

# FLOSSSim ODD

Nicholas P. Radtke  
Nicholas.Radtke@asu.edu

December 17, 2011

The model description follows the ODD (Overview, Design concepts, Details) protocol for describing individual- and agent-based models [1, 2].

## 1 Purpose

The purpose of FLOSSSim is to gain a better understanding of the Free/Libre Open Source Software (FLOSS) development process through simulation. In particular, the goals of this model are to:

- Demonstrate that it possible to develop an empirically-grounded agent-based model that can explain historical patterns present in the FLOSS ecosystem.
- Gain insight into components of the FLOSS development process, such as how developers select projects and why some projects are successful.
- Explore what it means for a project to be successful in the FLOSS domain.
- Demonstrate that with sufficient calibration, modeling can be used to accurately predict components of the FLOSS ecosystem.

## 2 Entities, State Variables, and Scales

The model universe consists of two types of entities: agents and FLOSS projects.

Table 1 contains the state variables of agents. Table 2 contains the state variables of projects.

Table 1: Agent state variables.

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.	

The model’s performance is evaluated after 250 time steps, with a time step  $t$  equal to one week, resulting in a simulated period of just under five years.

### 3 Process Overview and Scheduling

Time is modeled as discrete steps with a weekly resolution. Projects are updated synchronously (i.e., new project values are temporarily stored until all agents have completed their tasks and then all projects are updated at once).

Pseudocode outlining the model’s schedule is provided in Fig. 1.

### 4 Design Concepts

Emergence: The emergence of several aggregate-level patterns are used for validation purposes by comparing the model’s output to empirical data. The patterns considered are: distribution of projects in development stages [3], number of developers per FLOSS project [3], and number of FLOSS projects per developer [4]. In addition, the model is able to reproduce a projects’ downloads distribution that is similar to empirical data.

Table 2: Project state variables.

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.	$\mathbb{R} [0.0, \textit{maxResources}]$
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 $\mathbb{R} [0.0, 1.0]$

Projects accumulate resources from agents' contributions. Successful projects emerge based on the amount of contributions received, frequency of contributions, etc. Different definitions of success are explored.

Adaptation: Agents use probabilistic choice when selecting projects to contribute to or to download, based on the calculated utility of each project contained in an agent's memory. As memory changes over time, agents gain and lose access to projects, which subsequently affects an agent's choice of projects. Selection is modeled through a multinomial logit equation resulting in imperfect choice, meaning agents do not always select the project with the highest perceived utility.

```

create projects
create agents
week t = 0
while (t < 250)
  for i from 1 to number of agents
    update agent[i] memory
    if agent[i] is consuming
      pick projects to consume
      download selected projects
    if agent[i] is producing
      pick projects to produce
      contribute resources to selected projects
  for j from 1 to number of projects
    update project[j] state variables
  add new projects
  t = t + 1

```

Figure 1: FLOSSSim schedule.

**Objectives:** Agents calculate a perceived utility of each project based on properties of the project and properties of the agent. When contributing to or downloading a project, agents select projects with high utilities with a higher probability.

**Learning:** The utility of a project to an agent increases if the agent worked on the project in the previous time step, simulating the ease of continuing to work on a familiar project versus the cost of learning to work on a new unfamiliar project.

**Sensing:** Agents know the maturity, current resources, cumulative resources, download count, and needs vector of all projects currently in the agent’s memory.

**Interaction:** Agents interact with projects by either downloading or contributing resources to a project. Agents are only able to interact with projects contained in the agent’s memory.

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. Agents are also able to discover new projects, adding them to their memory, based on this assumption.

Projects are assumed to be completely independent of one another; there are no explicitly modeled links between projects.

Stochasticity: Agents’ state variables and projects’ resources for completion, cumulative resources, and needs vectors are initialized via pseudo-random number generators.

Agents discovering and forgetting projects is a stochastic process. Likewise, the process of agents choosing projects includes probabilistic selection.

