What Makes Free/Libre Open Source Software (FLOSS) Projects Successful? An Agent-Based Model of FLOSS Projects

Nicholas P. Radtke, Arizona State University, USA

Marco A. Janssen, Arizona State University, USA

James S. Collofello, Arizona State University, USA

ABSTRACT

The last few years have seen a rapid increase in the number of Free/Libre Open Source Software (FLOSS) projects. Some of these projects, such as Linux and the Apache web server, have become phenomenally successful. However, for every successful FLOSS project there are dozens of FLOSS projects which never succeed. These projects fail to attract developers and/or consumers and, as a result, never get off the ground. The aim of this research is to better understand why some FLOSS projects flourish while others wither and die. This article presents a simple agent-based model that is calibrated on key patterns of data from SourceForge, the largest online site hosting open source projects. The calibrated model provides insight into the conditions necessary for FLOSS success and might be used for scenario analysis of future developments of FLOSS. [Article copies are available for purchase from InfoSci-on-Demand.com]

Keywords: Agent-Based Model; Emergent Properties; FLOSS; Open Source; Prediction Success; Simulation

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 (*Analysis of the Linux Kernel*, 2004; *Linux Kernel Software*, 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 & 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. 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 (T. 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 article presents an initial step towards the development of 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, the largest online site hosting open source projects. 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 & 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 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 et al., 2005). 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 pioneers new territory 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 without including the complexities of social networking. 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 (Raymond's (2000) gift giving culture postulates) to surveys of developers (Rossi, 2004) to case studies using data mined from SourceForge (Michlmayr, 2005). 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. Therefore, the research presented here approaches simulating the FLOSS development process using agent-based modeling instead of machine learning.

To encourage more simulation of the FLOSS development process, Antoniades, Samoladas, Stamelos, Angelis, and Bleris (2005) created a general framework for FLOSS models. The model presented here follows some 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. Some possible success indicators are:

- Completion of the project (Crowston, Howison, & Annabi, 2006)
- Progression through maturity stages (Crowston & Scozzi, 2002)
- Number of developers
- Level of activity (i.e., bug fixes, new feature implementations, mailing list)
- Time between releases
- Project outdegree (Wang, 2007)
- Active developer count change trends (Wang, 2007)

English and Schweik (2007) asked eight developers how they defined success and failure of an open source project. Answers varied for success, but all agreed that a project with a lack of users was a failure. Thus having a sufficient user-base may be another metric for success.

Papers that consider factors influencing success fall into two categories: those that look at factors that directly affect a project's success (Michlmayr, 2005; Stewart, Ammeter, & Maruping, 2006; S. C. Smith & Sidorova, 2003) and those that look for factors that attract developers to a project (and thus indirectly affect the success of a project) (Bitzer & Schröder, 2005; Rossi, 2004; Raymond, 2000; Lerner & 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 & Tirole, 2005; Michlmayr, 2005), and Kowalczykiewicz (2005) uses trends for prediction purposes. Wang (2007) demonstrates that certain factors can be used for accurate prediction using machine learning techniques. Koch (2008) considers factors affecting efficiency after first using data envelopment analysis to show that successful projects tend to have higher efficiencies.

In general, factors affecting FLOSS projects fall into two categories: technical factors 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.

The second category is social factors. 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 & 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, & Walker, 1994; Jerdee & Rosen, 1974; Tajfel, 1981; Axelrod, 1984; Fox & 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 this 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, Krieger, Glott, & Robles, 2002), and comments and opinions of FLOSS users (e.g., T. Smith, 2002).

INITIAL MODEL

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. 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) workweek.

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

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 Box 1.

Each term in the utility function represents a weighted factor that attracts agents to a project, where w_1 through w_5 are weights that control the importance of each factor, with $0.0 \leq w_1, w_2, w_3, w_4, w_5 \leq 1.0$ and $\Sigma_{i=1}^{5} w_i = 1.0$. 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 direc-

