A Social Network Approach to Free/Open Source Software Simulation

Patrick Wagstrom[1], Jim Herbsleb[2], Kathleen Carley[2]
pwagstro@andrew.cmu.edu, jdh@cs.cmu.edu, kathleen.carley@cmu.edu
[1] Department of Engineering and Public Policy
[2] Institute for Software Research International
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213 USA

January 15, 2005

Abstract

Free and Open Source Software (F/OSS) development is a complex process that is just beginning to be understood. The actual development process is frequently characterized as disparate volunteer developers collaborating to make a piece of software. The developers of F/OSS, like all software, spend a significant portion of their time in social communications to foster collaboration. We have analyzed several methods of communication; a social networking site, project mailing lists, and developer weblogs; to gain an understanding of the social network structure behind F/OSS projects. This social network data was used to create a model of F/OSS development that allows for multiple projects, users, and developers with varying goals and socialization methods. Using this model we have been able to replicate some of the known phenomena observed in F/OSS and provide a first step in the creation of a robust model of F/OSS.

Keywords: Open Source, community, simulation, social networks, collaboration

1 Introduction

Free and Open Source Software (F/OSS) has changed the way that businesses operate. To ignore the changes of the open source revolution is to risk loss of market share to competitors who can create projects, faster, cheaper, and better thanks to F/OSS. Fully embracing F/OSS software means relying on a little understood process and various desires of different developers. Investing large amounts of manpower in a project that loses its community may be worse than never using F/OSS in the first place.

When considering what F/OSS software to adopt, or contribute to, the decision may often be helped by examining what other companies have adopted that piece of software. This is possible for software that has wide distribution, an established track record, and a supporting community, such as Mozilla, GNOME, and KDE. For a new project, or a project that fills a niche need, the community may be small and still developing. Given that many F/OSS projects end up failing, determining involvement with a small or new project is a difficult decision. With commercial software it is possible to perform a vendor analysis and obtain an idea about the stability of the vendor and the product. For an entirely volunteer F/OSS project predicting future releases may be very difficult.

While it may not be possible to do a formal business analysis on a group of developers working on a project together, it is possible to analyze their interactions as a team. Many F/OSS projects utilize hosting environments such as SourceForge.Net or savannah.gnu.org to manage their source code, mailing lists, and bug tracking. Such hosting environments create logs of most project related actions that can later be mined to understand the community structure and interaction patterns. Large portions of this information has already been analyzed by Xu and Madey[15] and Madey, Freeh, and Tynan[11]. Such an analysis, while providing a good first step, captures only the formal interaction through channels such as shared code creation and bug-tracking. We believe that utilizing a variety of different data streams can create a more robust data set that more accurately represents developer interactions by capturing both the formal and informal interactions of developers and users.

This information can then be used as the basis for a simulation to predict future outcomes for a group of projects in a competing space. Such information could be critical for a corporation when deciding which project to adopt or which project to contribute too. Previous projects have also used simulation to predict the results of F/OSS projects. In particular, Dalle and David's SimCode accurately reflects the wishes of individual developers contributing to a single project[1] and Gao, Madey, and Freeh have attempted to model the complete ecosystem of F/OSS from a social network analysis point of view using data from SourceForge.Net[10, 3, 2]. Our work goes beyond these by allowing for multiple projects and following not only the number of developers on a given project, but also the progression of the code quality in the project, as represented by a fitness function.

The rest of the paper is structured as follows: section 2 describes the setup for the collection of social network data and the incorporation of that data into the simulation; section 3 provides a brief overview of the results of the social network analysis on three different data sets; section 4 provides a brief overview of the simulation model; the paper concludes with a discussion of contributions and suggestions for future research in modelling and simulation of F/OSS in section 5.

2 Methodology

We conducted two related studies to further the understanding of F/OSS development. In the first study a variety of data streams were utilized to understand the social networks for F/OSS developers. In the second study we used the information obtained through F/OSS social network analysis to build a simulation of developer interaction as a step toward a complete F/OSS simulation.

Three data streams with differing constraints on the community structure were analyzed to build representative social networks. First, we data from a web site, Advogato.org, that allows F/OSS developers to "certify" other developers as having attained a particular skill level. This gives us a very " liberal" view of developers' social networks, since they may be certified by others who don't know them well, or that know them only by reputation. Thus, links in Advogato reflect only that one developer believes he or she knows something about another developer's skill level. Second, we use mailing list archives for three different projects. This gives us a more stringent network measure, since links are generated only when one developer communicates directly with another. Finally, we used links in F/OSS developers' weblogs. Links here indicate that a developer read another developers weblog, found an entry relevant to his or her own writing, and inserted a pointer. This is the most stringent indicator of a social network link.

