Thesis Proposal: Vertical Interaction in Open Software Engineering Communities

Patrick Wagstrom

Department of Engineering and Public Policy

and

Computation, Organizations, and Society

Carnegie Mellon University

Pittsburgh, PA 15213

February 29, 2008

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 takes a vertical examination of Open Source ecosystems to identify the way that foundations, firms, and individuals come together to produce world class software despite their differing goals and values. At the macro level, I examine how competitors work together under open source foundations to create a stable market for each participant. Next I propose a study of interactions between firms in the community to identify to what degree the stated collaboration is actually present. At the next level down I examine how the presence of commercial firms in a community impacts that participation of individual developers within the community. After I analyze the methods of individual communication in Open Source communities to see if the patterns of communication can be used as a predictive element for project success. Finally, I close with a set of recommendations for firms wishing to participate in Open Source.

Introduction

Software engineering is evolving at a rapid pace. Firms that previously used only internally developed components are finding themselves working in new world with globally distributed development teams, proprietary off the shelf components, Open Source software, standards driving foundations, and industry consortia. Older firms with established products find themselves constrained as platforms evolve, methods change, and experienced developers retire exposing gaps in organizational knowledge. These factors are changing the face of software engineering, forcing firms to collaborate and coordinate across organizational boundaries like never before.

One of the biggest driving factors in this shift is the rise of the Internet and Open Source Software (OSS). Early descriptions of the OSS community often portrayed it as a wild west form of software development: developers work in their free time, code is donated for the good of the community, commercial use is frowned upon, and the only thing that matters is how well you code[13]. While such a view may have been appropriate for small projects in the 1990's, the reality is that OSS has shifted dramatically. Many commonly used Open Source software packages are written by developers working for commercial firms that seek exploit this software as part of their business plan. In other communities, volunteers and commercial developers come together to form foundations to steward and market other projects with little desire for direct commercial gain[14, 12, 17].

The OSS process has become so successful, that many firms are now using similar processes internally for their own development. VA Software markets an enterprise edition of the popular SourceForge.Net software that corporations can install to manage all of their projects. IBM is developing the Jazz platform to support radically distributed and agile development practices. Like OSS, it focuses heavily on computer mediated communication, iterative development, and gives individual developers the ability to manage their own branches of code and share those branches with other developers. Even Microsoft, which has long been viewed as the antithesis of Open Source, has allowed Open Source like sharing of code between licensees of the Windows CE operating system and recently submitted to licenses for approval to the Open Source Institute – the body that owns the rights to the term Open Source and validates licenses as meeting the requirements for Open Source.

Another major change introduced by Open Source is the commiditizing force it provides[18]. With many complicated pieces of software, there exists common pieces of functionality that each piece of software must implement. For example, in the integrated development environment (IDE) market, each product needs to interface with an editor, various revision control system, and a compiler. Before the success of the Eclipse project, each firm in the market space implemented each of these tools, taking resources away that could otherwise be used to focus on core competencies. Now, various vendors that compete in the market and build on the Eclipse platform can utilize these common components provided by Eclipse and use those resource previously allocated to standard features to further differentiate and enhance their product. Commoditization also provides an opportunity for specific vendors who focused on creating excellent versions of a common component, such as a text editor, to better showcase their work by making it easier for them to create a complete product or integrate their product with others [7, 4].

All of these changes induced by Open Source require additional collaboration and coordination. This required collaboration comes on a variety of levels. At the the highest level is the firm based collaboration and coordination that is necessary for foundations and firms and consortia to collaborate on commoditized software. At a lower level firms need to coordinate their actions and consider the strategies of their competitors in the broader Open Source community when considering what to release to the Open Source community. In addition, when entering an existing Open Source community, if a firm wishes to preserve the delicate ecosystem already existing around the project they must work within the norms and expectations of the community. Finally, individual developers must be able to communicate efficiently and directly with other programmers, architects, testers, and potentially users – a breadth of communication not previously needed.

What started with Open Source has since greatly expanded, other scientific and knowledge industries are taking adopting similar business strategies[1]. Many of these industries are still nascent in their development of norms and processes, however, if they continue to mirror early Open Source efforts then there is little doubt that similar structures and institutions will emerge. Using open source as a well developed example, this thesis seeks to examine interactions of foundations, firms, and individuals in open collaborative environments. Specifically I examine how Open Source creates new opportunities for interaction, cooperation, and competition across all levels of the Open Source ecosystem. I address these issues through a vertical analysis utilizing both quantitative and qualitative methods to best understand the community, model its interactions, and generate viable models for the development process.

