Understanding the Free/Libre Open Source Software (FLOSS) Development Process An Agent-Based Model

N.P. Radtke[1,2], M.A. Janssen[2,3], and J.S. Collofello[1]

ABSTRACT

Free/Libre Open Source Software (FLOSS) is the product of volunteers collaborating to build software in the public domain. The large number of FLOSS projects, combined with the data that is inherently archived with this online process, make studying this phenomenon attractive. Some FLOSS projects are very functional, well-known, and successful, such as Linux, the Apache web server, and Firefox. However, for every successful FLOSS project there may be 100's of projects that are unsuccessful. These projects fail to attract sufficient interest from developers and users and become inactive or abandoned before useful functionality is achieved. The goal of this research is to better understand the open source development process and gain insight into why some FLOSS projects succeed while others fail.

This paper presents a simple agent-based model of the FLOSS development process. In the model agents, as either developers or users, select from a landscape of FLOSS projects to be involved in. Via the selections that are made, and subsequent contributions, some projects are propelled to success while others are stagnant and inactive. The model is able to reproduce key characteristics observed in empirical FLOSS data and offers insight into how developers select FLOSS projects, ultimately causing certain projects to flourish while others are abandoned.

Key words: *open source, FLOSS, simulation, emergent properties, agent-based model, SourceForge*

1 School of Computing, Informatics, and Decision Systems Engineering, Ira A. Fulton School of Engineering, Arizona State University, P.O. Box 878809, Tempe, AZ 85287-8809, U.S.A.
2 Center for the Study of Institutional Diversity, Arizona State University, P.O. Box 872402, Tempe, AZ 85287-2402, U.S.A.
3 School of Human Evolution and Social Change, Arizona State University, P.O. Box 872402, Tempe, AZ 85287-2402, U.S.A.

INTRODUCTION

Although the concept of Free/Libre Open Source Software (FLOSS) has been around for many years, it has recently increased in popularity as well as received media attention, not without good reason. Certain characteristics of FLOSS are highly desirable: some FLOSS projects have been shown to be of very high quality (Linux Kernel Software 2004; Analysis of the Linux Kernel 2004) and to have low defect counts (Chelf 2006); FLOSS is able to exploit parallelism in the software engineering process, resulting in rapid development (Kogut and Metiu 2001); FLOSS sometimes violates Brooks' law (Rossi 2004), which states that "adding manpower to a late software product makes it later" (Brooks 1975); and FLOSS development thrives on an increasing user- and developer-base (Rossi 2004).

As open source has become a prominent player in the software market, more people and companies are faced with the possibility of using open source products, which often are seen as free or low-cost solutions to software needs (Wichmann 2002). However, choosing to use open source software is risky business, partly because it is unclear which FLOSS will succeed. To choose an open source project, only to find it stagnates or fails in the near future, could be disastrous, and is cited as a concern by IT managers (Smith 2002). Accurate prediction of a project's likelihood to succeed/fail would therefore benefit those who choose to use FLOSS, allowing more informed selection of open source projects.

This paper presents an agent-based model that simulates the development of open source projects. Findings from a diverse set of empirical studies of FLOSS projects have been used to formulate the model, which is then calibrated on empirical data from SourceForge[4], the largest online site hosting open source projects, and other sources of public FLOSS data. Such a model can be used for scenario and sensitivity analysis to explore the conditions necessary for the success of FLOSS projects.

BACKGROUND

There have been a limited number of attempts to simulate various parts of the open source development process (Dalle and David 2004). For example, Dalle and David (2004) use agent-based modeling to create SimCode, a simulator that attempts to model where developers will focus their contributions within a single project. However, in order to predict the success/failure of a single FLOSS project, other existing FLOSS projects, which are vying for a limited pool of developers and users, may need to be

4 http://www.SourceForge.net

considered. This is especially true when multiple FLOSS projects are competing for a limited market share (e.g., two driver projects for the same piece of hardware, or rival desktop environments such as GNOME and the KDE). Wagstrom, Herbsleb, and Carley (2005) created OSSim, an agent-based model containing users, developers, and projects that is driven by social networks. While this model allows for multiple competing projects, the published experiments include a maximum of only four projects (Wagstrom, Herbsleb, and Carley 2005), whereas SourceForge has 1000's of projects in competition. Preliminary work on modeling competition among projects is currently being explored by Katsamakas and Georgantzas (2007) using a system dynamics framework. By using a population of projects, it is possible to consider factors between the projects, e.g., the relative popularity of a project with respect to other projects as a factor that attracts developers and users to a particular project. Therefore, our model differs from others by attempting to simulate across a large landscape of FLOSS with agent-based modeling.

Gao, Madey, and Freeh (2005) approach modeling and simulating the FLOSS community via social network theory, focusing on the relationships between FLOSS developers. While they also use empirical data from the online FLOSS repository SourceForge to calibrate their model, they are mostly interested in replicating the network structure and use network metrics for validation purposes (e.g. network diameter and degree). Our model attempts to replicate other emergent properties of FLOSS development, such as the distribution of projects in development stages. However, both teams consider some similar indicators, such as the number of developers working on a project, when evaluating the performance of the models.

In addition, there have been attempts to identify factors that influence FLOSS. These have ranged from pure speculation (e.g., Raymond's (2000) gift giving culture postulates) to surveys of developers (e.g., Ghosh et al. 2002; Lakhani, Wolf, and Bates 2002) to case studies using data mined from SourceForge, freshmeat[5], etc. (e.g., Michlmayr 2005; Stewart, Ammeter, and Maruping 2006; Crowston and Scozzi 2002; Wang 2007; Stewart and Ammeter 2002). Wang (2007) demonstrates specific factors can be used for predicting the success of FLOSS projects via K-Means clustering. However, this form of machine learning offers no insight into the actual underlying process that causes projects to succeed. Gao, Huang, and Mady (2004) similarly use K-Means clustering but go a step further by trying to manually find rules to categorize projects. The research presented here approaches simulating the FLOSS development process using agent-based modeling in order to gain an understanding of the underlying

5 http://freshmeat.net is a website that tracks new releases and updates for software, with an emphasis on FLOSS. Both SourceForge and freshmeat are owned by Geeknet.

process that cannot be acquired through machine learning.

Smith, Capiluppi, and Fernández-Ramil (2006) create a model that is unusual in that it includes both developers and users. In this case, users are responsible for creating requirements that developers may or may not then implement. Smith, Capiluppi, and Fernández-Ramil note that the role users play in FLOSS has not been deeply explored, but in their model, the behavior of users clustering around new requirements is important in order to reproduce code growth patterns, including both continuous growth and growth spurts. It is unclear how significant users are to the FLOSS process, but most FLOSS models ignore them; therefore, our model includes users for the purpose of exploring their impact on open source development.