Table 1. Agent properties

Property	Description	Type/Range
Consumer number	Propensity of an agent to consume (use) FLOSS.	Real [0.0, 1.0]
Producer number	Propensity of an agent to contribute to (develop) FLOSS.	Real [0.0, 1.0]
Needs vector	A vector representing the interests of the agent.	Each scalar in vector is real [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.	Real [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.	Real
Cumulative resources	The sum, over time increments, of all resources contributed to the project.	Real
Resources for completion	The total number of resources required to complete the project.	Real
Download count	The number of times the project has been downloaded.	Integer
Maturity	Six ordered stages a project progresses through from creation to completion.	{planning, pre-alpha, alpha, beta, stable, mature}
Needs vector	An evolving vector representing the interests of the developers involved in the project.	Each scalar in vector is real [0.0, 1.0]

Box 1.

$$\begin{aligned} utility &= w_1 \cdot \mathrm{similarity}\left(agentNeeds, projectNeeds\right) \\ &+ w_2 \cdot currentResources_{\mathrm{norm}} \\ &+ w_3 \cdot cumulativeResources_{\mathrm{norm}} \\ &+ w_4 \cdot downloads_{\mathrm{norm}} \\ &+ w_5 \cdot \mathrm{f}\left(maturity\right) \end{aligned} \tag{1}$$

tion of a project; it is currently 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 (e.g., $downloads_{\mathrm{norm}}$ is a project's download count divided by the maximum number of downloads that any project has received). The discreet function $^{\mathrm{f}}$ maps each of the six maturity stages into a value between 0.0 and 1.0, corresponding to the importance of each maturity stage in attracting developers. Since all terms 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. 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 exploration is to determine which of these factors are relevant. See the Calibrating the Model and Results sections below.

Agents use utility scores in combination with a multinominal logit equation to probabilistically select projects. The multinominal logit allows for imperfect choice, i.e., not always selecting the projects 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, where the new vector is partially based on the project's previous vector and partially on the needs vectors of the agents currently contributing to the project. 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. This represents the direction of a project being influenced by the developers working on it. Finally, project maturity stages are computed based on percent complete threshold values.

VALIDATION METHOD

Creating a model that successfully 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. For example, Weiss (2005) surveyed the distribution of projects at SourceForge in each of six development categories: planning, pre-alpha, alpha, beta, stable, and mature. Therefore, the model will need to produce a distribution of projects in each stage similar to that measured by Weiss. In addition, two other emergent properties were chosen to validate the initial model:

- Number of developers per FLOSS project.
- Number of FLOSS projects per developer.
- By creating a model that mimics a number of key patterns of the data, confidence is derived about the model.

CALIBRATING THE MODEL

The model has a number of parameters that must be assigned values. A small 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. Since an exhaustive search is not practical, the use of genetic algorithms from evolutionary computation is used to explore the parameter space (Kicinger, Arciszewski, & De Jong, 2005). This is done as follows: an initial population of 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 repeated. In this case, a 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. 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 Box 2.

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). Thus, the model's performance was evaluated by running the model for 250 time steps, with a time step of one week, for a total simulated time equivalent of a little over five years. All metrics were gathered immediately following the 250th time step.

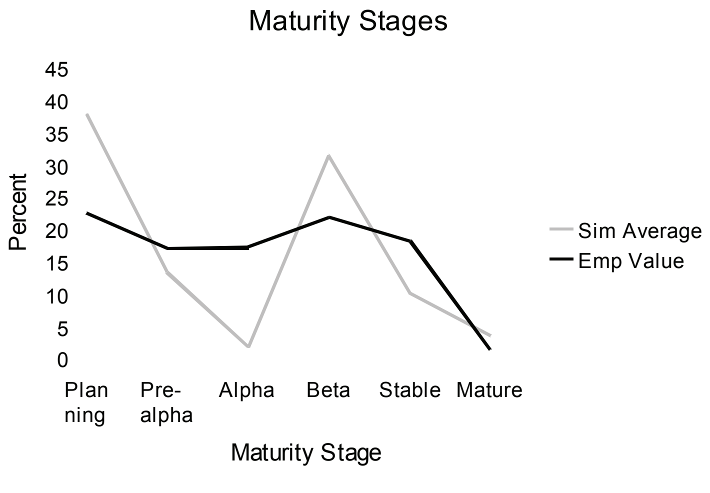
The averaged data (over 4 runs) from the simulator's best parameter set, along with the empirical data, is shown in Figs. 1, 2, and 3.