The social network information was used as a basis for social interaction in a simulation model. The model is an extension of model we previously designed called OSSim, short for Open Source Simulation[14]. A series of virtual experiments was run by varying parameters

such as socialization, coupling, and developer preference to try and replicate known phenomena in F/OSS development.

3 Social Network Analysis

Much is known about the attributes of individual developers through a series of survey and overview research over the past few years[4, 8, 6]. We know have a good idea of what causes a developer to contribute to F/OSS in the first place and are able to characterize them with a fair degree of accuracy. However, little work has been down with regards to the attributes of the developer network as a whole. Notable pieces of work are that of Xu and Madey[15] and Madey et. al.[11], who state that project size roughly follows a power law. However, their model has some critical flaws, notably it assumes all developers on a project know each other regardless of the size of the project. This is not realistic for many projects, for example the Linux Kernel has 463 developers listed in the `CREDITS` file and hundreds more who are not listed[1].

To overcome this restriction and provide possible future extension into the temporal domain, we utilized three distinct data collection methods. First, we obtained a copy of and analyzed the complete database from Advogato.org[9], a social networking site for F/OSS developers that utilizes a trust metric to rank developers. Secondly, we have created a system called BlogLinks that analyzes the links inherent in F/OSS developer weblogs to infer a social network. Finally, the mailing list archives for several projects have been downloaded and analyzed to provide a third data stream.

3.1 Advogato.Org - Social Networking for F/OSS Developers

Analysis of the data from Advogato.org yields an intertwined social network that differs greatly from the network proposed by Xu and Madey. Contrary to their network, which has a few linchpins holding groups together, the network obtained by the certifications in Advogato indicate a highly connected network.

Within Advogato there are three different classes of developers: apprentice, journeyman, and master. Apprentices generally are low volume developers who contribute during their free time. Journeymen typically lead a project, but usually do not develop full time or lead any large community efforts. Masters usually have a job that allows them to develop F/OSS full time and frequently lead large scale community efforts. Level in the system is determined through a trust metric and the path of certifications[2]. Because sign up is available to anyone, even those not involved in F/OSS development, there is a fourth class of users, the unranked (observer) users who are not part of the social network. A breakdown of group size and aggregate social network size is shown in table 1.

The trust network in Advogato works by having users certify the level of other users and calculating a trust metric where some developers have a known skill level. For example, I may be a "Journeyman" and wish to certify that Richard Stallman is at "Master" level. An analysis inter-level certifications shows either that most people tend to overestimate the skill of others when making a certification or that there is a trend toward certifying people who are well known. In reality, both are true, over-estimation of skills is common and many users feel compelled to certify people such as Richard Stallman at master level.

[1] As of version 2.6.9.

[2] The exact method of establishing developer rank is through the use of a trust metric. The development of Advogato was done as a test bed to test various resilient trust metric algorithms and this research will serve as the basis of Raph Levien's thesis at UC Berkeley.

Class	N	Out Links	In Links
Observer	8046	27053	20919
Apprentice	296	3213	2747
Journeyman	869	16341	15115
Master	235	5733	13559

Table 1: Breakdown of Social Networks from Advogato.org Data. Developers who are part of the trust network are in the Apprentice, Journeyman, and Master categories.

