual would be completely left out of the network.

5.6 Discussion

This chapter has extensions and contributions to the continued study and use of socio-technical congruence as a tool for understanding team performance. Firstly, it was able to successfully replicate the original results in an open source context. This is valuable because few environments are as controlled as was the original study on STC. By showing its validity in an Open Source context this provides additional incentive for tool builders to create suites of tools for developers and project managers based on STC. It is not necessary to devote an individual to work for weeks to unify the data, or to have an organization that is CMM Level 5, to get useful results. Rather, using the data from an open environment can provide similar valuable insights.

To better understand exactly how STC works, I also proposed breaking the ratio apart

and showed that the presence of "extra" communication is also related to increased performance. This indicates that in most cases, more communication is good, although there most likely is a point diminishing returns from team members pending too much time communicating. The evidence suggests that rather than creating tools that rely solely on the STC metric as a ratio, incorporating matched communication, coordination requirements, and extra communication as elements may prove beneficial.

This chapter has introduced the concept of decay to STC and found that it is most useful on long lived projects when a substantial portion of the team leaves or there is a major change in architecture. However, the value to use for decay is not yet clear and most likely depends on team work processes and the window size for generating the metrics. Clearly this is an area ripe for future study.

Next, I addressed the issues of the structuring of the task dependency network, $\mathbf{T_D}$, and whether it should be a purely backwards looking network or it should include all data as there may be dependencies in the network that have not yet manifested themselves. For many projects there is very little difference between the values generated using the different formulations, especially in a stable project. This finding indicates that tools developed that calculate STC on a rolling basis as development proceeds, which necessitates the use of a progressive $\mathbf{T_D}$, should produce statistically significant results.

This work has also addressed stability issues related to STC and concluded that stability of the metrics is not a major issue. In particular, when assuming even up to 30% noise in the system the deviation of the metrics generally remained low. Furthermore, one

of the primary concerns was that an individual may be completely missed in the course of collecting data about actual coordination, $\mathbf{C_A}$. Within the GNOME Open Source community this was found to not be a concern as there were few times when individuals who had high coordination requirements were completely missing from the network. However, we note that narrowing the window of analysis too much will lead to cases where individuals are missed because of personal reasons, such as vacation, illness, or work schedules, and introduce error into the use of STC as a viable metric for software developers.

Based on these findings, I recommend that STC be calculated on a rolling basis by breaking the metric apart into it's constituent parts of matched communication, coordination requirements, and extra communication. Furthermore, a small decay factor should be applied to the networks to account for changes in project structure and team membership.

Chapter 6

Conclusions

Although the term "Open Source" is only 11 years old, it has made dramatic impacts on the field of software engineering and on commercial software development. What first started as a social movement has shown itself to be a robust way to develop software in commercial environments. In this thesis I built upon previous work that examined individual interactions in single Open Source projects by expanding and evaluating interactions within Open Source ecosystems at multiple levels: foundations, firms, and individuals.

6.1 Contributions

I began with a thorough examination of the ecosystem of commercial firms around the Eclipse integrated development environment and how the non-profit Eclipse Foundation

helps to drive value for those firms. Key properties identified were the non-market nature of the Eclipse foundation, which allows firms to focus on their specialties with less of a worry that the main project driver will implement similar features and destroy their market. The introduction of process and the joint marketing efforts that the foundation can put forth were also highlighted as key factors driving success. Finally, the structure of the Eclipse ecosystem is such that Eclipse is regarded as a platform, rather than a single tool. This allows firms to quickly innovate in new and radical ways without having to master all aspects of the ecosystem.

In chapter 3 I examined the interactions of the firms in the Eclipse ecosystem as they collaborate through code artifacts. This found that despite the fact that although IBM is no longer the dominant player in Eclipse from a legal perspective, it still is the dominant player when it comes to developing and contributing code to Eclipse. In particular, IBM plays key roles in the development of the reusable platform components of Eclipse. This over dependence on IBM introduces some weaknesses into the ecosystem, as was highlighted when IBM recused itself of internationalization for Eclipse and it several releases were issued without internationalization. This pattern is significantly different than the pattern that was seen in volunteer-founded GNOME ecosystem and, while lacking as much corporate support, has a more diverse set of firms contributing to the core portions of the platform.

Next, a study of the relationship between firms and volunteers in an Open Source community found that the general presence of commercial developers in volunteer communities

has no statistically significant relationship to the ability of projects to attract additional volunteer developers. However, when the firms are classified by the nature of their business within the community, divergent effects are found. The presence of developers working for firms that package the complete output of the community, dubbed community focused firms, is related to an increase in the amount of volunteer developers at a later time. In contrast, the presence of developers working for firms that focus only on niche projects within the community, known as product focused firms, is related to a decrease in the amount of volunteer developers working on the project at a later time.