Formally, there are four major sections of this thesis each addressing a particular level of communication within Open Source software engineering. The first section examines the Eclipse community to identify the ways that collaborators and competitors rally around the guiding focus of the Eclipse foundation to create high quality software and extract value from and emerging market. The second section continues to look at Eclipse to understand the firm to firm interactions at various levels in the community and to verify if the communication patterns resemble those that the members of community believe occur. The third section moves further down and examines the relationship between commercial firms and the individual volunteer developers through a longitudinal analysis of the GNOME project. Finally, the fourth section analyzes individual communication patterns within Open Source by building upon the concept of Socio-Technical

[1]For example `http://www.myexperiment.org/` is a collaborative site for computational biology researchers to share data, workflows, and integrate research that uses concepts from Web 2.0 and Open Source to build a community of researchers. In the Aerospace industry Boeing has announced that suppliers collaborating with it on the next rocket to the moon will be required to make their designs Open Source.

Congruence.

1 Open Source Ecosystems

While Open Source technology and products have been around since the dawn of computers, the term itself is relatively new, having arisen in 1998 as an alternative to the historic and sometimes confusing "Free Software," which indicates software that is both free as in speech and free as in zero direct cost. This confusion, and licensing terms around the dominant software license of the time, the GNU General Public License, had led many firms to avoid Free software and utilize a variety of proprietary or commercial solutions. Even at this young point, there were firms that had been eeking out a narrow existence in the field for years, but the announcement of the new name and formal definition that was friendlier to business, along with the changes in the information technology industry caused an explosion of new firms to emerge in the latter half of the 1990's. This rapid growth led to a plethora of unsuccessful business models and a handful of successful ones. Amongst the notable successful models were Cygnus Solutions' method of providing customized software development to work with existing tools in the embedded market[15] and Red Hat Software's strategy of giving away a free version of their Linux and then charging for a higher level support for commercial firms that require it[19].

This period also saw the origins of academic research on Open Source and its associated business models. In particular, Hecker described some of the more general reasons why firms might be willing to go into Open Source through his own experiences of releasing the code for the Netscape Web Browser, now called Mozilla, under an Open Source license while at Netscape[8]. Later, Martin Fink identified many of the economic and business options for individual firms working on Open Source, focusing on a more general method of utilizing Open Source strategies and the process of bringing those practices into a firm[5]. His work addresses many of the terms that are commonly utilized in Open Source and begins to address some of the community issues related to participating in Open Source from a corporate perspective. In 2003, Krishnamurthy bean to flesh out some of the more general business models that are in use by firms in Open Source. In particular, he identified four major categories of business models. The first model is that of a software distributor, which is the traditional model of firms such as Red Hat. In this model, firms take a community developed product and repackage the product, providing it for free or a fee and also providing service and support for the product. The second model is that of a derived software producer – a firm that takes the community developed product and derives a new product from it and charges for the enhanced product. A variation of the software producer is a software producer who is working with a produce under the GNU General Public License[6], which requires that derived products also be published and made available under the same license. In this model, the firm may charge for the derived product and support that goes with it, but they must also provide their modifications back to the community. The final related business model is that of the third party service provider which only provides expert service for a set of Open Source projects and does not directly provide contributions to the project[9].

While these previous efforts all provide valuable contributions to the scholarship in the field, their focus remains on the individual firm despite the fact that the landscape of Open Source has evolved. Successful projects have grown and now feature many competing firms working together on the same components of a larger project and seeking to differentiate themselves based on higher levels of functionality. In particular, most of the models cannot address the difficult issues of what happens when a commercial firm releases an internal project under and Open Source project to the community. In such a case, complete understanding of the contribution requires knowledge of the project and the method of community participation. As this thesis focuses on the interactions between commercial firms and their associated communities, it is important to develop a cohesive model that addresses the communal aspect of participation along with the business strategy for participation.

This section builds on my current work with Jim Herbsleb and Sonali Shah addressing value chains in the Eclipse Ecosystem. Our major data sources are a series of interviews with developers, managers, and executives involved in Eclipse, my personal attendance at EclipseCon in 2007 and soon in 2008, and an analysis of the business models, methods of interaction, and incentives for participation of the strategic developers and plug-in providers of the Eclipse Foundation. Much of this work has already been completed, it will be supplemented with feedback obtained from presentations to the Eclipse Foundation Members Meeting and a presentation at EclipseCon in Mid-March, 2008.

We are currently identifying the various business and interaction models in use in the community, addressing how money is made as a result of the model, the necessary resources, and how this forces competition in the community. Thus far, our research has identified the following dominant models for firm participation on the Eclipse Ecosystem:

- **Market Consolidation** - By promoting a product as Open Source it gains significant advantages over proprietary solutions. Within the Eclipse community, getting a technology adopted as part of the Eclipse platform provides additional benefits from the massive network effects in the platform. A savvy firm with the first product to be included in Eclipse in a market space gains a huge boost in distribution and also forces their competitors to change strategies. This is similar to what Microsoft did with the bundling of Internet Explorer and Windows. Within Eclipse, small firms, such as Actuate, have leveraged this to compete against large established firms. At a larger level, IBM utilized their connections after releasing Eclipse to consolidate the market into their ecosystem – once again placing IBM at the forefront of developer tools.