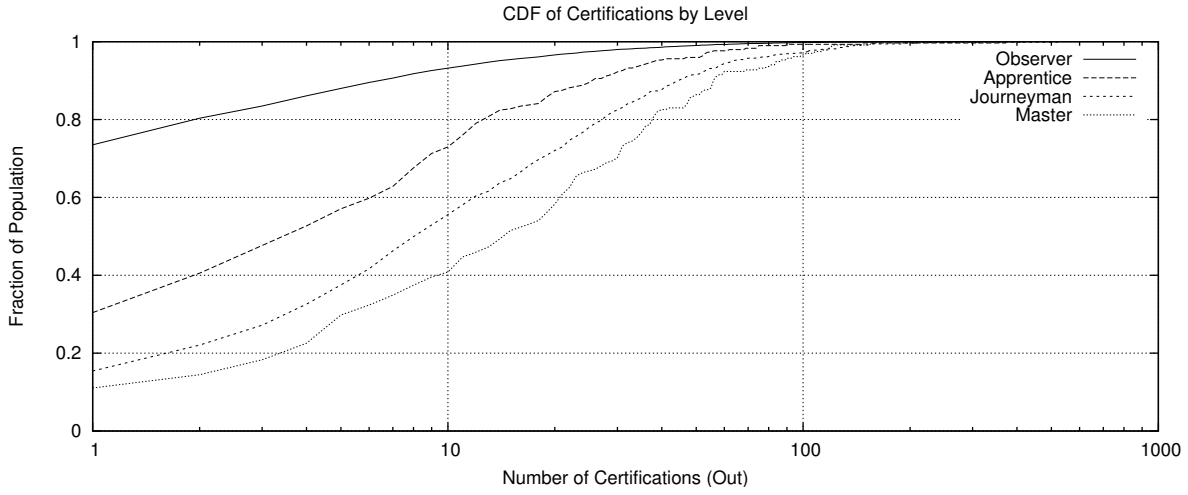

Figure 1: Cumulative distribution function (CDF) of Social Network Size from Advogato.Org. As should be expected, social network size, as determined by the out degree of a user, is directly related to the skill class in the dataset. This is indicative of the networks of people who are managers and more prominent versus people who work on the edges of the network. The very long tails on the distribution are a result of the ease with which one can establish a link in the network.

Lastly, an analysis of the social network size of the developers was performed and the distribution can be seen in figure 1. Because it is so easy to certify people, and no confirmation is needed from the other party, many people certify others who they only partially know or may have heard of. When combined with the fact that the data are static, this very likely overestimates the true size of the social networks within F/OSS.

3.2 BlogLinks

Another method used to analyze the social network of F/OSS developers was the creation of a project called BlogLinks that tracks weblog aggregators[3] of F/OSS projects. Fourteen different projects with 560 developers were tracked from September 15 to November 15, 2004, yielding more than 12,000 entries with more than 19,000 links to more than 15,000 distinct URLs. The concept of a direct link was used to infer social network structure. If a link exists from the weblog of one developer to an entry in the weblog of another developer, then a directed link

[3]Aggregators are pieces of software that retrieve the RSS, or site syndication feeds, for weblogs of individual developers. The results are then presented on a single web page for ease of viewing and tracking the community. Many F/OSS projects use the software called Planet, which can be found at `http://www.planetplanet.org/`.

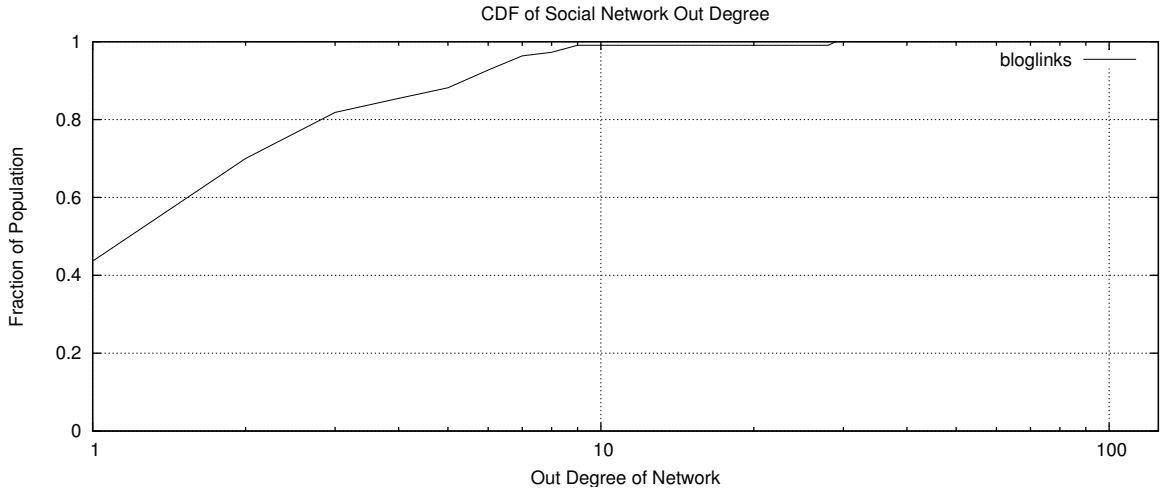

Figure 2: CDF of Social Network Size from BlogLinks. Similar to what was seen in the mailing lists, the size of social network obtained by analysis of weblog entries shows that most users, 99%, have users have a social network of fewer than 10 alters for the time period studied.

is placed in the social network graph. Using this information, it was possible to build a social network for 287 of the 560 developers.