Figure 1 shows the generated percentage of projects in each maturity stage is a similar shape to the empirical data, with the main difference being the highs are too high and the lows are too low in the simulated data. This disparity may be a result of initial model startup conditions. At time 0, the model starts with all projects in the planning stage. This is obviously different than SourceForge, where the projects were gradually added over time, not all at once in the beginning. While the model does add new projects each time step, with a growth rate based on the rate of increase of projects at SourceForge, it may take

Box 2.

$$fitness = 1 - \frac{\text{sum of square of errors}}{\text{maximum possible sum of square of errors}} \qquad (2)$$

Figure 1. Percentage of FLOSS projects in maturity stages. Empirical data from (Weiss, 2005)



more than 250 time steps for maturity stages to stabilize after the differing initial condition. At the end of the simulation run, just short of 60% of the projects were created sometime during the simulation while the remaining 40% were created at time 0.

As shown in Figure 2, the number of developers per projects follows a near-exponential distribution and the simulated data is similar, especially for projects with fewer than seven developers. Note that the data in Figure 2 uses a logarithmic scale to help with a visual comparison between the two data sets. Beyond seven 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 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.

Figure 3 shows the number of projects per developer is a relatively good match between the simulated and empirical data, with the main difference being the number of developers working on one project. It is likely that this could be corrected via additional experimentation with parameters.

Table 3 contains the average fitness scores for each of the emergent properties for the top performing parameter set. These values provide a quantitative mechanism for confirming the visual comparisons made above: the maturity stage fitness score is indeed lower than the other two properties. 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.

Finally, examining the evolved utility weights of the top 10 performing parameter sets provides insight into what factors are important in the model for reproducing the three properties examined. Table 4 contains the averages and standard deviations for each

Figure 2. Percentage of projects with N developers. Empirical data from (Weiss, 2005)

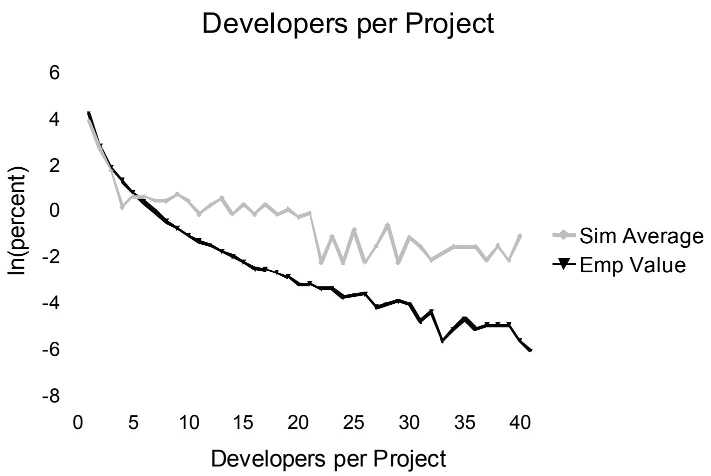


Figure 3. Percentage of developers with N projects. Empirical data from (Ghosh et al., 2002)

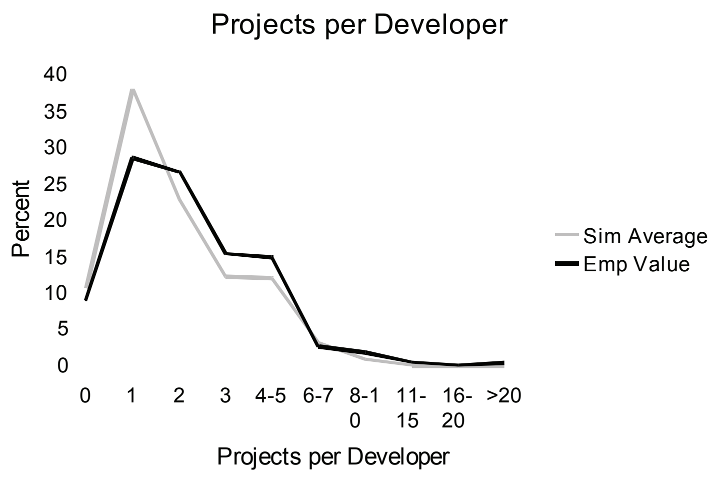


Table 3. Averaged fitness scores for the best

Emergent Property	Fitness Score
Maturity stage	0.9679
Devs per project	0.9837
Projects per dev	0.9938
Combined	0.9818