- **Commodity Utilization/IDE Enhancement** - Many of the firms that previously competed in the space that has been consolidated may still have products and customers they wish to support. These firms utilize the base of the Eclipse project to build new tools for development. For example, Adobe utilizes Eclipse as the basis for their Flex development studio.

- **Plugin Sales** - While much of the code in the ecosystem is written by professional developers, there is still great incentive to keep modifications proprietary. Some firms develop add-ons for the IDE that are a product unto themselves and would not exist without the IDE as a development platform.

This has led to a small number of firms arising that create plugins to enhance the experience, usually by improving over the Open Source tools, as is the case with Instantiations and their GUI designer too, or to provide some new functionality. This differs from the previous category in scale and also distribution method. These plugins are wholly dependent on a vibrant Eclipse ecosystem and work within the Eclipse community. When a firm uses the Eclipse platform as a commodity in developing a new IDE, they often fork off and create a new community.

- **Complimentary Goods** - As a project becomes a dominant ecosystem, vendors of support projects are pressed to create interfaces to Eclipse. Companies adopting this strategy create customized plugins for Eclipse so users and developers have direct access to their tools, which can be sold through other channels. This is particularly prevalent for existing source code management and build systems and is the strategy taken by OpenMake and CollabNet, among others. In addition, hardware vendors, such as Nokia and Intel, provide plugins to ease developers in creating solutions for their platforms.

- **Nested Platform Building** - As the Eclipse ecosystem continues to grow, the code has become more modular and attractive for purposes other than just and IDE. In particular, the OSGi component model and rich client platform (RCP) models make it easy for a firm or group of firms to create solutions without a need for a heavy infrastructure. This trend is becoming more common especially as firms seek to consolidate commodity components across an industry. For example the Higgins project seeks to provide a common framework for identity management, while the Aperi project utilizes RCP and OSGi to create a standard set of tools for storage management.

- **Customization/Consulting** - As more firms use Eclipse for internal development, there is a greater desire for customized plugins to work with existing work flows. Likewise, there is increased demand for support and easy distribution of Eclipse. A variety of firms have sprung up that provide these solutions, such as Innoopract, which develops an Eclipse "distribution" that packages up eclipse with a set of tested plugins, much like a Linux distribution does.

- **Users** - End users of a software project are often lost when examining a development community. These users play a key role in requesting features, providing testing, and often moving the market in a direction to increase the market share of the product. A wide variety of companies utilize Eclipse, and while they may not be highly visible, they play a key role in the ecosystem.

This section will continue to expand on the current work with Jim Herbsleb and Sonali Shah by expanding and enhancing the data. In our current work we have looks primarily at business models in the community, I intend to examine how the business models relate to the governance of the ecosystem through the Eclipse foundation and flesh out a complete view of the ecosystem of member companies, something which has not yet be done by academia, industry, or the Eclipse Foundation. This will take the form of additional interviews with members of the Eclipse community during and after EclipseCon 2008.

This section of the work greatly adds to the knowledge of software development and Open Source by providing a holistic analysis of a market leading Open Source ecosystem. Indeed, as more and more projects move toward Open Source methods of collaboration such an analysis will prove invaluable in strategic planning and design of such communities.

2 Firm to Firm Interaction in Open Source

While organizations typically have a plan for how they participate in Open Source projects, the reality of their interaction is a very different situation. Understanding the real methods of interaction for communities and firms is beneficial in understanding not only the process of Open Source software development, but also is a valuable component in helping to identify best practices in Open Source software development. This section builds on the qualitative results addressing how firms interact within foundations in section 1 by providing a quantitative analysis of how the firms act through observable channels in the ecosystem. In particular, this section will address the following questions:

- How do firms divide work in the Eclipse ecosystem?
- How do we best model the network of interactions between firms?
- What are the true interaction patterns across different communication mediums in the Eclipse ecosystem?

The data for this section will be collected from publicly available Eclipse resources; CVS source code repositories, Bugzilla bug tracking, project email lists, and community newsgroups. Portions of this data are publicly available through the work of the annual Mining Software Repositories workshop while other elements will need to be coordinated with individuals in the Eclipse community, something I am currently working on.

2.1 How is work divided in the Eclipse ecosystem?

Before founding a project in the Eclipse community (and similarly in the Apache community), projects are placed in an incubation stage. During this incubation stage each project has a mentor or set of mentors to help guide the process in the norms of working within the community – such as open decision making, documenting modifications to the software, and true collaboration with competitors. A project must fulfill a number of requirements before it can "graduate" from incubation – including demonstrating that a project appeals to individuals from more than one firm. This rule is designed to ensure that projects have sufficiently broad appeal and also to prevent the loss of a single entity from completely sidelining a project.