The CDF of this social network is shown in figure 2 and indicates a smaller social network than the data from Advogato.org. Primarily this is because of the differences in requirements for inclusion: the criteria for inclusion in the BlogLinks social network is much tighter as both participants must maintain weblogs and link existence is predicated on developer A linking to an entry in developer B's weblog.

We note in passing that simple network measures often associated with leadership and importance (such as betweenness and centrality), computed on the BlogLinks data, tended to identify significant "hackers." Four of the top five for these measures were members of the GNOME foundation, the fifth is a developer on Debian GNU/Linux and now works for Canonical, Ltd. on the Ubuntu GNU/Linux distribution. This suggests that it is possible to extract useful information about a project structure and developer skill by social network analysis of weblogs despite the sparsity of the data set and provides an interesting avenue for future research.

3.3 Mailing List Analysis

Another source of social network information is mailing lists; most F/OSS projects make mailing lists archives freely downloadable online. Three different projects were selected and their mailing lists were analyzed for a four month period from August 1, 2004 to November 30, 2004. The mailing lists were the developers list from a well deployed database server, the general list for an text and file processing library, and the general list for a smaller F/OSS web browser.

A directed social network was constructed, establishing links if A sent a response message to B through the mailing list. The CDF of the social network size, as seen in figure 3, was found to be slightly larger than BlogLinks but still has a low median value and long tails. The average social network size is smaller than networks from Advogato not only because of the need for active communication between the participants, but also because of the focused nature of communications. Unlike weblogs, which allow communication on any topic, straying from the topic on a project mailing list is generally regarded as poor etiquette. Each message typically

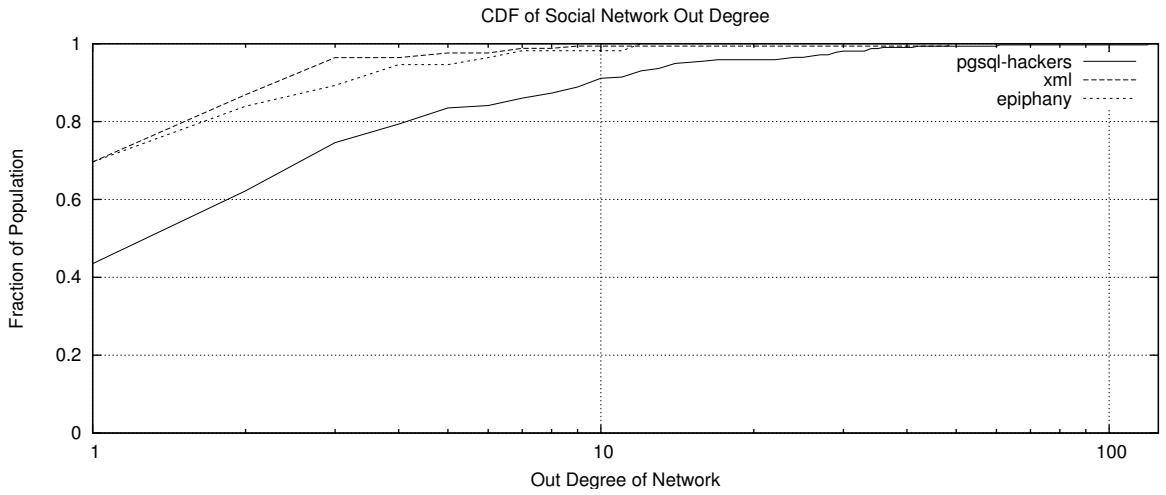

Figure 3: CDF of Social Network Size from Mailing List Analysis. The general form is similar to that of BlogLinks and Advogato.Org. This is the only set of focused communities in the analysis.

has a precise focus is frequently directed at a specific subset of readers, as opposed to weblogs which may be directed to the general public.

3.4 Aggregate Network Properties

These social network findings provide reasonable upper and lower bounds for social network size for open source developers. In our initial simulation, described below, we selected values as follows for a network of 50 agents. Maximum initial network size was set to 25 and minimum initial size was 1. Average network size was 8 and 75% of all agents had networks smaller than 10. In the future, we plan to use these data to establish a distribution of network sizes for our simulations. We also think that it may prove appropriate in some simulations to use different network sizes for different parts of a simulation, e.g., to reflect differences in reputation networks (larger) versus communication networks (smaller).

4 F/OSS Simulation

Using network size parameters derived from our empirical work described in the previous section, we constructed a multi-project simulation. Our goal for this simulation was to create a simple model that describes how developers choose what project to work on, and the global effects of these choices under conditions in which different projects have different levels of developer skill, and where features added by developers have the potential to interact negatively with other features.