To encourage more simulation of the FLOSS development process, Antoniades et al. (2005) created a general framework for FLOSS models. The model presented here follows many of the recommendations and best practices suggested in this framework. In addition, Antoniades et al. (2005) developed an initial dynamical simulation model of FLOSS. Although the model presented here is agent-based, many of the techniques, including calibration, validation, and addressing the stochastic nature of the modeling process, are similar between the two models. One difference is the empirical data used for validation: Antoniades et al.'s (2005) model uses mostly code-level metrics from specific projects while the model presented here uses higher project-level statistics gathered across many projects.

IDENTIFYING AND SELECTING INFLUENTIAL FACTORS

Factors which are most likely to influence the success/failure of FLOSS must first be identified and then incorporated into the model. Many papers have been published in regards to this, but most of the literature simply speculates on what factors might affect the success and offers reasons why. Note that measuring the success of a FLOSS project is still an open problem: some metrics have been proposed and used but unlike for commercial software, no standards have been established (Crowston, Howison, and Annabi 2006; Koch 2009). Some possible success indicators are:

- Completion of the project (Crowston, Howison, and Annabi 2006)
- Progression through maturity stages (Crowston and Scozzi 2002)
- Number of developers (Crowston, Howison, and Annabi 2006; Capiluppi and Michlmayr 2007; Capiluppi, Lago, and Morisio 2003; Stewart and Ammeter 2002; Wang 2007)

- Level of activity (i.e., bug fixes, new features implemented, mailing list traffic) (Crowston, Howison, and Annabi 2006; Stewart, Ammeter, and Maruping 2006; Crowston and Scozzi 2002; Capiluppi, Lago, and Morisio 2003; Weiss 2005a; Mao, Vassileva, and Grassmann 2007; Wang 2007)
- Release frequency (Crowston, Howison, and Annabi 2006; English and Schweik 2007; Stewart, Ammeter, and Maruping 2006; Crowston and Scozzi 2002; Wang 2007; Capiluppi, Lago, and Morisio 2003; Stewart and Ammeter 2002)
- Project outdegree (Wang 2007)
- Active developer count change trends (Wang 2007)
- Developer satisfaction (Crowston, Howison, and Annabi 2006)
- User satisfaction (Crowston and Scozzi 2002)
- Popularity with users (i.e., number of users, number of downloads, frequency of use) (Crowston and Scozzi 2002; Wang 2007; Michlmayr 2005; Weiss 2005a; English and Schweik 2007)
- Number of subscribers to project news feeds (Stewart and Ammeter 2002)

Papers that consider factors influencing success fall into two categories: those that look at factors that directly affect a project's success (e.g., Michlmayr 2005; Stewart, Ammeter, and Maruping 2006; Smith and Sidorova 2003) and those that look for factors that attract developers to a project (and thus indirectly affect the success of a project) (e.g., Bitzer and Schröder 2005; Rossi 2004; Raymond 2000; Lerner and Tirole 2005). A few go a step further and perform statistical analyses to discover if there is a correlation between certain factors and a project's success/failure (Lerner and Tirole 2005; Michlmayr 2005), and Kowalczykiewicz (2005) uses trends for prediction purposes. Wang (2007) demonstrates that certain factors, such as distinct code committer counts and number of releases, can be used for accurate prediction using machine learning techniques. Koch (2008) considers the affect communication and coordination tools (e.g., bug trackers, mailing lists, version control systems) have on the efficiency of projects but is unable to reach a solid conclusion.

Factors affecting FLOSS projects include both technical and social factors. Technical factors are aspects that relate directly to a project and its development and are typically both objective and easy to measure. Examples of technical factors include lines of code and number of developers. Social factors pertain to aspects that personally motivate individuals to engage in open source development/use. Examples of social factors include reputation from working on a project, matching interests between the project

and the developer/user, popularity of the project with other developers/users, and perceived importance of the code being written (e.g., core versus fringe development (Dalle and David 2004)). Most of the social factors are subjective and rather difficult, if not impossible, to measure. Despite this, it is hard to deny that these might influence the success/failure of a project and therefore social factors are considered in the model. Fortunately, the social factors being considered fall under the domain of public goods, for which there is already a large body of work published (e.g., Ostrom, Gardner, and Walker 1994; Jerdee and Rosen 1974; Tajfel 1981; Axelrod 1984; Fox and Guyer 1977). Most of this work is not specific to FLOSS, but in general it explores why people volunteer to contribute to public goods and what contextual factors increase these contributions.

The findings of the public goods literature are applied when designing the model, as are findings from publications investigating how FLOSS works, extensive surveys of developers asking why they participate in FLOSS (e.g., Ghosh et al. 2002; Lakhani, Wolf, and Bates 2002), and comments and opinions of FLOSS users (e.g., Smith 2002).

MODEL DESCRIPTION

The model universe consists of agents and FLOSS projects. Agents may choose to contribute to or not contribute to, and to consume (i.e. download) or not consume FLOSS projects. At time zero, FLOSS projects are seeded in the model universe. These initial projects vary randomly in the amount of resources that will be required to complete them based on an exponential distribution, resulting in many small projects and few large projects. At any time, agents may belong to zero, one, or more than one of the FLOSS projects. The simulation is run with a time step (t) equal to one (40 hour) work week.

Table 1 contains the properties of agents. Table 2 contains the properties of projects.

Table 1: Agent properties.

Property	Description	Type/Range
Consumer number	Propensity of an agent to consume (use) FLOSS.	$\mathbb{R}$ [0.0, 1.0]
Producer number	Propensity of an agent to contribute to FLOSS.	$\mathbb{R}$ [0.0, 1.0]
Needs vector	A vector representing the interests of the agent.	Each scalar in vector is $\mathbb{R}$ [0.0, 1.0]
Resources number	A value representing the amount of work an agent can put into FLOSS projects on a weekly basis. A value of 1.0 represents 40 hours.	$\mathbb{R}$ [0.0, 1.5]
Memory	A list of projects the agent knows exist.	

Table 2: Project properties.