In some projects it may appear that there is a wide breadth of interest in a project, but what firms are really contributing? Even in platform building projects, such as the Aperi Storage Management framework which sees contributions from most enterprise storage vendors, the bulk of the work is led by only a handful of firms. From a strategic perspective, such inequity forces forms to overly reveal and contribute to a public good for which their competitors may be free riders. Conversely, a firm which contributes most of the work to a project may find itself in a beneficial position because of its ability to drive the direction of the project and the resultant community.

This section will examine a selection of projects within the Eclipse community to identify the distribution of work within the community. A longitudinal analysis will be performed to analyze the contributions by firms through release cycle intervals in the community to understand how projects increase (or possibly decrease) participation over their lifespan. The major elements of analysis will be an examination of contribution to project source code, both at an overall level, by module within the project[2], and an analysis of contributions to project bug tracking. The distribution of work will also be grouped by the business models and incentives for participation identified in section 1. This comparison allows for validation of interview statements from firms. For example, several plug-in developers have indicated that the architecture of Eclipse allows them to focus in a very narrow area of the overall project without being concerned about underlying technologies. Likewise, we expect that leaders of the platform should have a broad range of contributions across the entire platform.

The key component for this analysis is the identification of commercial developers and which firms they work for. Fortunately, within the Eclipse community there exists a norm of always using work email addresses for communication, allowing easy identification of contributions by firm. For a selection of approximately 10 major projects in the Eclipse community (to be determined based on data availability), the distribution of code contributions by firm over time will be calculated. Similarly, activity of firms on that projects bug tracker will also be analyzed to provide another view of interactions with the community.

[2]There are multiple ways to determine module structure in source code repositories. For this work, I take advantage of Java's explicit module structure.

2.2 How do we best model the network of interactions between firms?

The communication patterns fostered by Open Source software development typically create sparsely populated social networks with islands and a solid core-periphery structure[2]. However, such analyses typically look at only a specific communication medium, such as discussion on project forums, leaving out a multitude of other interactions. Furthermore, no such analysis has connected the actions of individuals to the firms they work for. This creates artificial holes in the information sharing network that exist through offline communication. In this section, I evaluate a variety of metrics for generating networks from three different communication streams:

- **Highly Technical Interactions - Source Code Networks** - there are multiple methods to construct source code networks, most of which rely on simple links between developers, A and files F, which is termed the AF matrix (in the congruence paper, this was termed the task assignment matrix[1]). Multiplying this matrix by its transpose yields links developers who have a potential for communication because they have modified the same file. However, the larger question is whether or not there should be link from developer a to file f. Should this network be built for all of time, or timeout after some predefined period? What if the network is only built for each release, how does the structure change between releases? How does the structure of the network evolve within releases?

  Furthermore, relationships between files can be calculated using a variety of different methods. In the past we have used a method that groups together logical dependencies as those files that were modified together and committed in the same atomic commit. This method is particular useful because it is independent of programming language constructs, however, it is sensitive to the work patterns of individuals in the community. Static code analysis, such as that used by MacCormack in his analysis of the Linux kernel and Mozilla projects, has the potential to capture relations that pervade the environment, such as the cascading dependencies that changing a low level debugging infrastructure would have[11]. Another method to compare to is the use of logical code and package structure, such as what is exhibited in the Java and C# programming languages, to create the dependency links between files.

- **Highly Social Interactions - Mailing Lists** - perhaps the easiest method of communication to analyze. The norm on most lists is to include the original poster of a message in the CC or To fields of the messages. Barring that, most email messages contain reply-to fields that identify the original message and who posted it. However, it may be worthwhile to go beyond this simple structure and include all participants in a thread or a branch of a thread, as a group.
  - **Clique** – Working at the thread level, all individuals who posted messages on a particular thread are considered to be in a clique with each other. On a physical level, this would be similar to all individuals being present a shared physical space and interacting with each other.
  - **Timed** – A link is added from an individual to all individuals posting within the thread who posted before the individual regardless of what branch they're communicating on. In this method, it is inferred that the individual wishes to communicate with all those who posted before him, but not all those who post after him.
  - **Thread** – A link is added from an individual to all individuals communicating on the same branch of communication along a path going back to the root of the conversation. In this way, a communication is not assumed to reach all those involved in the thread, only those along the line of communication within the thread.
  - **Direct** – The most conservative method of link generation that assumes a communication only between direct pairs of individuals in the thread reply network.
  - **Weighted** – A compromise for many of these methods is to use weighted links and then dichotomize the network at some threshold level.

My previous preliminary research on this topic has shown that the use of the direct method for link creation results in a substantially different ordering of nodes according to network metrics than clique, timed, and thread methods. However, such research was only a small sample mailing list and examined only eigenvector and betweenness centrality measures.