Observation: Data from projects is collected and aggregated following the termination of a simulation run. Data collected includes project maturity, number of developers per project, number of projects per developer, number of downloads, number of agents with interests similar to the project, and project state variables. Output data are dumped to a file for later analysis.

## 5 Initialization

At time  $t = 0$ , the model is seeded with 389 agents and 1024 projects.

The initialization values included in this section, unless explicitly stated otherwise, were determined by using genetic algorithms to find well-performing values.

Agents’ state variables are initialized as follows: The consumer number is generated from a normal distribution pseudo-random number generator with  $\mu = 0.78$  and  $\sigma = 0.68$  truncated to be in the range  $[0.0, 1.0]$ . The producer number is generated from a normal distribution with  $\mu = 0.91$  and  $\sigma = 0.04$  truncated to be in the range  $[0.0, 1.0]$ . The needs vector length is three and each element is generated from a uniform distribution in the range  $[0.0, 1.0]$ . Resources numbers are normalized to a 40 hour work week and are generated and assigned to agents based on the distribution found in a survey[4] that inquired how long developers spend working on FLOSS projects per week, as shown in Fig. 2. Within each discrete category (e.g., <2 hours, 2–5 hours) resources are assigned uniformly. An agent’s memory is seeded with five projects, where each project in the entire population of projects has an equal probability of being one of the five selected.

Projects’ state variables are initialized as follows: 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

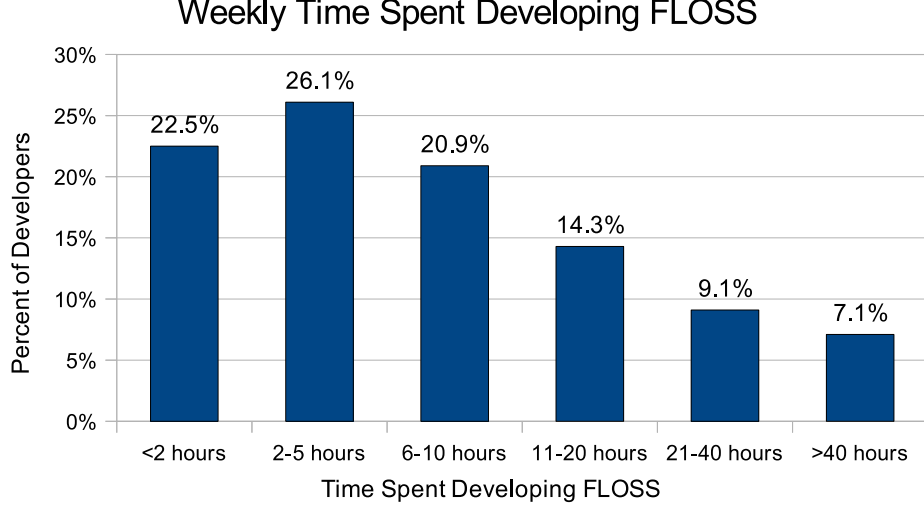


Figure 2: Resources number distribution based on weekly time spent developing FLOSS. Agents’ resources numbers are generated based on the distribution found in [4].

few large projects. Specifically, a project’s resources for completion is generated as shown in (1):

$$resourcesForCompletion = maxResources \cdot P_{expPRNG} \quad (1)$$

where

$maxResources$  defines the resources required to complete the largest possible project in the model (i.e. defines the upperbound size of projects)

$P_{expPRNG}$  is a truncated pseudo-random number generator based on the negative exponential distribution  $\lambda e^{-\lambda x}$  where  $\lambda = 5$  and bounded by the range  $[0.0, 1.0]$

$maxResources$  is set to 10,000, meaning that the largest possible project that could be created in a simulation run would take 10,000 40 hour work weeks to complete, the equivalent of 48 people working full time on a project for 4 years.