We have created a model of rival F/OSS development called OSSim (pronounced Awesome!!). The model allows for multiple users, multiple developers, multiple projects, and various levels of developer socialization. Each user or developer is represented by a semi-autonomous agent with a set of problems that it wishes to solve. Problems are abstract concepts within the agent, for example "I need a text editor that can edit web pages" or "I'd like a game that I can play online with my Dad in Minnesota." The problem is then broken into a set of features and interaction between those features. If we examine the text editor problem, a user may find a feature rich

application useful, but only if it has good online help. Another user may want the application to remain as light weight as possible and does not want either additional features or in depth online help. Projects are represented as entities within the model that users can interact with. They contain the code for a project, and also facilitate some communication between developers on a project. In this sense they act as a "walled server" to exert control on a community.

In the model, agents have problems that they want to solve through the use of an existing piece of software or by "scratching an itch" and creating a new piece of software, as described by Raymond[13]. A problem is represented as a string of all of the possible features an interactions. This problem string is then passed to various candidate projects for an evaluation of how well they meet the requirements. Evaluation and problem representation is based on Kauffman's NK model[7]. The traditional NK model has two primary parameters: N is the number of features (elements) possible for the abstract problem, K is the feature interactions at that location. Evaluation of a string in the NK model results from taking each element and concatenating it with its K neighbors to create N substrings of length $K + 1$. Each of these substrings is then used as an index into a different lookup table (together these N tables represent the fitness landscape) that returns a value between 0 and 1. These values are then averaged over all N elements to give a final fitness between 0 and 1.

In the original NK model, each string was considered a unique organism that could mutate itself to obtain higher fitness. A value of K greater than 0 meant that this process was not strictly a hill climbing process because of the interaction. Within OSSim the change system is slightly different, because it is assumed that developer preferences change a slower rate than they can change the code, developers instead change the fitness landscape to mutate and improve fitness of the program. This is accomplished by randomly selecting a feature (out N possible) and then selecting one entry from within the table, that is used by other agents, and increasing the fitness. After this increase is made, the agent randomly selects another feature (the same or different) and randomly decreases the fitness the value of one entry in the table as a balancing effect. This process can be repeated multiple times depending on the skill of the agent. Distribution of agent skill is provided by the analysis of the different levels from Advogato.Org and research on interpersonal variation[12, 5].

At the end of each simulation period, the developers and users of a project vote on which set of changes they wish to keep. Users will vote yes for a set of changes if it improves their overall fitness and no if it doesn't. Each user votes on the changes from each developer and the set of changes with the most votes becomes the basis for the next time period. If a user has become unsatisfied with the project, because it is moving to slow or their changes have been rejected too often, the will start looking at other projects to fill their needs.

Agents become aware of other projects and agents through a socialization process. The initial social network is obtained through the empirical network analysis discussed in the previous section. That data provides rough estimates for the size and density of different social networks. The strength of each tie is currently randomly assigned. When an interaction takes place, there is a small probability that information about other users of the software, or other project may be transferred. As a developer learns of more users, they are more apt to make changes that are beneficial to those users. As users and developers learn of new projects, their choice of projects to solve a particular problem increases.

To test the robustness of this model, two different virtual experiments were run. The first experiment involved the perceived phenomenon of user drop off when a project looses the last of its developers. The second experiment involved the decrease in productivity and increase in switching between very complex and coupled projects.