Finally, in chapter 5 I examined the patterns of individual communication and coordination requirements. This section used an Open Source community to successfully reproduce the earlier results that Cataldo et. al. found in a commercial software environment[15]. This allowed the expansion of the socio-technical congruence metric and found that communication that matches coordination requirements has a very strong impact on time to resolve software defects and, perhaps more importantly, even communication which does not line up with coordination requirements has a beneficial impact on the time to resolve software defects, although to a lesser degree than communication that matches coordination dependencies. A key implication of this finding is that it allows the differentiation between individuals who communicate across coordination requirements and those who merely communicate a lot – providing a powerful validation for the concept of socio-technical congruence. The work on the metric was further expanded to propose a decay parameter to account for changing project membership, dependencies between tasks, and

shifting patterns of communication. Finally, analysis of the metric with respect to noise in the data found that random errors in the observed networks caused only very slight variations in the value and significance of STC.

These findings all contribute to the knowledge of Open Source ecosystems as a new method for collaboration across firms and markets. In the remaining this chapter I provide a set of recommendations for individuals, firms, foundations, and community designers to better build communities that function in these complex ecosystems that see volunteers, multiple firms, and foundations all working together toward a shared goal.

6.2 Recommendations

This thesis examined participation in Open Source projects at three different levels, foundation, firm, and individual. Here I make recommendations to participants at each of those levels while expanding the definition of foundation to include any group that may wish to foster the creation of an Open Source community.

6.2.1 Recommendations for Individuals

Recommendation 1 *Lower your ideological goals to embrace and work with commercial developers*

Many individuals within Open Source ecosystems are motivated by ideological goals. Particularly in Europe it is common for developers to express a desire to work in a completely free (as in liberty) software environment[65]. Such a desire often leads to distrust of commercial firms in the community and creates a disconnect between the users of the software who often desire the Open Source because of it's low cost and the ideological developers[23]. In chapter 4, however, it was shown that commercial firms had typically had a positive impact on the number of volunteers participating on a project, indicating that at least some of the concerns of ideological developers regarding commercial participation may be unfounded.

Recommendation 2 *Focus communication to address coordination dependencies to maximize the benefit of communication*

From the perspective of individual developers attempting to work in an Open Source community, this research suggests that with respect to the time to resolve software defects more communication is almost always beneficial. While communication that satisfies coordination dependencies is most helpful, communication that is unmatched with coordination dependencies also decreases the time to resolve software defects. This may be because some of the communication that does not match dependencies serves to update all members of the community about the developer's current status and how future dependencies may be resolved. Given limited time and attention resources of developers, focusing on meeting individual coordination requirements will have a greater impact than general communication.

6.2.2 Recommendations for Commercial Firms

Recommendation 3 *A perceived loss of control over technology is rarely a sufficient reason to avoid participating in Open Source because it is possible to participate without giving up control*

Although the state of commercial involvement in Open Source has matured significantly in recent years, there still are many ways in which this thesis can guide firms as they participate in Open Source communities. While tech giants such as Intel are frequently members of multiple Open Source communities, there still exists numerous firms that are hesitant to participate in Open Source at any level. This research has demonstrated that although joining an Open Source community means a change in work practices to match that of the community, it does not mean giving up control of components or technologies to other firms. In reality, most projects within Eclipse are managed by a single firm which is able to guide and develop the technology according to internal roadmaps while leveraging the benefits of the Open Source platform.

The major downside to participating in an Open Source community is that it requires publication of the project source code. In certain cases this may be untenable due to external requirements from customers and clients that the source code remain proprietary. In these cases, although the firm still would maintain all rights to the software, Open Source may not be a feasible strategy.

Recommendation 4 *When entering an existing community with volunteer developers, firms*

should survey the community to ensure maximum compatibility and that the partnership is beneficial for all sides

Firms cannot blindly enter into an Open Source community and expect that they will be welcomed with open arms. On the contrary, firms that wish to attract additional individuals to their project should be heedful of the scope and methods they use to interact in the community. For example, releasing a project as Open Source does not immediately attract new developers and contributors. In many cases providing a "code dump" to an Open Source community with little context will cause community members to be hesitant about the commitment to maintain the code and work with the community in the future. Greater interaction with the community as a whole, which is the pattern of community focused firms, will lead to additional participation from community members. Such participation, however, has cost as members of the firm must still work their way into the meritocracy of projects.

This suggests that there may be cases when it is beneficial for a firm to create their own fork of a project and not contribute directly back to the community (provided the license allows it). An example of such a situation is a product focused firm that intends to provide a specific commercial application of the project that feels it may cause more confusion and do more harm to the community by participating than by maintaining their own source code tree.

Recommendation 5 *Firms entering an Open Source ecosystem need not completely re-*

orient their business model to participate

When Netscape embraced Open Source in 1998 it was widely seen as a last ditch effort to fight back against Microsoft's rising dominance. When they released the source of their flagship product as Open Source, Netscape effectively bet their entire business on a radical change. History has shown they lost the bet. Within Eclipse, there are numerous firms that are part of the Eclipse community, but release only small amounts of code as Open Source. Furthermore, many of these firms have little need to collaborate with other firms in the process of developing their product. Often times these firms use participation in the Eclipse ecosystem as a way to leverage additional resources, but offering a small amount of their own technology, they're able to use and guide a much larger amount of technology.