Table 4. Utility function weights from the best 10 param

Weight	Mean	Std. Dev.
w_1 (similarity)	0.1849	0.1137
w_2 (current resources)	0.3964	0.1058
w_3 (cumulative resources)	0.0003	0.0003
w_4 (downloads)	0.0022	0.0039
w_5 (maturity)	0.4163	0.1534

of the weights. It appears that the cumulative number of resources and download counts are not important in reproducing the examined properties in the model. This conclusion is reached by observing these weight's small values (low mean and small variance) in comparison to the other weights (high means and larger variance). Unfortunately, the high variance of the remaining three weights makes it difficult to rank them in order of importance. Rather, the conclusion is that similarity, current resources, and maturity are all important in the model.

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 5 contains the evolved mean and standard deviation for the producer and consumer numbers averaged from the top 10 parameter sets. Notice that the mean producer number is very high at 0.9801 and very stable across the top 10 parameter sets, with a standard deviation of 0.0079. Likewise, the standard deviation is relatively low at 0.1104 and also stable with a standard deviation of 0.0101. 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) produces better matching of the empirical data. This is in alignment with the notion that FLOSS is a developer-driven

Table 5. Evolved producer/consumer number distributions parameters

Producer/Consumer Number		Parameter statistics from top 10 parameter sets	
		Mean	Std. Dev.
Producer number	Mean	0.9801	0.0079
	Std. Dev.	0.1104	0.0101
Consumer number	Mean	0.6368	0.1979
	Std. Dev.	0.3475	0.3737

process. The evolved consumer number mean is much lower and 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 10 parameter sets. This indicates that the consumer number distribution has little effect on matching the empirical data for the top 10 parameter sets. Note that this is in alignment with the evolved weight for downloads approaching 0.0 in the utility functions. Consumers are not the driving force in matching the empirical data in the model.

DISCUSSION

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 et al. (2005) address this issue by using probabilities based off data from SourceForge to determine when developers continue with or leave a project they are currently involved with in their FLOSS model.

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.

Currently, needs vectors for projects and agents are generated independently. This has the problem of creating projects which have 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 driving force to reproduce the empirical data. 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) consider 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. The model is formulated from empirical studies and calibrated using SourceForge data. The calibrated version produces reasonable results for the three emergent properties examined. From the calibrated data, it is concluded that the similarity between a developer and a project, the current resources going towards a project, and the maturity stage of a project are important factors. However, the cumulative resources and number of downloads a project has received are not important in reproducing the emergent properties.

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). Research report. (Coverity Incorporated)

Antoniades, I., Samoladas, I., Stamelos, I., Angelis, L., & Bleris, G. L. (2005). Dynamical simulation models of the open source development process. In S. Koch (Ed.), *Free/open source software development* (pp. 174–202). Hershey, PA: Idea Group, Incorporated.

Axelrod, R. (1984). *The evolution of cooperation*. New York: Basic Books.

Bitzer, J., & Schröder, P. J. (2005, July). Bug-fixing and code-writing: The private provision of open source software. *Information Economics and Policy*, *17*(3), 389-406.

Brooks, F. P. (1975). *The mythical man-month: Essays on software engineering*. Reading, MA: Addison-Wesley.

Chelf, B. (2006). *Measuring software quality: a study of open source software.* Research report. (Coverity Incorporated)

Crowston, K., Howison, J., & Annabi, H. (2006, March/April). Information systems success in free and open source software development: Theory and measures. *Software Process: Improvement and Practice*, *11*(2), 123–148.

Crowston, K., & Scozzi, B. (2002). Open source software projects as virtual organizations: competency rallying for software development. In *IEE proceedings software*, 49, 3–17).

Dalle, J.-M., & David, P. A. (2004, November 1). *SimCode: Agent-based simulation modelling of open-source software development* (Industrial Organization). EconWPA.

English, R., & Schweik, C. M. (2007). Identifying success and tragedy of FLOSS commons: A preliminary classification of Sourceforge.net projects.

In *FLOSS '07: Proceedings of the first international workshop on emerging trends in FLOSS research and development* (p. 11). Washington, DC, USA: IEEE Computer Society.

Fox, J., & Guyer, M. (1977, June). Group size and others' strategy in an n-person game. *Journal of Conflict Resolution*, *21*(2), 323–338.