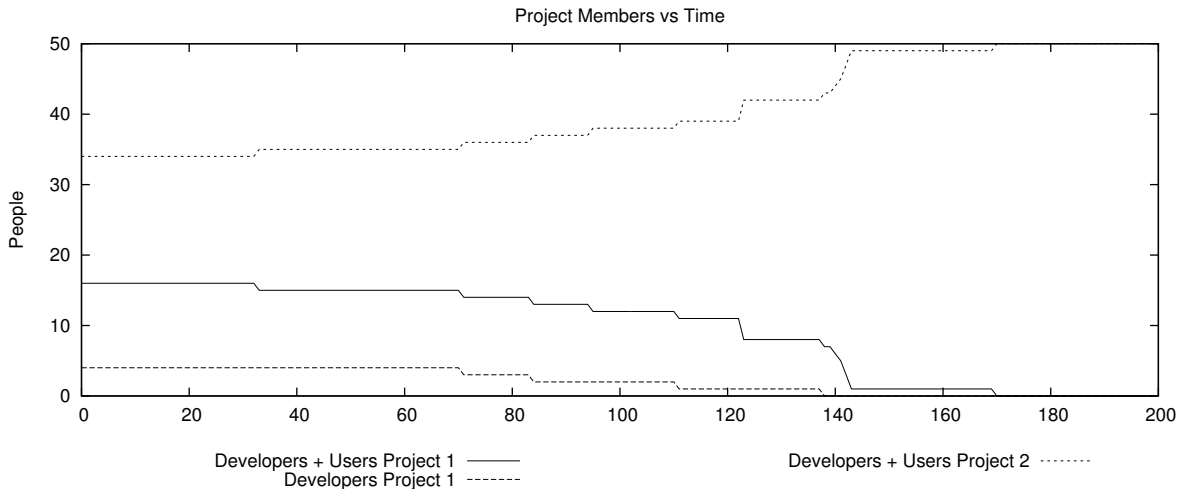

Figure 4: Illustrating User Drop-off with Unmaintained Software

4.1 Usage of Unmaintained Software

For this experiment, two identical projects were created and the a pool of agents was allocated between each project. One project was given developers of a high skill level while the other was given developers of a lower skill level. The social networks were originally set so the better developers had a larger social network, reflecting built up social capital. The simulation is considered a success if all users and developers end up working on the same project, having abandoned the project created by low skill developers. A typical result of this configuration can be seen in figure 4.

The model has been able to reflect what is known about software usage and reflects conventional wisdom about using software. In particular, this run illustrates a gradual loss of developers, with the last developer leaving at time 138. Shortly after this, at time 141, a large section of the user population switches to the other project. One lone user remains for an extended period of time, however even he leaves after a prolonged period of no software maintenance. The result is robust over a number of runs with the final developer or user usually leaving between time period 100 and 180. This suggests that lower developer skill may be a primary cause for project failure.

4.2 Understanding Coupling Effects

One of the key parameters regarding the development of a project is the degree of interaction, or coupling, between features, as represented by the parameter K in the model. At the most basic level, with $K = 0$, there is no feature interaction, and development proceeds as a simple hill climbing procedure. At the other extreme $K = N - 1$, the landscape is fully rugged and development becomes unpredictable. It was expected that as feature interaction increased, the overall progress on the project would decrease and the amount of switches between projects would increase. This would reflect the increased difficulty in writing highly coupled code, along with the user's frustration at the lack of progress from such issues.

To analyze the effects of this feature interaction, a virtual experiment was performed. In this experiment, four rival projects were created that each competed for the same pool of developers and users. All projects were created equal with elements having a default average fitness of 0.10 and developers and users were not given a preference for one project over another. The

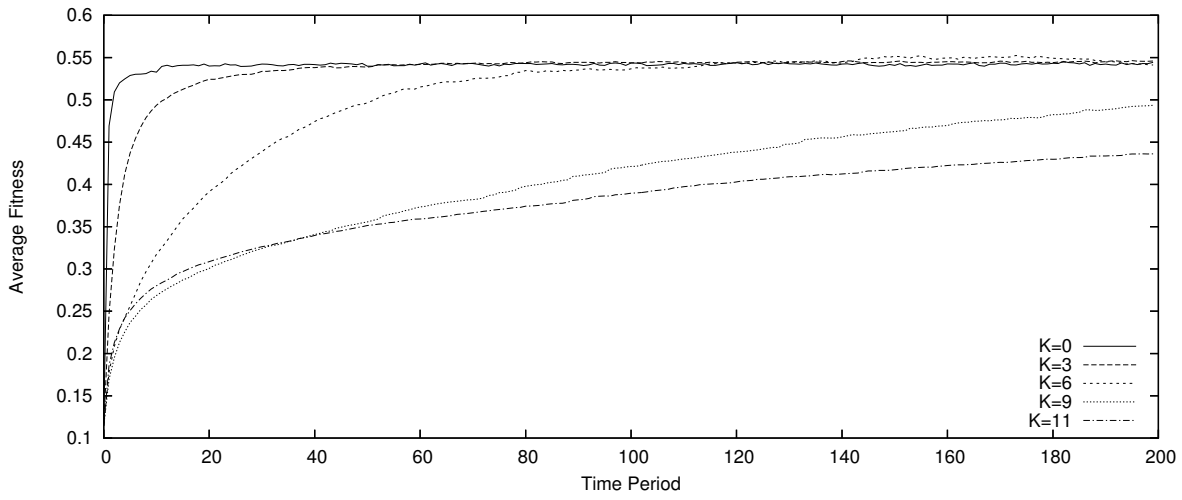

Figure 5: Average Fitness Across All Projects With Respect to Time and K (20 runs)

results of this experiment can be seen in figure 5 and the project size throughout the simulation is shown in figure 6.