It is also incorrect to assume that a firm can choose to participate in an Open Source ecosystem or create an Open Source project based on previously proprietary technology and not have to alter its business model. Firms need to be intimately aware of the desire of community members to have real impact on the design and implementation of the software. Firms must be willing to dedicate additional resources to activities that have little immediately quantifiable benefit, such as developer time, to building and supporting the community. This necessitates that many firms change how they account for developer time by acknowledging the social side of building and participating in an Open Source community.

6.2.3 Recommendations for Foundations and Community Designers

Recommendation 6 *Forward thinking modular architectures should be used to promote innovation by community members with minimal overhead*

Individuals involved in the Eclipse community consistently extolled the virtues of the highly modular architecture of the Eclipse ecosystem. This architecture allowed firms to easily build tools without necessitating a complete knowledge of the architecture and intricate details of the implementation of key components. Modular architectures also reduce the amount of communication and coordination necessary as developers can treat substantial components as black boxes. In addition to fostering development at the core of the project, the modular architecture of Eclipse was seen as supporting radical innovation, including efforts to bring Eclipse technologies to the server and web based engines.

In the broader context of loosely federated distributed systems a service oriented architecture (SOA) can play a similar role as the modular architecture of Eclipse. However, like a modular architecture, care must be taken to ensure that individual components are properly documented, tested, and verified. Although modular architectures, such as Eclipse, saw contributions in many projects centered around a single firm, most prominent projects had some contributions from multiple firms. Within a SOA based architecture where components are developed and hosted by disparate entities, this may not be possible and therefore may limit the overall success of the community.

Recommendation 7 *When building an Open Source community, give the community con-*

trol over the community

One of the key issues arising from the analysis of the Eclipse platform is that a substantial portion of the project code was written by a single firm, IBM. Within the Eclipse ecosystem there has, to this point, been little dissent with regards to IBM having a high amount of control over the platform, however this is not true across Open Source ecosystems. For example, the tight control that Sun MicroSystems exerts over core portions of OpenOffice.org has led to fragmentation and dissent in the community[88]. Part of the success of Eclipse in this regard may be expressly because IBM, although maintaining de facto control through their contributions to the Eclipse platform, is still beholden to the larger community of firms in the Eclipse ecosystem through the work of the Eclipse Foundation.

This is a bold step for a community founder to undertake as it requires giving up additional rights to the intellectual property and giving others a major stake in project management, leaving open the possibility that the goals of the project will diverge from the founder's goals. In practice, with regards to the Eclipse Foundation the goals of the community have expanded and varied from IBM's original goals of creating an extensible IDE. However, this is not a case of the size and prominence of IBM's interests getting smaller, rather the entire ecosystem has grown, making room for firms with innovative new business models to participate. The nature of Open Source also provides additional protections for a community founder as they can choose to retain rights to their original code, providing security that original contributions can never be taken from contributors without their consent.

Recommendation 8 *Recognize that some components of an ecosystem will be dominated by single firms and plan accordingly*

The centralization of the Eclipse platform around a single firm, and the focus of many firms on only a handful of projects illustrates a fundamental problem with many communities: when firms focus on and contribute primarily to their areas of expertise, it is difficult to get contributions to core technologies that all firms build upon, but are commonly regarded as a commodity. Even GNOME, which saw many firms contributing to the core technologies of the GTK+ widget set, still experienced some centralization. The initial design for the GTK+ widget set was done by a small group of developers who sought to create a robust widget set for a paint program. Since that point, much of the work on the widget set has been undertaken by Red Hat, and it is only the social norms of the community, which is much more open and amenable to individuals contributing to multiple projects, that has allowed so many firms to contribute to the core technologies. Yet, the contributions of other firms are still dwarfed by that of Red Hat.

From the perspective of a firm or institution choosing to participate in an ecosystem, adoption of components developed primarily by an external entity may limit the ability to direct some aspects of their own project. For example, a firm that builds an application using the Eclipse Rich Client Platform may have little control over how the platform and base user interface components evolve and may create licensing implications for the tool. Such concerns bear little difference to those concerns needed when evaluating proprietary toolkits, with the major differences being the lesser cost and frequently greater access of

utilizing tools from an Open Source ecosystem.

Recommendation 9 *In communities without a dominant market player additional incentives may be needed to develop some key components*

This centralization issue highlights a fundamental challenge in building platform infrastructure software, unless a single firm stands to obtain a disproportionately large benefit from the code, it may be difficult to find firms willing to invest their resources for development. While the Eclipse platform has been successful because of IBM's utilization of core technologies in many products from Lotus, there are numerous examples where platform infrastructure projects end up as too customized or underdeveloped. In particular, in the field of scientific computing, there are frequently scores of programs that all provide a small portion of the functionality needed, but many often require expansion. For example, there exists at least fourteen different java based libraries for visualizing social networks. While a handful of the libraries share code, many reimplement functionality present in other projects, such as the code to load and save network data sets. This reimplementation in addition to being an unnecessary expenditure of time undoubtedly introduces bugs into the software and leads to incompatibilities.