- **Semi-Technical Interactions - Bug Trackers** - traditionally the level of participation on bug trackers has resided somewhere between mailing lists and source code, however, recent tools that walk users through the process of filing a bug have greatly simplified this process and increased participation. The most popular bug tracker system is Bugzilla, but a variety of other tools also exist. Common among these tools is the ability to receive emails when new comments are added to the bug or fields are changed on the bug. The comment streams for almost all bug trackers is flat, which means it can be difficult to discern the network structure. In general, I will use the same methods proposed in for mailing lists with slight modifications on the non-threaded bug tracker communications.

The appropriate methods of network generation will be selected by analyzing the sensitivity of a family of metrics listed in table 1 for errors caused by omission and commission of links, both of which are potential issues when understand how Open Source ecosystems actually communicate. The goal here is not to identify a single best method, but to be aware of the limitations inherent in generating networks from sparse Open Source communication data. The sensitivity analysis will be done through repeated Monte Carlo perturbation of observed networks on each of the created network structures.

This research is vital for the modelling of Open Source systems as currently there is no “standard” way of building and understanding social networks from available data. Furthermore, most research that addresses these issues is vague on the methods used to generate the networks, leading to the current state where it is impossible to compare most social network results based on software engineering data (either Open or Closed source). By identifying the issues related to building these networks, the effects of noise, and how the construction affects common social network metrics, we provide valuable insight for interpretations and lay a foundation for improved research in the future.

Table 1: Metrics analyzed on various network constructions

Node Properties	Network Level Properties
• Node Degree	• Density
• Betweenness Centrality	• Generalized Network Structure
• Eigenvector Centrality	• Network Level Congruence
• Closeness Centrality	• Average nodal metrics
• Network Size	
• Network Density	
• Task Exclusivity	

2.3 What are the true interaction patterns across different communication mediums in the Eclipse ecosystem?

The final portion of this section analyzes interactions between firms, grouping individuals by firm to identify how often and across what mediums do firms communicate and collaborate. For each of the projects within Eclipse I propose to generate a network of actual communication of the firms using the methods derived in section 2.2. I will compare these networks to the networks observed from interviews and analysis of the Eclipse.org project pages that list the firms active on each project.

This provides an important real world verification of firm communication in Open Source software and can have potentially large impacts on the development of Open Source as a viable communication medium if it is found that there is little communication between firms, which means that a single firm could easily steer the project in a way that is not beneficial to the entire community, or if there is little cooperation on projects, which means that firms may be gaming the rules of the foundation to push projects that have little interest. From a community maintenance standpoint, if such tools become standard in the community, a project with little communication would be susceptible to business pressures on a single firm.

3 Corporate Involvement in Volunteer Communities: There Goes the Neighborhood

While much of the publicity on Open Source arises from commercial entities releasing previously proprietary projects as Open Source – for example the release of the Eclipse, Netscape, and Solaris source code – a frequent mode of interaction by firms with Open Source is to contribute to an already existing project with a stable community. In this alternative model, firms must work within the bounds of the community, surrendering some elements of control in exchange for the goods of the community. Often times firms participating on such projects continue to rely on the contributions of the individuals to the projects for core functionality and complete control of the project is not desired.

While many projects see corporate involvement as a validation of the success of the project and community they have worked so hard to create, there exists possible negative consequences to volunteer communities that welcome commercial developers with open arms. This section of identifies and identifies several key issues related to commercial participation in existing Open Source software communities.

In the context of this section, an Open Source software community is a community which produces several different projects, each of which may have different sets of developers. Typically these communities surround larger organizations and foundations which oversee a set of related projects, such as the Eclipse foundation and Apache Software Foundation. Within these communities, most of the software serves a set of related needs and have similar work practices and licensing structures. This allows a single developer to easily change between projects in the community or to work on multiple projects in the community.

The work patterns of commercial developers tend to differ dramatically from those of volunteer developers – not only do commercial developers typically spend much more time working on projects, they also have the ability to collaborate more closely with other developers working at the same firm. These actions allow developers to write more code in a shorter period of time, and may allow for code to be less modular because of the ability to extract information from co-workers at the same site. At the micro level, these changes would be manifest within various smaller modules of individual projects. We reason that the increased **cognitive complexity** of software projects as a result of co-located or full time commercial developers working on a volunteer Open Source project will drive volunteers from those areas of projects populated by commercial developers.

At a higher level, the participation of commercial developers in an Open Source project is likely seen as a mark of validation for the project providing **momentum**, increasing the profile of the project, and attracting developers who wish to signal their attractiveness to potential employers[10] along with volunteer developers who desire to work in a stable and successful project. In this context, we expect that commercial developers participating in an Open Source project lead to additional new volunteers working on the project.

Commercial developers, in addition to bringing resources and momentum also introduce an element of **heterogeneity** to the project. When a previously all volunteer project attracts commercial developers, individuals may feel as though their work is being exploited for commercial gain or that commercial interests are trying to push the project in a direction that is not suitable for volunteer developers. This causes tension in the community and may force volunteers to leave the project.