Property	Description	Type/Range
Current resources	The amount of resources or work being contributed to the project during the current time interval.	$\mathbb{R}$
Cumulative resources	The sum, over time increments, of all resources contributed to the project.	$\mathbb{R}$
Resources for completion	The total number of resources required to complete the project.	$\mathbb{R}$
Download count	The number of times the project has been downloaded.	$\mathbb{N}_0$
Maturity	Six ordered stages a project progresses through from creation to completion.	{planning, pre-alpha, alpha, beta, production/stable, mature}
Needs vector	An evolving vector representing the interests of the developers involved in the project.	Each scalar in vector is $\mathbb{R}$ [0.0, 1.0]

At each time step, agents choose to produce or consume based on their producer and consumer numbers, values between 0.0 and 1.0 that represent probabilities that an agent will produce or consume. Producer and consumer numbers are statically assigned when agents are created and are drawn from a normal distribution. If producing or consuming, an agent calculates a utility score for each project in its

memory, which contains a subset of all available projects. The utility function is shown in (1):

(1)

Each term in the utility function represents a factor that attracts agents to a project, where α_1 through α_5 are weights that control the importance of each factor, with $0.0 \leq \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5 \leq 1.0$ and $\alpha_1 = 1.5 \alpha_2 \alpha_3 = 1.0$. Factors were selected based on both FLOSS literature and our own understanding of the FLOSS development process. Keeping it simple, a linear utility equation is used for this version of the model. The first term represents the similarity between the interests of an agent and the direction of a project; it is calculated using cosine similarity between the agent's and project's needs vectors. The second term captures the current popularity of the project and the third term the size of the project implemented so far. The fourth term captures the popularity of a project with consumers based on the cumulative number of downloads a project has received. The fifth term captures the maturity stage of the project. Values with the subscript "norm" have been normalized by dividing each project's value by the maximum value over all projects. For example, norm is the ith project's download count divided by the maximum number of downloads that any project has received, as shown in (2):

where (2)

represents the number of downloads of the ith project

The discreet function δ maps each of the six development stages into a value between 0.0 and 1.0, corresponding to the importance of each development stage in attracting developers. To determine the importance of each stage, the number of new developers in each stage was counted, where new developers are defined as developers that have not been involved in any of a project's previous stages, and used as a proxy for the attractiveness of each stage. Data was collected from the FLOSSmole database (Howison, Conklin, and Crowston 2006) and through the use of CVSAnalY2 (Garcia-Campos 2009), a tool for converting SCM logs into relational database tables. Projects that had progressed through four or more development stages were selected, resulting in a dataset of 76 projects[6]. For each project, the new developer counts were

6 Since the goal was to determine the relative importance of all six development stages, it was necessary to select projects that had progressed through as many stages as possible. After removing projects that were listed in multiple stages at the same time, there were zero projects that had gone through six stages and only two that had gone through five stages, making it necessary to lower the

normalized by finding the stage with the maximum number of new developers and dividing all stages by this number, as shown in (3):

(3)

where

is the number of new developers for the ith project in the jth development stage

For each development stage, the normalized values were averaged across the 76 projects, resulting in a normalized importance value for each of the development stages. These values were used to define the discrete function as shown in (4):

where (4)

By definition, all the developers are considered new when a project is first released as open source, resulting in a high importance value for the planning stage. The data show that in general, fewer and fewer new developers join as a project matures. Unfortunately, in the dataset there were only four projects that progressed to the mature stage, and none of these projects gained new developers once in this stage. As a result of the small number of mature projects, the zero weight reported for this stage might not be accurate.

Since all terms in the utility function are normalized, the utility score is always a value between 0.0 and 1.0. Both consumers and producers use the same utility function. This is logical, as most FLOSS developers are also users of FLOSS (Crowston and Scozzi 2002; Raymond 2000). For consumers that are not producers, arguably the terms represented in the utility function are still of interest when selecting a project. There is relatively little research published on users compared to developers of FLOSS, so it is unclear if selection criteria are different between the two groups.

It is possible that some of the terms included in the utility function are redundant or irrelevant. Part of the model calibration process is to determine which of these factors are important. See the Model Calibration and Results sections below.

---

threshold to projects that had progressed through four or more stages in order to obtain a sufficient number of projects to study.

Agents use utility scores in combination with a multinominal logit equation to probabilistically select projects. The probability of selecting the ith project is shown in (5):

where (5)

is the probability of selecting the ith project

is the utility of the ith project

The multinominal logit allows for imperfect choice, i.e., not always selecting the projects with the highest utility, and may be adjusted by changing the parameter μ. When μ is 0, all projects are chosen with equal probability regardless of the utility. The larger the value of μ, the greater the chance an agent will select the project with the highest utility.

There is no explicit formulation of communication between agents included in the model; implicitly it is assumed that agents share information about other projects and thus agents know characteristics of projects they are not currently consuming/producing. At each time step, agents update their memory. With a certain probability an agent will be informed of a project and add it to its memory, simulating discovering new projects. Likewise, with a certain probability an agent will remove a project from its memory, simulating forgetting about or losing interest in old projects. Thus, over time an agent's memory may expand and contract.

Projects update their needs vector at each iteration using a decaying equation, as shown in (6):

(6)

(6a)

where (6b)

is the ith project's needs vector at time step

is the ith project's needs vector at time step

is the lth agent's contribution to the ith project at time step

is the lth agent's needs vector

The ith project's vector at time □, □□□□□□□□,□ , is partially based on the project's

vector at time $t-1$ (6a) and partially on the needs vectors of the agents currently contributing to the project (6b). The rate of decay is controlled by ϵ. An agent's influence on the project's vector is directly proportional to the amount of work the agent is contributing to the project with respect to other agents working on the same project at time t. This represents the direction of a project being influenced by the developers working on it, with core developers having a larger influence than peripheral developers in steering the project.

Finally, project maturity stages are computed based on percent complete threshold values. These values were determined by examining the percent of code commits that occurred in each development stage for projects on SourceForge that had progressed through four or more development stages. The dates projects changed stages was determined using the FLOSSmole database (Howison, Conklin, and Crowston 2006); CVSAnalY2 (Garcia-Campos 2009) was then used to build database tables from each project's SCM logs, which was subsequently queried to determine the number of commits in each stage. The mean percentage of commits that occur in each stage is shown in Fig. 1.

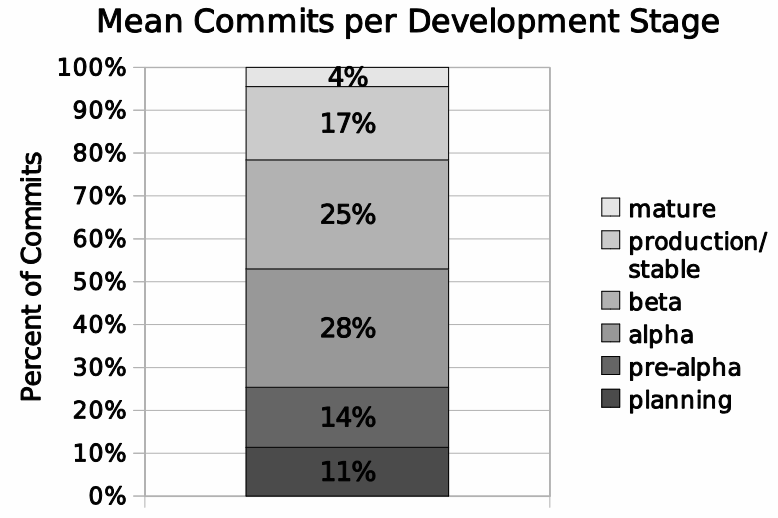

Figure 1: Mean percentage of code commits that occur in each development stage.

For example, a project in the model will be considered in the alpha stage if between 25% (i.e., *11+14*) and 53% (i.e., *11+14+28*) of the work has been completed.