In the context of scientific software, which often has little commercial value and for which the authors frequently receive little credit, but is valuable to many scientific projects, providing additional incentives for projects to Open Source their software could prove a boon for scientific research. For example, the National Science Foundation could provide

additional funding to projects developing social network analysis tools. These funds would be specifically designated to support building a community around the code, much in the same way that the Eclipse Foundation has taken a strategic approach to building a community. While such a strategy would not completely resolve all differences between projects, by providing a supported community an ecosystem for research could easily be created.

Such a system would differ significantly from the way that most academic software is currently released as Open Source, where support is only available from already stressed academics and the communities are typically very small. Indeed, the willingness to dedicate employees to manage the Open Source community was previously shown to have significant benefits for research on Java virtual machines in the Jikes RVM project[1]. As an added benefit, the availability of source code and the support necessary to compile and run the source code may assist in replication of results from experiments. While there have been some efforts in this direction, most notably from the United Kingdom's funding from OMII-UK's MyExperiment and Taverna projects[84], there has, to this point, been little funding from the United States government for such work.

6.3 Future Work

While this thesis has made significant progress in understanding commercial participation in Open Source communities, the are still many ripe opportunities for additional exploration. In particular, this work focused on two of the most successful Open Source com-

munities and examined commercial involvement as a whole. Both of these communities support numerous firms that embed and extend the technologies for very specific purposes, product focused firms in the parlance of chapter 4. Relative to community focused firms these firms have decreased motivation to contribute their innovations back to the community. An analysis of these firms and how they contribute back to the community could prove beneficial for foundations that wish to build community and ensure that development continues.

Bibliography

[1] ALPERN, B., AUGART, S., BLACKBURN, S., M, B., COCCHI, A., CHENG, P., DOLBY, J., FINK, S., GROVE, D., HIND, M., MCKINLEY, K., MERGEN, M., MOSS, J., NGO, T., SARKAR, V., AND TRAPP, M. The jikes research virtual machine project: Building an open-source research community. *IBM Systems Journal 44*, 2 (2005), 399–418.

[2] BONACCORSI, A., GIANNANGELI, S., AND ROSSI, C. Entry strategies under competing standards: Hybrid business models in the open source software industry. *Management Science 52*, 7 (July 2006), 1085–1098.

[3] BONACCORSI, A., AND ROSSI, C. Why open source software can succeed. *Research Policy 32*, 7 (July 2003), 1243–1258.

[4] BOORMAN, S. A., AND WHITE, H. C. Social structure from multiple networks. II. role structures. *American Journal of Sociology 81*, 6 (1976), 1384–1446.

[5] BROWN, A., AND BOOCH, G. Reusing Open-Source software and practices: The impact of Open-Source on commercial vendors. In *Proceedings of the Seventh International Conference on Software Reuse* (Austin, TX, Apr. 2002), Springer, pp. 381–428.

[6] BROWN, G. Linux - a platform for innovation in converged mobile handsets. *BT Technology Journal 25*, 2 (Apr. 2007), 126–132.

[7] BROY, M. Challenges in automotive software engineering. In *Proceedings of the 28th international conference on Software engineering* (Shanghai, China, 2006), ACM, pp. 33–42.

[8] BURTON, R. M., AND OBEL, B. *Strategic Organizational Diagnosis and Design: The Dynamics of Fit*, 3rd ed. Springer, Dec. 2003.

[9] CANONICAL LTD. About us. http://www.canonical.com/aboutus, Oct. 2008.

[10] CANONICAL LTD. TimeBasedReleases - ubuntu wiki. https://wiki.ubuntu.com/TimeBasedReleases, 2009.

[11] CANONICAL LTD. Ubuntu home page | ubuntu. http://www.ubuntu.com/, 2009.

[12] CAPEK, P. G., FRANK, S. P., GERDT, S., AND SHIELDS, D. A history of IBM's

Open-Source involvement and strategy. *IBM Systems Journal 44*, 2 (2005), 249–257.

[13] CARBONE, P. Competitive open source. *Open Source Business Resource* (July 2007), 4–6.

[14] CATALDO, M., HERBSLEB, J. D., AND CARLEY, K. M. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (Kaiserslautern, Germany, 2008), ACM, pp. 2–11.

[15] CATALDO, M., WAGSTROM, P., HERBSLEB, J., AND CARLEY, K. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* (Banff, Alberta, Canada, Nov. 2006), ACM Press, pp. 353–362.

[16] CERNOSEK, G. A brief history of eclipse. http://www.ibm.com/developerworks/rational/library/nov05/cernosek/, Nov. 2005.

[17] CONWAY, M. How do communities invent? *Datamation 14*, 5 (Apr. 1968), 28–31.

[18] CROWSTON, K., ANNABI, H., HOWISON, J., AND MASANGO, C. Effective work practices for FLOSS development: A model and propositions. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on* (2005), p. 197a.