Gao, Y., Madey, G., & Freeh, V. (2005, April). Modeling and simulation of the open source software community. In Agent-Directed Simulation Conference (pp. 113–122). San Diego, CA.

Ghosh, R. A., Krieger, B., Glott, R., & Robles, G. (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.

Jerdee, T. H., & Rosen, B. (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, E., & Georgantzas, N. (2007). Why most open source development projects do not succeed? In *FLOSS '07: Proceedings of the first international workshop on emerging trends in FLOSS research and development* (p. 3). Washington, DC, USA: IEEE Computer Society.

Kicinger, R., Arciszewski, T., & De Jong, K. A. (2005). Evolutionary computation and structural design: A survey of the state of the art. *Computers and Structures*, *83*(23-24), 1943-1978.

Koch, S. (2008). Exploring the effects of SourceForge.net coordination and communication tools on the efficiency of open source projects using data envelopment analysis. In S. Morasca (Ed.), Empirical Software Engineering: Springer.

Kogut, B., & Metiu, A. (2001, Summer). Open-source software development and distributed innovation. *Oxford Review of Economic Policy*, *17*(2), 248-264.

Kowalczykiewicz, K. (2005). Libre projects lifetime profiles analysis. In *Free and open source software developers' European meeting 2005*. Brussels, Belgium.

Krishnamurthy, S. (2002, June). Cave or community?: An empirical examination of 100 mature open source projects. *First Monday*, *7*(6).

Lerner, J., & Tirole, J. (2005, April). The scope of open source licensing. *Journal of Law, Economics, and Organization*, *21*(1), 20–56.

Linux kernel software quality and security better than most proprietary enterprise software, 4-year Coverity analysis finds. (2004). Press release. (Coverity Incorporated)

Michlmayr, M. (2005). Software process maturity and the success of free software projects. In K. Zielinski & T. Szmuc (Eds.), *Software engineering: Evolution and emerging technologies* (p. 3-14). Krakow, Poland: IOS Press.

Ostrom, E., Gardner, R., & Walker, J. (1994). *Rules, games and common pool resources*. Ann Arbor, MI: University of Michigan Press.

Raymond, E. S. (2000, September 11). *The cathedral and the bazaar* (Tech. Rep. No. 3.0). Thyrsus Enterprises.

Rossi, M. A. (2004, April). *Decoding the "Free/Open Source(F/OSS) Software puzzle" a survey of theoretical and empirical contributions* (Quaderni No. 424). Dipartimento di Economia Politica, Università degli Studi di Siena.

Smith, S. C., & Sidorova, A. (2003). Survival of open-source projects: A population ecology perspective. In *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, K. J., Ammeter, A. P., & Maruping, L. M. (2006, June). 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.

Tajfel, H. (1981). *Human groups and social categories: Studies in social psychology*. Cambridge, UK: Cambridge University Press.

Wagstrom, P., Herbsleb, J., & Carley, K. (2005). A social network approach to free/open source software simulation. In *First international conference on open source systems* (pp. 16–23).

Wang, Y. (2007). *Prediction of success in open source software development*. Master of science dissertation, University of California, Davis, Davis, CA.

Weiss, D. (2005). Quantitative analysis of open source projects on SourceForge. In M. Scotto & G. Succi (Eds.), *Proceedings of the first international conference on open source systems (OSS 2005)* (pp. 140–147). Genova, Italy.

Nicholas P. Radtke is a PhD candidate in computer science at Arizona State University. His research focuses on understanding and modeling free/libre open source software engineering processes.

Marco A. Janssen is assistant professor on formal modeling of social and social-ecological systems within the School of Human Evolution and Social Change at Arizona State University. He is also the associate director of the Center for the Study of Institutional Diversity. His formal training is within the area of operations research and applied mathematics. His current research focuses on the fit between behavioral, institutional and ecological processes. In his research he combines agent-based models with laboratory experiments and case study analysis. Janssen also performs research on diffusion processes of knowledge and information, with applications in marketing and digital media.

James S. Collofello is currently computer science and engineering professor and associate dean for the Engineering School at Arizona State University. He received his PhD in computer science from Northwestern University. His teaching and research interests lie in the software engineering area with an emphasis on software quality assurance, software project management and software process modeling and simulation.