The results show that as the coupling of the elements increases, the rate of fitness improvement decreases. This is echoed in commercial software development. Simple projects are easily developed and can be completed by a single person in a small time; large and highly coupled software projects are plagued by slow development as it becomes increasingly difficult to satisfy the highly coupled constraints.

5 Discussion

This research is a preliminary step in the creation of a robust model of F/OSS software development. We have been able to use a variety of different community data sources with different attributes to create a generalized social network for F/OSS. A middle ground between the data sets that overestimate social network size, such as Advogato, and the data sets that underestimate social network size, such as culling Blog information needs to be struck. It seems that in most cases this can be estimated by building social networks from mailing list data. This also has the advantage that it is the easiest to collect as many projects provide archives of data. This information has been combined with a simulation of rival project free software development. Agents are able to interact on the basis of social networks and find out about new projects through these social channels.

We have also uncovered interesting information about the social network structure inherent in F/OSS communities. In general there are more links between projects than originally thought, indicating a cohesive community. However, there are fewer links within communities, in particular the clique assumption does not hold according to the data we collected. It seems as though the distribution of network out degree is a highly skewed distribution with very long tails for a few developers who are able to work full time on F/OSS.

One area that is lacking in F/OSS simulation is that of validation methodologies. It can be difficult to validate a predictive model when there are so many different elements acting on it. Clearly the community needs to work together to create a set of standards for validation. A good place to start is with a set of attributes that are commonly seen in rival F/OSS development. Including some of the characteristics shown in this model, such as developer socialization,