[19] CROWSTON, K., AND HOWISON, J. The social structure of free and open source software development. *First Monday 10*, 2 (Feb. 2005).

[20] CROWSTON, K., WEI, K., LI, Q., AND HOWISON, J. Core and periphery in Free/Libre and open source software team communications. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)* (2006), p. 118a.

[21] DAFT, R. L., AND LENGEL, R. H. Organizational information requirements, media richness and structural design. *Management Science 32*, 5 (1986), 554–571.

[22] DE VEN, A. H. V., AND DELBECQ, A. L. A task contingent model of Work-Unit structure. *Administrative Science Quarterly 19*, 2 (1974), 183197.

[23] DEDRICK, J., AND WEST, J. Movement ideology vs. user pragmatism in the organizational adoption of open source software. In *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, K. Kraemer and M. Elliot, Eds. Information Today, Medford, NJ, 2007.

[24] DES RIVIERES, J., AND WIEGAND, J. Eclipse: A platform for integrating development tools. *IBM Systems Journal 43*, 2 (2004), 371–383.

[25] EICK, S. G., GRAVES, T. L., KARR, A. F., MARRON, J., AND MOCKUS, A. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering 27*, 1 (2001), 1–12.

[26] FELLER, J., AND FITZGERALD, B. A framework analysis of the open source software development paradigm. In *Proceedings of the twenty first international conference on Information systems* (Brisbane, Queensland, Australia, 2000), Association for Information Systems, pp. 58–69.

[27] FIELDING, R. T. Shared leadership in the apache project. *Commun. ACM 42*, 4 (1999), 42–43.

[28] FINK, M. *The Business and Economics of Linux and Open Source*, 1st ed. Prentice Hall PTR, Sept. 2002.

[29] FISHER, K. Microsoft antitrust finally over? http://arstechnica.com/news.ars/post/20021102-1030.html, Nov. 2002.

[30] FOGEL, K. *Producing Open Source Software*. O'Reilly & Associates, Sebastapol, CA, 2005.

[31] FRANKE, N., AND VON HIPPEL, E. Satisfying heterogeneous user needs via innovation toolkits: the case of apache security software. *Research Policy 32*, 7 (July 2003), 1199–1215.

[32] FREE SOFTWARE FOUNDATION. GNU general public license. http://www.gnu.org/copyleft/gpl.html, June 1991. available at http://www.gnu.org/copyleft/gpl.html – Visited April 28, 2007.

[33] FREE SOFTWARE FOUNDATION. Mailman, the GNU mailing list manager. http://www.gnu.org/software/mailman/, Dec. 2008.

[34] GALBRAITH, J. Organization design: An information processing view. *Interfaces 4*, 5 (May 1974), 28–36.

[35] GALBRAITH, J. R. *Designing Complex Organizations*. Addison Wesley, Oct. 1973.

[36] GALL, H., HAJEK, K., AND JAZAYERI, M. Detection of logical coupling based on product release history. In *14th IEEE International Conference on Software Maintenance* (Mar. 1998), IEEE Press.

[37] GEER, D. Eclipse becomes the dominant java IDE. *IEEE Computer 38*, 7 (2005), 16–18.

[38] GERMAN, D. The GNOME project: a case study of open source, global software development. *Software Process: Improvement and Practice 8*, 4 (Sept. 2004), 201–215.

[39] GERMAN, D. Software engineering practices in the GNOME project. In *Perspectives on Free and Open Source Software*, J. Feller, B. Fitzgerald, S. A. Hissam, K. R. Lakhani, and M. Cusumano, Eds. MIT Press, 2005, pp. 211–226.

[40] GHOSH, R. A., GLOTT, R., KRIEGER, B., AND ROBLES, G. Free/Libre and open source software: Survey and study. Tech. rep., International Institute of Infonomics University of Maastricht, The Netherlands, June 2002.

[41] GLYNN, E., FITZGERALD, B., AND EXTON, C. Commercial adoption of open source software: an empirical study. In *Empirical Software Engineering, 2005. 2005 International Symposium on* (2005), p. 10 pp.

[42] GOTH, G. Beware the march of this IDE: eclipse is overshadowing other tool technologies. *IEEE Software 22*, 4 (2005), 108–111.

[43] GRIMM, K. Software technology in an automotive company - major challenges. In *Software Engineering, 2003. Proceedings. 25th International Conference on* (2003), pp. 498–503.

[44] HALLORAN, T., AND SCHERLIS, W. High quality and open source practices. In *2nd Workshop on Open Source Software Engineering* (Orlando, Florida, May 2002).

[45] HARDIN, G. The tragedy of the commons. *Science 162*, 3859 (Dec. 1968), 1243–1248.

[46] HARS, A., AND OU, S. Working for free? motivations for participating in Open-Source projects. *International Journal of Electronic Commerce 6*, 3 (2002), 25–39.

[47] HECKER, F. Setting up shop: The business of open-source software. *IEEE Software 16*, 1 (1999), 45–51.