It is also unrealistic to group all commercial firms into the same broad categories, while some firms have a very good reputation for working with volunteer communities, other firms are known for being difficult because of their goals or work policies. To account for this, firms are broken into two different categories which may be seen as proxies for the **norms and values** of the firm:

- **Community-focused Distributors** – Firms that focus on taking the complete output of the community and marketing it as a product. For example, Linux distributors such as Red Hat and Mandriva, and Eclipse distributors such as Genuitec and Innoopract, take complete products from Open Source communities, provide additional packaging and clean up the software some, and create a marketable product.

- **Product-focused Manufacturers** – Firms that don't need the entire output or work only on a particular aspect of the software package. For example, IBM utilizes the Apache Web Server as part of the Web Sphere application server, but not the complete output of the Apache Project. Likewise, the One Laptop Per Child contributes to portions of the GNOME project, but only needs the results of some portions of the project.

The classification here is different than that presented in section 1 largely because there are only nine major commercial firms participating in GNOME. In addition, the architecture of the project – it lacks a common plugin architecture – and the federated nature of the community make many of those business models not applicable.

These issues are addressed through the longitudinal analysis of a large and successful volunteer originated community, the GNOME project. A set of interviews with 18 of the developers helped to confirm our firm classification scheme and provide qualitative confirmation of our hypotheses. This was followed up with analysis of 10 years of longitudinal data collected from project source code, bug tracking, and mailing list repositories.

3.1 Preliminary Results

I've attached an advanced draft paper on this issue that will be going to ISR in the near future as appendix A in this proposal. In summary, we have found the following:

- **Cognitive Complexity** - There is no support for the hypothesis that commercial developers will drive volunteer developers to less complex areas of a project. This was tested by examining the effect of commercial developers at the module level.

- **Heterogeneity** - There is no support that the heterogeneity introduced by commercial developers causes volunteer developers to leave projects.
- **Momentum** - The presence of commercial developers was found to have a positive relation to increased volunteer participation.
- **Norms and Values** - When broken up by business model, community focused firms, which share the norms and values of the community were found to have a high and positive relationship with increased volunteer participation, while product focused firms had no statistically significant relation.

3.2 Proposed Future Analysis

This work has established that there is a temporal relationship between the presence of commercial developers and subsequent presence of volunteer developers, but did little aside from an examination of the business models of the firms to identify possible other observable elements that may lead to increased future volunteer participation. I propose to expand this analysis by performing more in-depth analysis of communication in the community. I have previously done some analysis which found that for the most part, community and product focused developers showed little preference in their targets of communication. However, this analysis was at the aggregate level and could have missed important relationships.

- **Preference of Volunteer Responses** - One thing we have not directly tested for is signalling in the community[10, 3]. If individuals are attracted by the presence of a job, they would be more likely to perform actions which can be seen as increasing their profile to commercial developers. I will generate the response networks for messages in the community and test if volunteers preferentially communicate with commercial developers and if such communication leads to sustained participation in the project.

- **Participation in Feature Development** - Another conjecture put forth in the work is that volunteer developers are attracted to community focused developers because the features they develop are more directly applicable to the small user, rather than many product focused firms that focus on enterprise class software development. Utilizing information out of the CVS repository and bugzilla databases this can be tested by clustering the networks to observe the propensity of volunteers to be grouped with community and product focused developers.

As a subtext, this section is the most fully developed section of the thesis, and currently stands on its own as a fairly complete work. Given that, the proposed future analysis is also the section that I'm most willing to remove.

4 Individual Communication and Socio-Technical Congruence

Previous research by Cataldo et. al. developed the metric of socio-technical congruence for use in evaluating the fit of an organization's communication patterns to task dependencies [1]. As an organization reaches higher overall congruence between task dependencies and communication, time to resolution of defects was reduced, indicating an improvement in organizational efficiency. As proposed, this metric is a matrix based algorithm that produces a single number for the entire organization. While such information is useful for a workgroup manager, who is responsible for understanding the overall status of a project, the usefulness of such a metric to individual developers is questionable.

Recently, Helander et. al. have extended the algorithm as a graph theoretic model with improved runtime efficiency and reduction of memory requirements[16]. Chief among the accomplishments of these extensions is the weighting of individual edges and their contribution to organizational congruence and the introduction of the concept of communication gaps to the analysis. In the case that all edge weights are set dichotomized to 0 or 1 then the this algorithm produces identical results to the congruence metric of Cataldo et. al.

I propose extending these metrics to create a family of metrics that individuals can better comprehend and adjust their behaviors accordingly. An individualized version of congruence is necessary to provide incentive for developers to take action to improve their own personal congruence, and thus improve the organizational congruence. A preliminary version of this work was presented at the 28th Sunbelt Conference in St. Pete Beach, Florida in January of 2008.