VALIDATION METHOD

Creating a model that correctly predicts the success or failure of FLOSS projects is a complicated matter. To aid in the iterative development process, the model is first calibrated to reproduce a set of known, emergent properties from real world FLOSS data. The following three emergent properties were chosen to validate the model:

Distribution of projects in development stages: In December 2004, Weiss (2005b) crawled SourceForge and counted the number of projects in the planning, pre-alpha, alpha, beta, production/stable, and mature development stages. Using data from FLOSSmole, the authors of this paper found the distribution of projects in development stages to be very similar in June, 2009. Weiss's data is used to validate the model.

Number of developers per FLOSS project: Data is used from Weiss's (2005b) December 2004 crawl of SourceForge.

Number of FLOSS projects per developer: Data is used from a survey conducted by Ghosh et al. (2002) from February through April, 2002 of 2784 developers.

By creating a model that mimics a number of key patterns of empirical data, confidence is derived about the model.

MODEL CALIBRATION

The model has a number of parameters that must be assigned values. A subset of these can be set to likely values based on statistics gathered from surveys or mined from FLOSS repository databases. For the remaining parameters, a search of the parameter space must be performed to find the combination that allows the model to most closely match the empirical data. Due to the large state space, an exhaustive search is not possible. Since genetic algorithms are known to perform well in high dimension, stochastic, non-linear spaces (Kicinger, Arciszewski, and De Jong 2005), genetic algorithms are employed to find near-optimal parameter sets that result in the model's output closely matching the empirical data. This is done as follows: an initial population of 30 model parameter sets is created randomly. The model is run with each of the parameter sets and a fitness score is calculated based on the similarity of the generated versus empirical data. The parameter values from these sets are then mutated or crossed-over with other parameter sets to create a new generation of model parameter sets, with a bias for selecting parameters sets that resulted in a high fitness; then the new generation of parameter sets are evaluated and the process is repeated. This repetition continues until individuals meet a fitness threshold[7]. In this case, a

7 For the results presented in this paper, 2721 generations were evaluated.

genetic algorithm is being used for a stochastic optimization problem for which it is not known when a global optimum is found. Genetic algorithms are appropriate for finding well-performing solutions in a reasonably brief amount of time and frequently outperform other optimization techniques, such as random search and hill climbing, when applied to non-linear, chaotic, or stochastic landscapes (Kicinger, Arciszewski, and De Jong 2005). Reviewing the values of the best performing parameters will help identify which factors are important/influential in the open source software development process.

The fitness function chosen for the genetic algorithm is based on the sum of the square of errors between the simulated and empirical data, as shown in (7):

(7)

Since there are three fitness values calculated, one per empirical data set, the three fitness values are averaged to provide a single value for comparison purposes.

RESULTS

Since the model includes stochastic components, multiple runs with a given parameter set were performed and the results averaged. In this case, four runs were performed for each parameter set after initial experimentation showed very low standard deviations even with small numbers of runs. The averaged model results were then compared to the empirical data.

As empirical investigations of FLOSS evolution note, it takes approximately four years for a project of medium size to reach a mature stage (Krishnamurthy 2002). To accommodate large projects and allow the simulation sufficient time to stabilize from startup conditions, the model's performance was evaluated after 750 time steps, with a time step equal to one week. All metrics were gathered immediately following the 750th time step.

Matching distributions

The averaged data (over 4 runs) from the simulator's best parameter set, along with the empirical data, is shown in Figs. 2, 3, and 4.

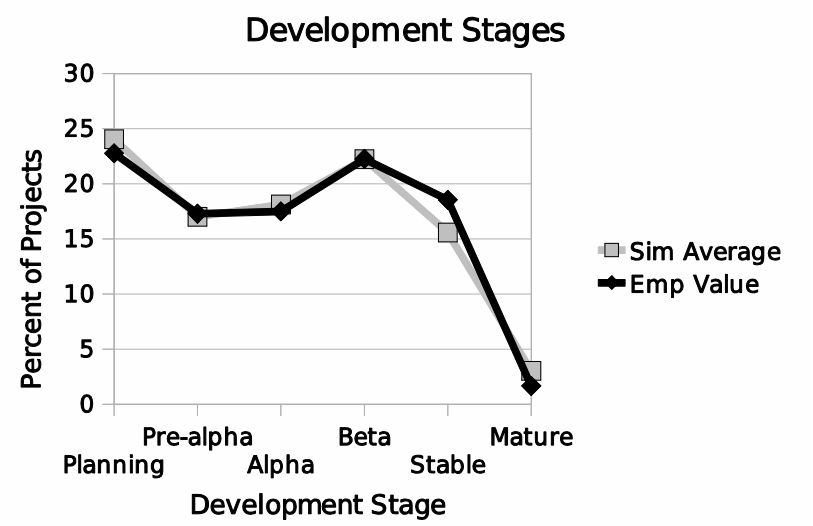

Figure 2: Percentage of FLOSS projects in development stages. Empirical data from (Weiss 2005b).

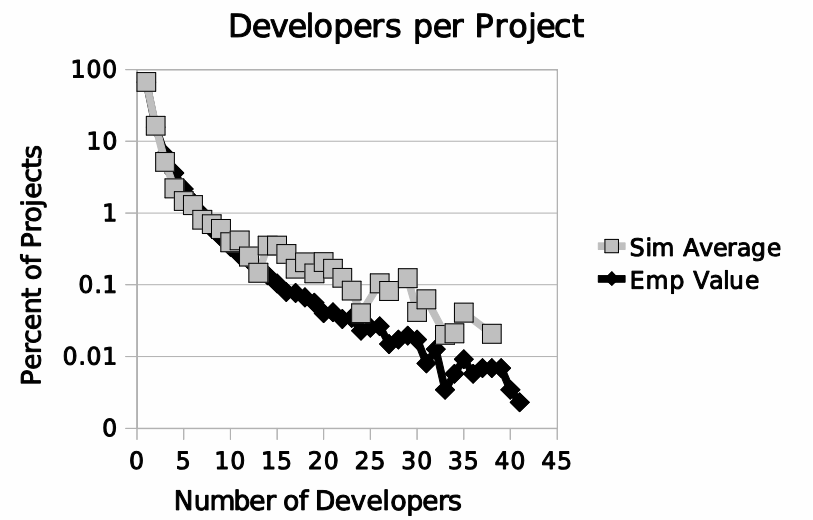

Figure 3: Percentage of projects with N developers. Empirical data from (Weiss 2005b).