[48] HENDERSON, R. M., AND CLARK, K. B. Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly 35*, 1 (Mar. 1990), 9–30.

[49] HERBSLEB, J., AND MOCKUS, A. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering 29*, 6 (June 2003), 1–14.

[50] HERBSLEB, J. D., MOCKUS, A., FINHOLT, T. A., AND GRINTER, R. E. An empirical study of global software development: distance and speed. In *Proceedings of the 23rd International Conference on Software Engineering* (Toronto, Ontario, Canada, 2001), IEEE Computer Society, pp. 81–90.

[51] HERTEL, G., NIEDNER, S., AND HERMANN, S. Motivation of software developers in open source projects: An internet-based survey of contributors to the linux kernel. *Research Policy 32*, 7 (July 2003), 1159–1177.

[52] HORMBY, T. VisiCalc and the rise of the apple II. http://lowendmac.com/orchard/06/visicalc-origin-bricklin.html, Sept. 2006.

[53] IBM. IBM press room - 1998-06-22 IBM enhances and expands WebSphere product line in collaboration with apache and NetObjects - united states. http://www-03.ibm.com/press/us/en/pressrelease/2587.wss, June 1998.

[54] JOHNSON, J. P. Open source software: Private provision of a public good. *Journal of Economics & Management Strategy 11*, 4 (2002), 637–662.

[55] KERSTEN, M., AND MURPHY, G. C. Mylar: a degree-of-interest model for IDEs. In *Proceedings of the 4th international conference on Aspect-oriented software development* (Chicago, Illinois, 2005), ACM, pp. 159–168.

[56] KLEIDMAN, R. Volunteer activism and professionalism in social movement organizations. *Social Problems 41*, 2 (May 1994), 257–276.

[57] KOCH, S., AND SCHNEIDER, G. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal 12*, 1 (2002), 27–42.

[58] KOGUT, B., AND MEITU, A. Open-Source software development and distributed innovation. *Oxford Review of Economic Policy 17*, 2 (2001), 248—264.

[59] KRISHNAMURTHY, S. Cave or community? an empirical examination of 100 mature open source projects. *First Monday 7*, 6 (June 2002).

[60] KRISHNAMURTHY, S. An analysis of open source business models. In *Perspectives on Free and Open Source Software*, J. Feller, B. Fitzgerald, S. Hissam, and K. R. Lakhani, Eds. MIT Press, Cambridge, MA, June 2005.

[61] KUWABARA, K. Linux: A bazaar at the edge of chaos. *First Monday 5*, 3 (Mar. 2000).

[62] LAKHANI, K., AND WOLF, R. Why hackers do what they do: Understanding motivation and effort in Free/Open source software projects. In *Perspectives on Free and Open Source Software*, J. Feller, B. Fitzgerald, S. Hissam, and K. R. Lakhani, Eds. MIT Press, Cambridge, MA, 2005.

[63] LATTIX, INC. Lattix - software for architecture management. http://www.lattix.com/, Feb. 2009.

[64] LERNER, J., AND TIROLE, J. Some simple economics of open source. *Journal of Industrial Economics 50*, 2 (June 2002), 197–234.

[65] LJUNGBERG, J. Open source movements as a model for organising. *European Journal of Information Systems 9*, 4 (Dec. 2000), 208–216.

[66] MACCORMACK, A., RUSNAK, J., AND BALDWIN, C. Y. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science 52*, 7 (July 2006), 1015–1030.

[67] MADEY, G., FREEH, V., AND TYNAN, R. The open source software development phenomenon: An analysis based on social network theory. In *Americas Conference on Information Systems* (2002).

[68] MALCOM, J. Problems in open source licensing. In *2003 Australian Linux Conference* (2003).

[69] MANCHESTER, P. Eclipse kills open-source SOA projects. *The Register* (Nov. 2008).

[70] MARCH, J., AND SIMON, H. *Organizations*. Wiley, New York, NY, 1958.

[71] MARKUS, M. L., MANVILLE, B., AND AGRES, C. E. What makes a virtual organization work? *Sloan Management Review 42*, 1 (2000), 13–26.

[72] MCGREW, J. F., BILOTTA, J. G., AND DEENEY, J. M. Software team formation and decay: Extending the standard model for small groups. *Small Group Research 30*, 2 (Apr. 1999), 209–234.

[73] MCKUSICK, M. K. Twenty years of berkeley unix: From AT&T-Owned to freely redistributable. In *Open Sources: Voices from the Open Source Revolution*, C. DiBona, S. Ockman, and M. Stone, Eds. O'Reilly Media, Inc., Sebastapol, CA, 1999, pp. 19–30.

[74] MELLOR, C. Aperi dies on its arse. *The Register* (2009).

[75] MOCKUS, A., FIELDING, R., AND HERBSLEB, J. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology 11*, 3 (July 2002), 309–346.

[76] MOODY, G. *Rebel Code: Linux and the Open Source Revolution*. Basic Books, 2001.