There are two classes of individualized congruence which I propose - unweighted and weighted individual congruence. Unweighted individual congruence, IC_u, is very similar to overall network congruence. A coordination requirements matrix $\mathbf{C_R}$ is generated for the entire network as is the actual coordination matrix $\mathbf{C_A}$. The unweighted for individual congruence for an ego is then calculated by observing the congruence between $\mathbf{C_R}$ and $\mathbf{C_A}$ only for those edges in $\mathbf{C_R}$ that are incident upon the ego in question.

Weighted individual congruence is slightly more complicated, a weighted coordination requirements matrix $\mathbf{C_R}$ is generated either by not dichotomizing the matrix, or through the graph theoretical model of Helander. The actual coordination matrix, $\mathbf{C_A}$ is assumed to be dichotomized.

Formally the individual weighted congruence, IC_w for ego i can be expressed as:

$$IC_{w,i} = \frac{\sum \left(\mathbf{C_R}\left[i,\right] \times dichot\left(\mathbf{C_A}\left[i,\right]\right)\right) + \sum \left(\mathbf{C_R}\left[,i\right] \times dichot\left(\mathbf{C_A}\left[,i\right]\right)\right)}{\sum dichot(\mathbf{C_R}\left[i,\right]) + \sum dichot(\mathbf{C_R}\left[,i\right])} \tag{1}$$

Where $dichot(\mathbf{x})$ represents dichotomizing matrix x to 0,1 such that all cells > 0 are set to 1, the multiplications are element wise multiplications, and we assume that there are no diagonals in the coordination requirements matrix ($\forall i : \mathbf{C_R}[i,i] = 0$).

4.1 Preliminary Results

Using data from the GNOME project with a dependant variable of time to resolve software defects, unweighted individual congruence was found to have no statistically significant relation. However, this data produces very sparse matrices, which may be overly or underly sensitive. Contrary to our expectations, high weighted individual congruence was found to be related to an increased time to resolve bugs. This result also mirrors analysis done on a proprietary project while at IBM Research. Also, consistent with research from IBM, we found that the more communication required by an individual, as shown by the number of edges in the $\mathbf{C_R}$ matrix for ego i, the lower the time to resolve software defects. Like the previous work, we found that high levels of organizational congruence were predictors of faster time resolve bugs. Thus, we have a situation where the organizational and individual level metrics predict results in different directions. The regression for the model can be found on page 14 of appendix B.

This is a rather counter-intuitive result as in the edge case it suggests that highly coupled code with many dependencies and developers who rarely communicate is best for individual performance, but worst for organizational performance. From a tool development perspective such a find is disconcerting because individuals may be wary of contributing toward an overall team goal if it will hurt their individual evaluation. Recently, this result has been found by a team from the MIT Media Lab when testing performance of individuals and teams in German a bank. High levels of overall communication were found to increase overall team performance, but lower individual performance (Waber et. al., submitted to Academy of Management).

I will continue to examine this phenomenon by looking at more data within the GNOME community to examine if this result is consistent across more than the ten sample projects previously presented. In addition, if resources and data avail themselves, I would like to compare the results found in the GNOME community to the results found in Eclipse, which has a more formal operations and participation model.

4.2 Uncertainty and Sensitivity in Socio-Technical Congruence

Despite best efforts in the field, social networks generated from archival data are passable at best, and quite frequently have severe errors and inconsistencies. This problem is magnified in measurements such as congruence where we are concerned not only with the presence of a link, but also the underlying semantic meaning of the link. For example, it may be possible to use email archives to infer that two developers, Alice and Bob, communicated during a period of study. However, absence a content analysis of all communication, it is not possible to know if the communications between Alice and Bob were relevant to their work assignments or not. This uncertainty hits at the core of socio-technical congruence and should it be found that the metrics are overly sensitive to noise, could seriously damage the validity of the metrics.

I propose two different models to understand the uncertainty in this model. In the first model, we assume that each link has a chance of error of omission or error of commission with various probabilities and re-analyze the network outcome. In the second model we assume a prior probability for communication between each ego and alter in the network, and when a communication is found, we update the probability. Uncertainty is accounted for by instantiating the actual communication network according to the new probabilities and calculating the network stats. Currently, due to technical constraints, I plan to only analyze uncertainty and sensitivity on the $\mathbf{C_A}$ matrix, which lends itself to more subjective methods of creation, as previously addressed in section 2.2 of this thesis.

4.2.1 Errors of Commission and Omission

This model of uncertainty addresses systemic errors in the data collection method, either caused by faulty data, or missing data on communication methods. I propose to run a set of Monte Carlo simulations on multiple sets of real world data. This is a two axis experiment where the rate of omission is scaled from 0 to 25% and the rate of commission is also scaled from 0 to 25%. At each point I will run 100 samples with each data set and generate a distribution of overall socio-technical congruence, along with weighted and unweighted individual socio-technical congruence.