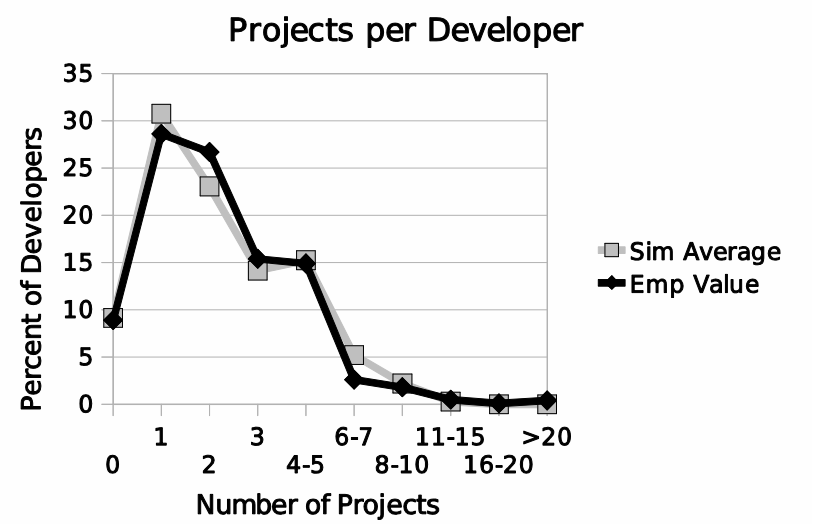

Figure 4**:** Percentage of developers working on N projects. Empirical data from (Ghosh et al. 2002).

Figure 2 shows the model's percentage of projects in each development stage is very similar to the empirical data. During the development of the model, this has been the hardest distribution to match. In earlier versions of the model, new projects were all created in the planning stage (Radtke, Janssen, and Collofello 2009). In reality, this is not the case; many projects, by the time they are released as open source on SourceForge, are beyond the planning stage. In fact, only 26.1% of projects start in the planning stage when they first appear on SourceForge, as shown in Table 3.

Table 3**:** Development stage of projects when first added to SourceForge, based on data mined from the FLOSSmole database (see Howison, Conklin, and Crowston 2006).

Initial Development Stage on SourceForge	Percent of Projects
planning	26.1%
pre-alpha	16.6%
alpha	18.1%
beta	22.4%
production/stable	15.6%
mature	1.2%

Mimicking this distribution when creating projects adds realism to the model and has helped match the empirical data.

Two thirds of FLOSS projects have only a single developer and 90% have fewer than four developers (Weiss 2005b). As shown in Fig. 3, the number of developers per projects follows a near-exponential distribution, and the simulated data is similar, especially for projects with fewer than 13 developers. Note that the data in Fig. 3 uses a logarithmic scale on the y-axis to help with a visual comparison between the two data sets. Beyond 13 developers, the values match less closely, although this difference is visually amplified as a result of the logarithmic scale and is actually not as large as it might initially appear. Since there are very few projects with large numbers of developers in the empirical data, the higher values may be in the noise anyhow and thus focus should be on the similarity of the lower numbers.

In FLOSS, the distribution of projects per developer is highly skewed, with the majority of developers working on one or just few projects (Ghosh and Prakash 2000; Ghosh et al. 2002). Meanwhile, a small subset of developers work on many projects (Ghosh et al. 2002; Krishnamurthy 2002). Figure 4 shows the model performs well in matching this distribution as well.

Table 4 contains the mean fitness scores for each of the emergent properties for the top performing parameter set.

Table 4**:** Mean fitness scores of the best performing parameter set for the three emergent properties.

Emergent Property	Fitness Score
Maturity stage	0.99936
Devs per project	0.99971
Projects per dev	0.99866
Combined (mean)	0.99924

These values provide a quantitative mechanism for confirming the visual comparisons made above. Indeed, all three fitness scores are high, indicating good matches for all three distributions. The combined fitness is simply the mean of the three fitness scores, although this value could be calculated with uneven weights if, say, matching each property was prioritized. Doing so would affect how the genetic algorithm explored the parameter space. It may be the case that certain properties are easy to reproduce in the model and work over a wide range of parameter sets, in which case these properties may be weighted less than properties that are more difficult to match. Properties which are always matched should be discarded from the model for evolution purposes as they do not discriminate against different parameter sets.

Analysis of the model runs show that none of the three distributions are trivial to match. To demonstrate this, 1000 random parameter sets were evaluated, representing a random set of points in the state space landscape. If a distribution is trivial to match, it will have high fitness scores over most or all of the state space; that is, it will be relatively non-discriminating and thus non-helpful in locating optima in the landscape. After inspecting the results, a fitness value of 0.98 is chosen as the threshold to distinguish between good and bad fit[8]. The range of values and the percent of fitness scores exceeding 0.98 are shown in Table 5.

8 A 0.98 fitness result is actually not a very good fit between the simulated and empirical data, but for this analysis it is preferable to error on the side of categorizing too many parameter sets as acceptable.

Table 5**:** Range of fitness scores and percent of scores exceeding 0.98 for a random sample of 1000 parameter sets.

	Fitness Score Range	Fitness Scores ≥ 0.98
Maturity stage	[0.4184, 0.9996]	45.9%
Devs per project	[0.7419, 0.9996]	18.4%
Projects per dev	[0.5576, 0.9967]	3.0%
Combined (mean)	[0.6532, 0.9904]	0.9%

The wide ranges of values demonstrate that the fitness values are not high over the entire state space; indeed, many parameters sets result in very poor matches for each of the distributions. The maturity stage fitness is greater than 0.98 for almost 46% of the random sample, indicating this may be the least discriminating of the three distributions. Although this value is high, the majority of the state space still results in poor fitness values, indicating matching this distribution helps calibrate the model. The developers per project and projects per developer have high fitness over a much smaller percent of the landscape, indicating these will be harder to match. The combined values exceed 0.98 for only 0.9% of the random parameter sets, indicating matching all three distributions at the same time is difficult. Based on this, it is concluded that all three distributions contribute in tuning the realism of the model.

Evolved parameters

Examining the evolved utility weights of the best performing 1% of parameter sets (i.e., the sets resulting in the highest combined fitness values) provides insight into what factors are important in the model for reproducing the three emergent properties examined. Using normal probability plots, it was determined that α_1, α_3, α_4, and α_5 are normally distributed. α_2 was marginal and may not follow a normal distribution. The averages and standard deviations for each of the weights from the parameter sets are contained in Table 6.

Table 6**:** Utility function weights from the top 1% performing (i.e., highest combined fitness) parameters sets.

Weight	Mean	Standard Deviation
(similarity)	0.0517	0.0433
(current resources)	0.2407	0.1415
(cumulative resources)	0.3500	0.1376
(downloads)	0.1869	0.1436
(maturity)	0.1708	0.0568

In general, the standard deviations are larger than desired, meaning a large range of values result in matching the empirical data and making it difficult to draw solid conclusions based on these values.

ω_1, the weight for the similarity between an agent and a project, is one of the more stable values. Surprisingly, the mean is rather low, indicating the interests of an individual may not be one of the most important factors in selecting a project. A possible explanation for this comes from Raymond's (2000) Linus' Law, which states that "Given enough eyeballs, all bugs are shallow." Similarly, with a large enough pool of heterogeneous developers, if a single developer finds a program interesting enough to write, there will be other developers that also find the program interesting. This results in similarity being less of a driving force because there is always a pool of individuals interested in any project, and thus it is not a discriminating factor in matching the empirical data.