[77] MOON, S., KIM, J., BAE, K., LEE, J., AND SEO, D. Embedded linux implementation on a commercial digital TV system. *Consumer Electronics, IEEE Transactions on 49*, 4 (2003), 1402–1407.

[78] MUSTONEN, M. Copyleft–the economics of linux and other open source software. *Information Economics and Policy 15*, 1 (Mar. 2003), 99–121.

[79] NEWMAN, M. Detecting community structure in networks. *The European Physical Journal B 38*, 2 (Mar. 2004), 321–330.

[80] OAKES, C. Netscape browser guru: We failed. *Wired* (Apr. 1999).

[81] OHLSSON, M. C., VON MAYRHAUSER, A., MCGUIRE, B., AND WOHLIN, C. Code decay analysis of legacy software through successive releases. In *Proceedings of the 1999 IEEE Aerospace Conference* (1999), vol. 5, pp. 69–81 vol.5.

[82] O'MAHONY, S. Guarding the commons: how community managed software projects protect their work. *Research Policy 32*, 7 (July 2003), 1179–1198.

[83] O'MAHONY, S. Non-Profit foundations and their role in Community-Firm software collaboration. In *Proceedings of the HBS-MIT Sloan Free/Open Source Software Workshop* (Cambridge, MA, 2003).

[84] OMII-UK. Welcome to OMII-UK. http://www.omii.ac.uk/, Feb. 2009.

[85] O'REILLY, T. Lessons from open-source software development. *Commun. ACM*

42, 4 (1999), 32–37.

[86] OSTERLOH, M., AND ROTA, S. Open source software development–Just another case of collective invention? *Research Policy 36*, 2 (Mar. 2007), 157–171.

[87] PARNAS, D. On the criteria to be used in decomposing systems into modules. *Communications of the ACM 15*, 12 (Dec. 1972), 1053–1058.

[88] PAUL, R. OpenOffice.org community conflict leads to fragmentation - ars technica. *ArsTechnica* (Oct. 2007).

[89] PAUL, R. Nokia to buy trolltech, will become a patron of KDE - ars technica. http://arstechnica.com/open-source/news/2008/01/nokia-buys-trolltech-will-become-a-patron-of-kde.ars, 2008.

[90] PELLED, L. H., EISENHARDT, K. M., AND XIN, K. R. Exploring the black box: An analysis of work group diversity, conflict, and performance. *Administrative Science Quarterly 44*, 1 (Mar. 1999), 1–28.

[91] PENNINGTON, H. Proposed release process/plans. http://mail.gnome.org/archives/gnome-hackers/2002-June/msg00041.html, June 2002.

[92] RAYMOND, E. S. A brief history of hackerdom. In *Open Sources: Voices from the Open Source Revolution*, C. DiBona, S. Ockman, and M. Stone, Eds. O'Reilly Media, Inc., Sebastapol, CA, 1999, pp. 19–30.

[93] RAYMOND, E. S. *The Cathedral and the Bazaar*. O'Reilly & Associates, Sebastapol, CA, Oct. 1999.

[94] RAYMOND, E. S. *The Art of UNIX Programming*, 1 ed. Addison-Wesley Professional, Oct. 2003.

[95] RICHARDS, J. Sun buys MySQL for $1 billion - times online. *Times Online* (2008).

[96] ROBERTS, J. A., HANN, I., AND SLAUGHTER, S. A. Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the apache projects. *Management Science 52*, 7 (July 2006), 984–999.

[97] ROONEY, P. Microsoft to publish 385 windows APIs, protocols to make antitrust case go away. *Computer Reseller News* (Aug. 2002).

[98] ROSEN, L. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall PTR, Aug. 2004.

[99] SARMA, A., MACCHERONE, L., WAGSTROM, P., AND HERBSLEB, J. Tesseract: Interactive visual exploration of Socio-Technical relationships in software development. In *Proceedings of the 2009 International Conference on Software Engineering* (Vancouver, BC, May 2009).

[100] SCACCHI, W. Free and open source development practices in the game community. *IEEE Software 21*, 1 (2004), 59–66.

[101] SENYARD, A., AND MICHLMAYR, M. How to have a successful free software project. In *11th Asia-Pacific Software Engineering Conference* (2004), pp. 84–91.

[102] SPJUTH, O., HELMUS, T., WILLIGHAGEN, E. L., KUHN, S., EKLUND, M., WAGENER, J., MURRAY-RUST, P., STEINBECK, C., AND WIKBERG, J. E. Bioclipse: An open source workbench for chemo- and bioinformatics. *BMC Bioinformatics 8*, 59 (Feb. 2007).

[103] STALLMAN, R. M. EMACS the extensible, customizable self-documenting display editor. *SIGPLAN Notices 16*, 6 (1981), 147–156.

[104] STALLMAN, R. M. *Using GCC: The GNU Compiler Collection Reference Manual for GCC 3.3.1*. Free Software Foundation, Oct. 2003.

[105] STEWART, K. J., AMMETER, A. P., AND MAURPING, L. M. Impacts of license choice and organizational sponsorship on users interst and development activity in open source software projects. *Information Systems Research 17*, 2 (June 2006), 126–144.