Not all FLOSS projects start from scratch as open source. Many projects are first partially developed and then released under an open source license. The development stage of projects when first added to SourceForge, as mined from the FLOSSmole database, is shown in Table 3. When creating projects,

Table 3: Development stage of projects when first added to SourceForge, based on data mined from the FLOSSmole database (see [5]).

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%

the cumulative resources and maturity are generated according to this distribution. The cumulative resources is set to the bottom threshold of each development stage, i.e., if a project is created in the alpha stage, then the cumulative resources is set to 25% of the resources for completion because 11% and 14% of the resources are contributed in the planning and pre-alpha stages respectively (see Fig. 3).

The project needs vector length is three and each element is generated from a uniform pseudo-random number generator in the range  $[0.0, 1.0]$ . A project’s current resources and downloads state variables are always initialized to zero.

## 6 Input

The model does not use input data to represent time-varying processes.

## 7 Submodels

### 7.1 Updating Agents’ Memory

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 probability of 0.065 an agent will be informed of a project and add it to its memory, simulating discovering new projects. Likewise, with a probability of 0.065 an agent will remove a project from its memory, simulating forgetting about or losing interest in old projects. These

probabilities were determined using evolutionary computation to find well-performing values. Thus, over time an agent’s memory may expand and contract.

## 7.2 Agents Selecting Projects

At each time step, agents choose to produce (i.e. develop) or consume (i.e. download) projects 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 (see Section 5).

All agents have memory which contains a subset of all available projects, and if producing or consuming, an agent calculates a utility score for each project in its memory. The utility function is shown in (2):

$$\begin{aligned}
 U = & w_1 \cdot \text{similarity}(\text{agentNeeds}, \text{projectNeeds}) \\
 & + w_2 \cdot \text{currentResources}_{\text{norm}} \\
 & + w_3 \cdot \text{cumulativeResources}_{\text{norm}} \\
 & + w_4 \cdot \text{downloads}_{\text{norm}} \\
 & + w_5 \cdot f(\text{maturity})
 \end{aligned} \tag{2}$$

Each term in the utility function represents a 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  $\sum_{i=1}^5 w_i = 1.0$ . Factors in the utility function were selected based on both FLOSS literature and a personal 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 with developers 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,  $\text{downloads}_{\text{norm}_i}$  is the  $i$ th project’s download count divided by the maximum



number of downloads that any project has received, as shown in (3):

$$downloads_{\text{norm}_i} = \frac{downloads_i}{\max(downloads_1, downloads_2, \dots, downloads_{\text{numOfProjs}})} \quad (3)$$

where

$downloads_i$  represents the number of downloads of the  $i$ th project

The discreet function  $f$  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. The number of new developers that join a project during each stage is used as a proxy to estimate the importance of each stage in attracting new developers. Using empirical data collected from the FLOSSmole database [5] and through the use of CVSSAnalY2 [6], the number of new developers in each development stage was counted for 76 projects that had progressed through four or more development stages; the discovered normalized average importance of each stage is shown in (4).

$$f(x) = \begin{cases} 0.62 & \text{if } x = \text{planning,} \\ 0.47 & \text{if } x = \text{pre-alpha,} \\ 0.27 & \text{if } x = \text{alpha,} \\ 0.23 & \text{if } x = \text{beta,} \\ 0.23 & \text{if } x = \text{production/stable,} \\ 0.0 & \text{if } x = \text{mature} \end{cases}$$

where  $x \in \{\text{planning, pre-alpha, alpha, beta, production/stable, mature}\}$  (4)

Evolutionary computation is used to determine combinations of the utility weights  $w_1$ – $w_5$  that perform well, which in turn provides information about the importance of each of the factors. Multiple clusters of weights that allow the model to match empirical data are found.

Since all terms in the utility function are normalized, the utility score is always a value between 0.0 and 1.0. In addition, the square root of the utility is used instead of  $U$  when calculating the utility for projects an agent developed for or downloaded in the previous time step, representing the added utility of continuing to work on the same project due to increased familiarity with the project.