ω_5, the maturity weight, is also a semi-stable value. The lower standard deviation adds confidence that approximately 17% of a project's perceived utility is based on the development stage of the project.

The three remaining weights, ω_2, ω_3, and ω_4, unfortunately all have large and similar variances, making it difficult to compare them to one another. It appears that current resources, cumulative resources, and downloads all play a significant role when individuals evaluate projects. With almost a 2 to 1 difference in values between ω_3 and ω_4, it is likely that the amount of work completed on a project is more important than the user interest in a project, as indicated by number of download.

Another interesting set of values evolved by the system are the parameters for the producer and consumer numbers. While the producer and consumer numbers are drawn from normal distributions bounded by 0.0 and 1.0 inclusive, neither the mean nor

standard deviations of these distributions are known. Therefore, these values are evolved to find the best performing values. Table 7 contains the evolved mean and standard deviation for the producer and consumer numbers averaged from the top 1% of parameter sets.

Table 7**:** Evolved producer/consumer number distributions parameters.

Producer/Consumer Number		Parameter statistics from the top 1% of parameter sets	
		Mean	Stdev
Producer number	Mean	0.9144	0.0066
	Stdev	0.0338	0.0145
Consumer number	Mean	0.3656	0.2728
	Stdev	0.3295	0.2045

Notice that the mean producer number is very high at 0.9144 and very stable across the top 1% of parameter sets, with a standard deviation of 0.0066. Likewise, the producer number standard deviation is low at 0.0338 and also stable with a standard deviation of 0.0145. This indicates that the top performing model runs had agents with high propensities to develop. In other words, having most agents produce frequently (i.e., most agents be developers) results in matching the empirical data. This is in alignment with the notion that FLOSS is a developer-driven process (Radtke and Janssen 2009; Crowston and Scozzi 2002; Raymond 2000; Lerner and Tirole 2005).

The evolved consumer number mean is much lower and the standard deviation is much higher compared to the producer number. Neither one of these parameters is particularly stable, i.e., both have large standard deviations over the top parameter sets. This indicates that the consumer number distribution has little effect on matching the empirical data. Recall that the evolved weight for downloads, α_4, while large enough to be considered influential, was also one of the smaller of the influential terms in the utility function. The conclusion is that consumers have some influence but are not the main driving force in matching the empirical data in the model.

Comparing success metrics

Although many success metrics have been proposed for FLOSS, it is unclear which are best or if there is even a significant difference among these metric. If there is a high correlation between certain metrics then the “easiest” metric, such as the metric that is most convenient to measure or collect, may be chosen over other metrics without affecting the results. To explore the similarities and differences, the following adaptations of proposed success metrics were considered:

Maturity threshold: Projects that are in the beta stage or higher are considered successful. This is based on the notion that projects that have reached the beta stage have produced useful software.

∆ **maturity:** In order to demonstrate satisfactory progress, projects must progress to a higher development stage every six months to be considered successful. Projects that are in the production/stable or mature stages are also considered successful, as these projects are at the upper limit and may remain active without ever moving stages.

∆ **developers:** The number of developers must increase every six months in order to demonstrate the vitality of the project.

∆ **downloads:** The number of downloads must increase every six months in order to demonstrate user interest in the project.

∆ **percent complete:** At least 10% of the project must be completed every six months in order for the project to be considered making satisfactory progress.

Completed projects: Projects that are complete are considered successful.

Some of these metrics are inherently difficult to measure for real projects, especially considering the amount of dirty data from sites like SourceForge or included in the FLOSSmole database. Therefore, the model is used to draw inferences about the real data. These particular success metric were chosen because they mapped into the model and therefore were feasible to collect. The six month time limit used in several of the metrics was chosen as a reasonable limit based on existing literature. As per Raymond's (2000) recommendation, open source projects should release early and release often in order to be successful. English and Schweik (2007) use a similar timeframe to the one chosen here, classifying projects as successes or failures based on if a version is released within one year. Similarly, Wang (2007) believes successful projects will release a stable version within six months and average at least one release per year. Therefore, the six month timeframe in the above metrics seems to be in a range that is generally accepted by other FLOSS researchers.

Using the parameter set that produced the highest fitness, the percentage of projects at the end of a run meeting each of the selected success criteria was calculated. The results are shown in Table 8.

Table 8**:** Percentage of successful projects based on different success metrics.

Success Metric	Successful Projects
Maturity threshold	39.83%
∆ maturity	18.02%
∆ developers	19.17%
∆ downloads	9.93%
∆ percent complete	0.01%
Completed projects	1.41%

From these results, it is clear that there are significant differences among the varying success metrics. The least discriminating metric, maturity threshold, categorizes almost 40% of the projects as successful while the ∆ percent complete metric considers almost none of the projects successful. What is not clear is how much overlap there is among the different metrics. Are most of the projects categorized as successful by one metric also considered successful by other metrics? To consider the overlap among sets of successful projects, the Jaccard similarity is calculated between each pair of metrics. The similarity values between sets are surprisingly small, indicating there is minimal overlap across the various metrics, as shown in Table 9.

Table 9**:** Jaccard similarity between different sets of successful projects.

	Maturity threshold	∆ maturity	∆ developers	∆ downloads	∆ percent complete	Completed projects
Maturity threshold	1.00	0.44	0.15	0.10	0.00	0.04
∆ maturity	0.44	1.00	0.1	0.09	0.00	0.08
∆ developers	0.15	0.10	1.00	0.18	0.00	0.00
∆ downloads	0.10	0.09	0.18	1.00	0.00	0.03
∆ percent complete	0.00	0.00	0.00	0.00	Undef.	0.00
Completed projects	0.04	0.08	0.00	0.03	0.00	1.00

This indicates that the proper choice of metric(s) is important as the metrics themselves differ. That is, “successful” projects may not be successful according to all metrics. It has been suggested that multiple metrics be used in calculating if a project is successful (Crowston, Howison, and Annabi 2006), as this may result in a more balanced evaluation process.

DISCUSSION AND FUTURE WORK

Once developers join a project, it is likely that they will continue to work on the same project in the future. This is especially evident in the case of core developers, who typically work on a project for an extended period of time. Currently, the model attempts to reproduce this characteristic by giving a boost (taking the square root) of the utility function for projects worked on in the previous time step. In effect, this increases the probability of an agent selecting the same projects to work on in the subsequent time step. Improvements to the model might include adding a switching cost term to the utility function, representing the extra effort required to become familiar with another project. Gao, Madey, and Freeh (2005) address this issue in their FLOSS model by using probabilities based on data from SourceForge to determine when developers continue with or leave a project they are currently involved with.