[106] STEWART, K. J., AND GOSAIN, S. The impact of ideology on effectiveness in open source software development teams. *MIS Quarterly 30*, 2 (June 2006), 291–314.

[107] TEASLEY, S. D., COVI, L. A., KRISHNAN, M., AND OLSON, J. S. Rapid software development through team collocation. *IEEE Transactions on Software Engineering 28*, 7 (July 2002), 671–683.

[108] THE ECLIPSE FOUNDATION. Eclipse public license - version 1.0. http://www.eclipse.org/legal/epl-v10.html, 2006.

[109] THE ECLIPSE FOUNDATION. Intellectual property policy. http://www.eclipse.org/org/documents/Eclipse_IP_Policy.pdf, Sept. 2008.

[110] THE ECLIPSE FOUNDATION. About the eclipse foundation. http://www.eclipse.org/org/, 2009.

[111] THE ECLIPSE FOUNDATION. BIRT home. http://www.eclipse.org/birt/phoenix/, 2009.

[112] THE ECLIPSE FOUNDATION. Eclipse foundation councils. http://www.eclipse.org/org/foundation/council.php, 2009.

[113] THE ECLIPSE FOUNDATION. Eclipse platform. http://www.eclipse.org/platform/, 2009.

[114] THE ECLIPSE FOUNDATION. Eclipse platform overview. http://www.eclipse.org/eclipse/eclipse-charter.php, 2009.

[115] THE ECLIPSE FOUNDATION. Higgins home. http://www.eclipse.org/higgins/, 2009.

[116] TIEMANN, M. Future of cygnus solutions: An entrepreneur's account. In *Open Sources: Voices from the Open Source Revolution*, C. DiBona, S. Ockman, and M. Stone, Eds. O'Reilly Media, Inc., Sebastapol, CA, 1999, pp. 71–91.

[117] TIEMANN, M. History of the OSI. http://www.opensource.org/history, Sept. 2006.

[118] TOULME, A. Eclipse and the lone developer at lunar ocean. http://www.lunar-ocean.com/eclipse-and-the-lone-developer/, 2009.

[119] VALETTO, G., HELANDER, M., EHRLICH, K., CHULANI, S., WEGMAN, M., AND WILLIAMS, C. Using software repositories to investigate Socio-Technical congruence in development projects. In *Mining Software Repositories 2007* (Minneapolis, MN, USA, May 2007).

[120] VAN WENDEL DE JOODE, R., DE BRUIJN, J. A., AND VAN EETEN, M. J. G. *Protecting the Virtual Commons*. Asser Press, Aug. 2003.

[121] VANCE, A. A software populist who doesnt do windows. *The New York Times* (2009).

[122] VON HIPPEL, E. Innovation by user communities: Learning from Open-Source software. *MIT Sloan Management Review 42*, 4 (2001), 82–86.

[123] VON HIPPEL, E. *Democratizing Innovation*. The MIT Press, Apr. 2005.

[124] VON HIPPEL, E., AND VON KROGH, G. Open source software and the "Private collective" innovation model: Issues for organizational science. *Organization Science 14*, 2 (Apr. 2003), 209–223.

[125] VON KROGH, G., SPAETH, S., AND LAKHANI, K. R. Community, joining, and specialization in open source software innovation: a case study. *Research Policy 32*, 7 (July 2003), 1217–1241.

[126] WATERS, J. K. Eclipse's third 'Release train' on schedule. *Application Development trends* (June 2008).

[127] WEBER, S. *The Success of Open Source*. Harvard University Press, Cambridge, MA, Apr. 2004.

[128] WEST, J., AND O'MAHONY, S. Contrasting community building in sponsored and community founded open source projects. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on* (2005), p. 196c.

[129] WILLIAMS, K., AND O'REILLY, C. Demography and diversity in organizations: A review of 40 years of research. *Research in Organizational Behavior 20* (1998), 77–140.

[130] WILLIAMS, S. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly Media, Inc., Mar. 2002.

[131] WOLFE, A. Eclipse: A platform becomes an Open-Source woodstock. *Queue 1*, 8 (2003), 14–16.

[132] YAMAUCHI, Y., YOKOZAWA, M., SHINOHARA, T., AND ISHIDA, T. Collaboration with lean media: how open-source software succeeds. In *Proceedings of the 2000*

ACM conference on Computer supported cooperative work (Philadelphia, Pennsylvania, United States, 2000), ACM, pp. 329–338.

[133] YE, Y., AND KISHIDA, K. Toward an understanding of the motivation of open source software developers. In *Proceedings of the 25th International Conference on Software Engineering* (Portland, OR, USA, 2003), pp. 419–429.

[134] YOUNG, R. Giving it away: How red hat software stumbled across a new economic model and helped improve an industry. In *Open Sources: Voices from the Open Source Revolution*, C. DiBona, S. Ockman, and M. Stone, Eds. O'Reilly Media, Inc., Sebastapol, CA, 1999, pp. 113–126.