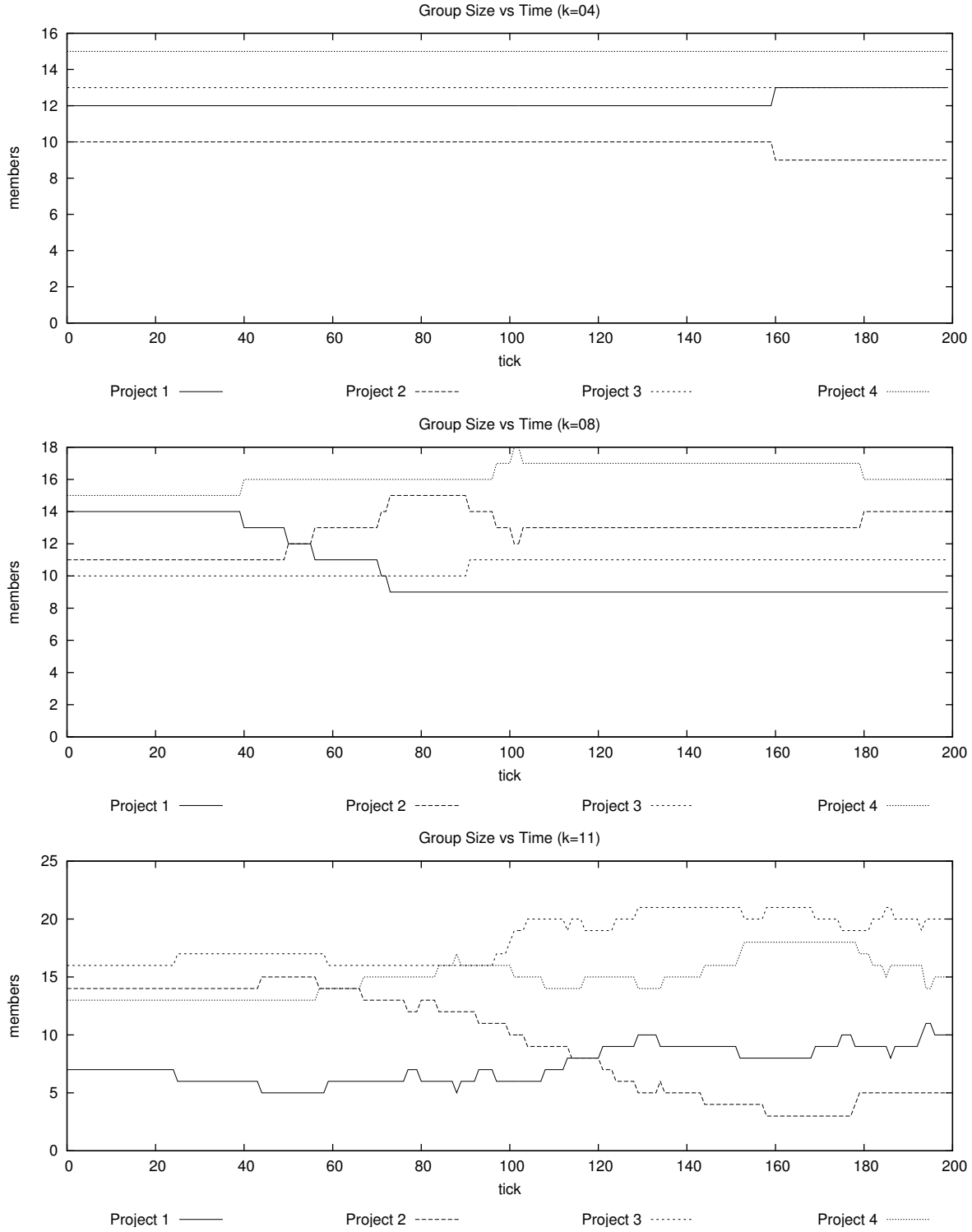

Figure 6: Project Size as K Varies (Examples from Individual Runs). A trend across all of the runs is that as complexity increases so does the rate at which agents switch projects. This is largely because the process is no longer a hill climbing process, instead there are a variety of local maxima that must be dealt with.

incomplete knowledge leading to suboptimal choices, user bodies following the developers, and coupling slowing down overall development.

Acknowledgements

We would like to thank Raph Levien for providing a copy of the Advogato.org database and Greg Madey, David Hinds, and Brad Malin for their helpful comments on a previous iteration of this model presented at NAACSOS 2004. This research was supported by the National Science Foundation IGERT Program grant no. 9972762, the National Science Foundation Graduate Research Fellowship Program, and the Carnegie Mellon University Center for the Computational Analysis of Social and Organizational Systems. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

References

[1] Jean-Michel Dalle and Paul David. SimCode: Agent-based simulation modeling of open-source software development. Technical Report 04-002, Stanford Institute for Economic Policy Research, 2004.

[2] Yongqin Gao, Vincent Freeh, and Greg Madey. Modeling and simulation of the OSS community. In *Seventh Annual Swarm Researchers Meeting*, April 2003.

[3] Yongqin Gao, Greg Madey, and Vincent Freeh. Modeling and simulation of a complex social system: A case study. http://www.nd.edu/∼oss/Papers/SimulationSym37_gao.pdf.

[4] Rishab Ghosh. Understanding free software developers: Findings from the floss study. In *Harvard Business School/MIT Sloan School Free/Open Source Software Conference: New Models of Software Development*, Cambridge, MA, June 2003.

[5] E. Eugene Grant and Harold Sackman. An exploratory investigation of programmer performance under on-line and off-line conditions. *IEEE Transactions on Human Factors in Electronics*, 8(1):33–48, March 1967.

[6] Guido Hertel, Sven Niedner, and Stefanie Herrmann. Motivation of software developers in open source projects: An internet-based survey of contributors to the linux kernel. *Research Policy*, 32:1159–1177, 2003.

[7] Stuart Kauffman. *The Origins of Order*. Oxford University Press, 1993.

[8] Karim R. Lakhani, Bob Wolf, Jeff Bates, and Chris DiBona. The boston consulting group hacker survey. http://www.osdn.com/bcg/, July 2002. Visited March 30, 2004.

[9] Raph Levien. Advogato.org. http://www.advogato.org/. Visited September 22, 2004.

[10] Greg Madey, Vincent Freeh, and Renee Tynan. Agent-based modeling of open source using swarm. In *Americas Conference on Information Systems*, pages 1472–1475, Dallas, TX, 2002.

[11] Greg Madey, Vincent Freeh, and Renee Tynan. The open source development phenomenon: An analysis based on social network theory. In *Eighth Americas Conference on Information Systems*, pages 1806–1813, Dallas, Texas, USA, 2002.

[12] Lutz Prechelt. The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really. Technical Report 1999-18, Universität Karlsruhe, December 1999.

[13] Eric Raymond. The cathedral and the bazaar. *First Monday*, 3(3), March 1998.

[14] Patrick Wagstrom. Toward a simulation model of open source software development. In *NAACSOS 2004*, Pittsburgh, PA, USA, June 2004.

[15] Jin Xu and Greg Madey. Exploration of the open source software community. In *NAACSOS 2004*, Pittsburgh, PA, USA, June 2004.