The model's needs vectors serve as an abstraction for representing the interests and corresponding functionalities of the agents and projects respectively. Therefore, the needs vector is at the crux of handling the matching of developers' interests with appropriate projects. For simplicity, initial needs vector values are assigned via a uniform distribution, but exploration of the effects of other distributions may be interesting. For example, if a normal distribution is used, projects with vector components near the mean will have an easy time attracting agents with similar interests. Projects with vector components several standard deviations from the mean may fail to attract any agents. A drawback of a normal distribution is that it makes most projects similar; in reality, projects are spread over a wide spectrum (e.g., from operating systems and drivers to business applications and games), although the actual distribution is unknown and difficult to measure. Changing the distribution used to generate the needs vectors likely would affect θ_1, the evolved similarity weight in the utility function.

Currently, needs vectors for projects and agents are generated independently. This has the problem of creating projects which are of no interest to any agents. An improvement would be to have agents create projects; when created, a project would clone its associated agent's needs vector (which would then evolve as other agents joined and contributed to the project). This behavior would more closely match SourceForge, where a developer initially registers his/her project. By definition, the project matches the developer's interest at time of registration.

For simplicity's sake, currently the model uses a single utility function for both producers

and consumers. It is possible that these two groups may attach different weights to factors in the utility function or may even have two completely different utility functions. However, analysis of the model shows that developers are the main driving force to reproduce the empirical data, with consumers providing a lesser influence. Exploration of a simplified model without consumers may show that concerns about using multiple utility functions are irrelevant.

One final complication with the model is its internal representations versus reality. For example, a suggested strategy for success in open source projects is to release early and release often (Raymond 2000). Using this method to determine successful projects within the model is problematic because the model includes no concept of releasing versions of software. Augmenting the model to include a reasonable representation of software releases is non-trivial, if possible at all. Likewise, it is difficult to compare findings of other work on conditions leading to success that map into this model. For example, Lerner and Tirole (2005) consider licensing impacts while Michlmayr (2005) considers version control systems, mailing lists, documentation, portability, and systematic testing policy differences between successful and unsuccessful projects. Unfortunately, none of these aspects easily map into the model for comparison or validation purposes.

CONCLUSION

A better understanding of conditions that contribute to the success of FLOSS projects might be a valuable contribution to the future of software engineering. An agent-based model was created to explore these conditions. The model is formulated from empirical studies and calibrated using multiple sources of FLOSS data. The calibrated version reproduces distributions that closely match the three emergent properties examined. From the calibrated data, it is concluded that the current resources going towards a project, the resources a project has already accumulated, the number of downloads a project has received, and the maturity of a project are all important factors. Surprisingly, the similarity between an agent and a project is not important in matching the empirical data. The notion that FLOSS is more of a developer-driven than user-driven process is also supported. Finally, different definitions of success are explored. It is concluded that different formulations of success result in different sets of projects being considered successful, with minimal overlap between pairs of sets.

The model presented here aids in gaining a better understanding of the conditions necessary for open source projects to succeed. With further iterations of development, including supplementing the model with better data-based values for parameters and

adding additional emergent properties for validation purposes, the model could move into the realm of prediction. In this case, it would be possible to feed real-life conditions into the model and then observe a given project as it progresses (or lack of progresses) in the FLOSS environment.

References

Analysis of the Linux Kernel. 2004. Analysis of the Linux Kernel. Research report. Coverity Incorporated.

Antoniades, Ioannis P., Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis, and Georgios L. Bleris. 2005. “Dynamical simulation models of the Open Source Development process.” In *Free/Open Source Software Development*, edited by Stefan Koch, 174–202. Hershey, PA: Idea Group, Incorporated.

Axelrod, Robert. 1984. *The Evolution of Cooperation.* New York: Basic Books.

Bitzer, Jurgen, and Philipp J.H. Schröder. 2005. “Bug-fixing and code-writing: The private provision of open source software.” *Information Economics and Policy* 17 (3): 389–406 (July).

Brooks, Frederick P. 1975. *The Mythical Man-month: Essays on Software Engineering.* Reading, MA: Addison-Wesley.

Capiluppi, Andrea, Patricia Lago, and Maurizio Morisio. 2003, May 3–11,. “Evidences in the Evolution of OS Projects through Changelog Analyses.” Edited by Joseph Feller, Brian Fitzgerald, Scott Hissam, and Karim Lakhani, *Taking Stock of the Bazaar: Proceedings of the 3rd Workshop on Open Source Software Engineering.* Portland, OR, USA, 19–24.

Capiluppi, Andrea, and Martin Michlmayr. 2007, June 11–14,. “From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects.” Edited by Joseph Feller, Brian Fitzgerald, Walt Scacchi, and Alberto Silitti, *Open Source Development, Adoption and Innovation.* International Federation for Information Processing, Limerick, Ireland: Springer, 31–44.

Chelf, Ben. 2006. Measuring Software Quality: a Study of Open Source Software. Research report. Coverity Incorporated.

Crowston, Kevin, James Howison, and Hala Annabi. 2006. “Information Systems Success in Free and Open Source Software Development: Theory and Measures.” *Software Process: Improvement and Practice* 11 (2): 123–148 (March/April).

Crowston, Kevin, and Barbara Scozzi. 2002. “Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development.” *IEE Proceedings*

Software, Volume 149. 3–17.

Dalle, Jean-Michel, and Paul A. David. 2004, November 1,. “SimCode: Agent-based Simulation Modelling of Open-Source Software Development.” Industrial organization, EconWPA.

English, Robert, and Charles M. Schweik. 2007. “Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of SourceForge.net Projects.” *FLOSS ’07: Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development.* Washington, DC, USA: IEEE Computer Society, 11.

Fox, John, and Melvin Guyer. 1977. “Group Size and Others’ Strategy in an N-Person Game.” *Journal of Conflict Resolution* 21 (2): 323–338 (June).

Gao, Yongqin, Yingping Huang, and Greg Mady. 2004, June. “Data Mining Project History in Open Source Software Communities.” *NAACSOS 2004*. North American Association for Computational Social and Organization Sciences, Pittsburg, PA, USA.

Gao, Yongqin, Greg Madey, and Vince Freeh. 2005, April. “Modeling and Simulation of the Open Source Software Community.” *Agent-Directed Simulation Symposium, SpringSim’05*. Society for Modeling and Simulation International, San Diego, CA, 113–122.

Garcia-Campos, Carlos. 2009, April. *The CVSAnalY Manual*. 2.0.0. LibreSoft.

Ghosh, Rishab Aiyer, Bernhard Krieger, Ruediger Glott, and Gregorio Robles. 2002, June. “Part 4: Survey of Developers.” In *Free/Libre and Open Source Software: Survey and Study.* Maastricht, The Netherlands: University of Maastricht, The Netherlands.