Both consumers and producers use the same utility function. This is logical, as most FLOSS developers are also users of FLOSS [7, 8].

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

$$P_i = \frac{e^{\mu * U_i}}{\sum_{k=1}^{\text{numOfProjs}} e^{\mu * U_k}} \quad (5)$$

where

$P_i$  is the probability of selecting the  $i$ th project

$U_i$  is the utility of the  $i$ th project

$\{\mu \in \mathbb{R} | \mu \geq 0\}$  adjusts the level of perfect choice

The multinomial logit allows for imperfect choice, i.e., not always selecting the projects with the highest utility, and may be adjusted by changing the parameter  $\mu$ . When  $\mu$  is 0, all projects are chosen with equal probability regardless of each project's utility. The larger the value of  $\mu$ , the greater the chance an agent will select the “best” project, that is the project with the highest utility. In the model  $\mu$  is assigned the relatively high value of 36 (determined via the use of genetic algorithms to find a well-performing value), a value that causes agents to make well-informed decisions.

Agents are limited to producing or consuming up to a maximum number of projects at each time step. When an agent is producing, the number of projects the agent is engaged in is generated from an exponential distribution with an upper cutoff, as shown in (6):

$$\text{numProducing} = \text{maxNumProducing} \cdot P_{\text{expPRNG}} \quad (6)$$

where

$\text{maxNumProducing}$  defines the upper limit for the number of projects an agent can develop for in a single time step

$P_{\text{expPRNG}}$  is a truncated pseudo-random number generator based on the negative exponential distribution  $\lambda e^{-\lambda x}$  where  $\lambda = 5$  and bounded by the range  $[0.0, 1.0]$

By using an exponential distribution, agents will frequently be involved with a small number of projects and only occasionally develop many projects. Developer surveys have confirmed that the majority of developers work on only one or a few projects [9, 4] and the minority engage in many projects simultaneously [4, 10].  $\text{maxNumProducing}$  is assigned a value of 12 based on the results of using evolutionary computation to find a well-performing value.

A truncated normal distribution is used when agents consume, as it is assumed that most FLOSS users use a similar number of projects and are not subject to the larger time dedication required when developers become involved in a project. Each time an agent chooses to consume, the number of projects downloaded is determined by equation (7):

$$numConsuming = maxNumConsuming \cdot P_{normPRNG} \quad (7)$$

where

$maxNumConsuming$  defines the upper limit for the number of projects an agent can download in a single time step

$P_{normPRNG}$  is a truncated pseudo-random number generator based on a normal distribution with  $\mu = 0.5$  and  $\sigma = 1/6$  and bounded by the range  $[0.0, 1.0]$

$maxNumConsuming$  is assigned a value of 3 based on the results of using evolutionary computation to find a well-performing value.

When producing, agents contribute their entire resources number to the project(s) they selected. If contributing to multiple projects during a single time step, an agent's resources are distributed directly proportionally to the utility score calculated for each project. When consuming, the downloads count of each selected project is incremented.

### 7.3 Updating Projects

Project state variables are updated asynchronously. That is, agents evaluate projects based on their state variable values at time  $t - 1$ . Contributions made to projects at time  $t$  are stored in temporary variables and then copied to the project state variables after all agents have completed evaluating, contributing to, and downloading projects.