From a technical perspective, such analysis requires significant amounts of computational power, however I have some prototype tools that utilize Amazon's EC2 computing service which can provide hundreds of computational nodes for approximately \$0.10/hr/node – reducing weeks worth of runs to a few hours.

4.2.2 Probabilistic Communication

Creating a probabilistic communication model is even more complex, as rather than utilizing a uniform probability for the presence of each link in $\mathbf{C_A}$, the probability is based on the value of that cell and the base probability for a link – in essence a slightly randomized dichotimization of the data. In many situations similar to this, Bayesian inference would be the ideal method for updating priors, yet the data in $\mathbf{C_A}$ represents only positive observations, with no concept of negative observations for decreasing a prior. It is possible that I could proceed using only a low prior and positive observations, and such an option will be explored.

This section of the work will provide valuable insight into the stability of individual socio-technical congruence and also provide additional insight into generating networks from observed data.

5 Timeline

- **March 3rd, 2008** - Submit preliminary workshop paper addressing Socio-Technical Congruence at the individual level (section 4) to STC 2008 Workshop
- **March 5th, 2008** - Propose
- **Early March - Mid March** - Collect data on Eclipse Project for section 2 – the main task here is writing a scraper to rebuild profile information for the bug database. Retool framework for Congruence to allow for multiple dependency network generation methods.

- **March 17th-20th** - Present preliminary findings for section 1 at EclipseCon. Conduct and schedule interviews to further research on sections 1 and 2.
- **March 20th - Mid April** - Conduct follow-up interviews. Parse and load the data related to Eclipse. Explore the implications of network structure and dependency generation on the congruence metric. Generate data from more projects to better understand the "communication overload" phenomenon.
- **April 18th** - Submit more advanced version of individual congruence to CSCW 2008.
- **Mid April - Early May** - Implement and analyze probabilistic model for congruence.
- **May 1st** - Sloan Industry Studies conference provides additional feedback on section 1.
- **May 10th** - STC 2008 workshop provides additional feedback on section 4.
- **Mid May - Early June** - Evaluate methods of network construction using Eclipse data. Perform large scale sensitivity analysis on data of network construction method.
- **Mid June - Early July** - Generate structures for Eclipse to evaluate how firms really work together within the Eclipse ecosystem.
- **Summer** - Hopefully get reviews from ISR for "There Goes the Neighborhood" paper, providing feedback on section 3.
- **Mid July - Early August** - Put final touches on writing of thesis. Convince my wife that she really wants to read thesis to help proofread it.
- **Mid August** - Prepare slides, fudge room of a week or two.
- **Late August 2008** - Defend

References

[1] Cataldo, M., Wagstrom, P., Herbsleb, J., and Carley, K. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *2006 Conference on Computer Supported Cooperative Work* (Banff, Alberta, Canada, Nov. 2006), ACM Press, pp. 353–362.

[2] Crowston, K., Wei, K., Li, Q., and Howison, J. Core and periphery in free/libre and open source software team communications. In *Hawaii International Conference on System Sciences* (2006), p. 118a.

[3] Dahlander, L., and Magnusson, M. G. Relationships between open source software companies and communities: Observations from nordic firms. *Research Policy 34* (May 2005), 481–493.

[4] des Rivieres, J., and Wiegand, J. Eclipse: A platform for integrating development tools. *IBM Systems Journal 43* (2004), 371–383.

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

[6] Foundation, F. S. Gnu general public license, June 1991. available at http://www.gnu.org/copyleft/gpl.html – Visited April 28, 2007.

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

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

[9] Krishnamurthy, S. *An Analysis of Open Source Business Models*, 1 ed. MIT Press, Cambridge, MA, June 2005.

[10] Lerner, J., and Tirole, J. Some simple economics of open source. *Journal of Industrial Economics 50* (June 2002), 197–234.

[11] 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* (July 2006), 1015–1030.

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

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

[14] Shah, S. K. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science 52* (July 2006), 1000–1014.

[15] Tiemann, M. *Future of Cygnus Solutions: An Entrepreneur's Account*. O'Reilly Media, Inc., Sebastapol, CA, 1999, pp. 71–91.

[16] 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 2008* (Minneapolis, MN, USA, May 2007).

[17] West, J. How open is open enough? melding proprietary and open source platform strategies. *Research Policy 32* (2003), 1259–1285.

[18] West, J., and Gallagher, S. *Patterns of Open Innovation in Open Source Software*. Oxford University Press, 2006, pp. 82–108.

[19] Young, R. *Giving it Away: How Red Hat Software Stumbled Across a New Economic Model and Helped Improve an Industry*. O'Reilly Media, Inc., Sebastapol, CA, 1999, pp. 113–126.

Appendices

A “Won’t You Be My Neighbor”

This paper is the preliminary draft for section 3

B Understanding Communication, Coordination, and Congruence at an Individual Level By Harnessing Baseball Statistics

This is a copy of the presentation that I gave at Sunbelt in January 2008 corresponding to section 4 of the thesis.