Ghosh, Rishab Aiyer, and Vipul Ved Prakash. 2000. “The Orbiten Free Software Survey.” *First Monday* 5, no. 7 (July 3,).

Howison, James, Megan Conklin, and Kevin Crowston. 2006. “FLOSSmole: A collaborative repository for FLOSS research data and analyses.” *International Journal of Information Technology and Web Engineering* 1 (July): 17–26.

Jerdee, Thomas H., and Benson Rosen. 1974. “Effects of Opportunity to Communicate and Visibility of Individual Decisions on Behavior in the Common Interest.” *Journal of Applied Psychology* 59 (6): 712–716.

Katsamakas, Evangelos, and Nicholas Georgantzas. 2007. “Why Most Open Source Development Projects Do Not Succeed?” *FLOSS ’07: Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and*

Development. Washington, DC, USA: IEEE Computer Society, 3.

Kicinger, Rafal, Tomasz Arciszewski, and Kenneth A. De Jong. 2005. "Evolutionary computation and structural design: A survey of the state of the art." *Computers and Structures* 83 (23-24): 1943–1978.

Koch, Stefan. 2008. "Exploring the effects of SourceForge.net coordination and communication tools on the efficiency of open source projects using data envelopment analysis." In *Empirical Software Engineering*, edited by Sandro Morasca. Springer.

_____. 2009. "Exploring the Effects of SourceForge.net Coordination and Communication Tools on the Efficiency of Open Source Projects using Data Envelopment Analysis." *Empirical Software Engineering* 14 (4): 397–417.

Kogut, Bruce, and Anca Metiu. 2001. "Open-Source Software Development and Distributed Innovation." *Oxford Review of Economic Policy* 17 (2): 248–264 (Summer).

Kowalczykiewicz, Krzysztof. 2005. "Libre projects lifetime profiles analysis." *Free and Open Source Software Developers' European Meeting 2005*. Brussels, Belgium.

Krishnamurthy, Sandeep. 2002. "Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects." *First Monday* 7, no. 6 (June).

Lakhani, Karim R., Bob Wolf, and Jeff Bates. 2002. The Boston Consulting Group Hacker Survey, Release 0.3. Online. http://flosscom.net/index.php?option=com_docman\&task= doc_view\&gid=45.

Lerner, Josh, and Jean Tirole. 2005. "The Scope of Open Source Licensing." *Journal of Law, Economics, and Organization* 21 (1): 20–56 (April).

Linux Kernel Software. 2004. Linux Kernel Software Quality and Security Better Than Most Proprietary Enterprise Software, 4-Year Coverity Analysis Finds. Press release. Coverity Incorporated.

Mao, Yan, Julita Vassileva, and Winfried Grassmann. 2007. "A System Dynamics Approach to Study Virtual Communities." *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences.* Washington, DC, USA: IEEE Computer Society, 178a.

Michlmayr, Martin. 2005. "Software Process Maturity and the Success of Free Software Projects." Edited by Krzysztof Zielinski and Tomasz Szmuc, *Software Engineering: Evolution and Emerging Technologies.* Krakow, Poland: IOS Press, 3–14.

Ostrom, Elinor, Roy Gardner, and James Walker. 1994. *Rules, Games and Common*

Pool Resources. Ann Arbor, MI: University of Michigan Press.

Radtke, Nicholas Patrick, and Marco A. Janssen. 2009. “Consumption and Production of Digital Public Goods: Modeling the Impact of Different Success Metrics in Open Source Software Development.” *International Journal of Intelligent Control and Systems* 14, no. 1 (March).

Radtke, Nicholas Patrick, Marco A. Janssen, and James Samuel Collofello. 2009. “What Makes Free/Libre Open Source Software (FLOSS) Projects Successful? An Agent-Based Model of FLOSS Projects.” *International Journal of Open Source Software and Processes* 1 (2): 1–13 (April–June).

Raymond, Eric Steven. 2000, September 11,. “The Cathedral and the Bazaar.” Technical Report 3.0, Thyrsus Enterprises.

Rossi, Maria Alessandra. 2004, April. “Decoding the “Free/Open Source(F/OSS) Software Puzzle” A Survey of Theoretical and Empirical Contributions.” Quaderni 424, Dipartimento di Economia Politica, Università degli Studi di Siena.

Smith, Neil, Andrea Capiluppi, and Juan Fernández-Ramil. 2006, May 20–21,. “Users and Developers: An Agent-Based Simulation of Open Source Software Evolution.” Edited by Qing Wang, Dietmar Pfahl, David M. Raffo, and Paul Wernick, SPW/ProSim, Volume 3966 of *Lecture Notes in Computer Science.* Shanghai, China: Springer, 286–293.

Smith, Shobha C., and Anna Sidorova. 2003. “Survival of Open-Source Projects: A Population Ecology Perspective.” *ICIS 2003. Proceedings of International Conference on Information Systems 2003.* Seattle, WA.

Smith, T. 2002, October 1,. Open Source: Enterprise Ready – With Qualifiers. theOpenEnterprise. http://www.theopenenterprise.com/story/TOE20020926S0002.

Stewart, Katherine J., and Anthony P. Ammeter. 2002, December 15–18,. “An Exploratory Study of Factors Influencing the Level of Vitality and Popularity of Open Source Projects.” Edited by L. Applegate, R. Galliers, and J DeGross, *Proceedings of the 23rd International Conference on Information Systems.* Barcelona, Spain, 853–857.

Stewart, Katherine J., Anthony P. Ammeter, and Likoebe M. Maruping. 2006. “Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects.” *Information Systems Research* 17 (2): 126–144 (June).

Tajfel, Henri. 1981. *Human Groups and Social Categories: Studies in Social Psychology.* Cambridge, UK: Cambridge University Press.

Wagstrom, Patrick, Jim Herbsleb, and Kathleen Carley. 2005, July 11–15,. “A Social Network Approach to Free/Open Source Software Simulation.” Edited by Marco Scotto and Giancarlo Succi, *First International Conference on Open Source Systems.* Genova, Italy, 16–23.

Wang, Yu. 2007, Spring. “Prediction of Success in Open Source Software Development.” Master of science dissertation, University of California, Davis, Davis, CA.

Weiss, Dawid. 2005a. “Measuring Success of Open Source Projects Using Web Search Engines.” Edited by Marco Scotto and Giancarlo Succi, *Proceedings of the First International Conference on Open Source Systems (OSS 2005).* Genova, Italy, 93–99.

_____. 2005b. “Quantitative Analysis of Open Source Projects on SourceForge.” Edited by Marco Scotto and Giancarlo Succi, *Proceedings of The First International Conference on Open Source Systems (OSS 2005).* Genova, Italy, 140–147.

Wichmann, Thorsten. 2002, July. “Part 2: Firms’ Open Source Activities: Motivations and Policy Implications.” *FLOSS Final Report: Free/Libre Open Source Software: Survey and Study.* Berlecon Research, Berlin, Germany.