Projects update their needs vector at each iteration using a decaying equation, as shown in (8).

$$pneeds_{i,t} = \epsilon * pneeds_{i,t-1} \quad (8a)$$

$$+ \frac{1 - \epsilon}{resources_{i,t}} * \sum_{l=1}^{numOfAgents} (c_{l,i,t} * aneeds_l) \quad (8b)$$

where

$pneeds_{i,t}$  is the  $i$ th project's needs vector at time step  $t$

$pneeds_{i,t-1}$  is the  $i$ th project's needs vector at time step  $(t - 1)$

$$\begin{aligned}
&\{\epsilon \in \mathbb{R} | 0.0 \leq \epsilon \leq 1.0\} \\
&c_{l,i,t} \text{ is the } l\text{th agent's contribution to the } i\text{th project at time step } t \\
&{aneds}_l \text{ is the } l\text{th agent's needs vector} \\
&resources_{i,t} = \sum_{l=1}^{\text{numOfAgents}} c_{l,i,t}
\end{aligned}$$

The  $i$ th project's vector at time  $t$ ,  $pneeds_{i,t}$ , is partially based on the project's needs vector at time  $t - 1$  (8a) and partially on the needs vectors of the agents currently contributing to the project (8b). The rate of decay is controlled by  $\epsilon$ . An agent's influence on the project's needs 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. The use of a decaying equation allows projects to change their direction based on the agents contributing to them while at the same time maintaining inertia, based on work already completed on the project. The speed at which a project is able to change is adjusted by changing  $\epsilon$ . A value of 0.5 is chosen, which allows projects to be agile and rapidly adapt as the developer population changes while still providing a level of inertia to the project that comes from the work performed by earlier developers.

Projects are assigned to maturity stages based on the percent of the project that is complete. Setting the thresholds for the development stages is based on empirical data; namely, the percentage of commits that occur in a development stage is used as a proxy for the amount of work (i.e. resources contributed) that occurred in that stage. The dates projects changed stages was ascertained using the FLOSSmole database [5]; CVSAly2 [6] was then used to build database tables from each project's SCM logs, which were 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. 3.

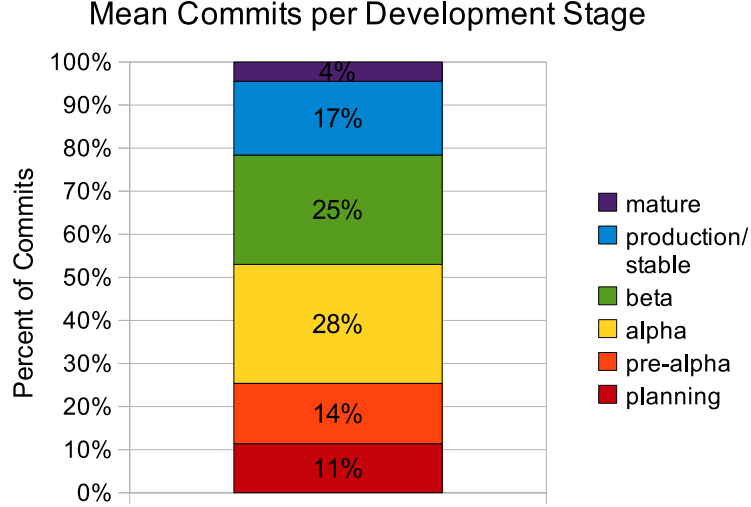


Figure 3: Mean percentage of code commits that occur in each development stage.

Projects are thus assigned to development stages based on (9).

$$d(x) = \begin{cases} \text{planning} & \text{if } 0.0 \leq x < 0.11, \\ \text{pre-alpha} & \text{if } 0.11 \leq x < 0.25, \\ \text{alpha} & \text{if } 0.25 \leq x < 0.53, \\ \text{beta} & \text{if } 0.53 \leq x < 0.78, \\ \text{production/stable} & \text{if } 0.78 \leq x < 0.96, \\ \text{mature} & \text{if } 0.96 \leq x \leq 1.0 \end{cases}$$

where  $x$  is the fraction of resources completed on a project, i.e.,  $\frac{\text{cumulativeResources}}{\text{resourcesForCompletion}}$  (9)

For example, a project in the model will be considered in the alpha stage if between 25% and 53% of the work has been completed.

A project is considered complete when its cumulative resources is equal to its resources for completion. Completed projects are not removed from the model. While they are not eligible for further contributions by developing agents, these projects may still be downloaded by users.

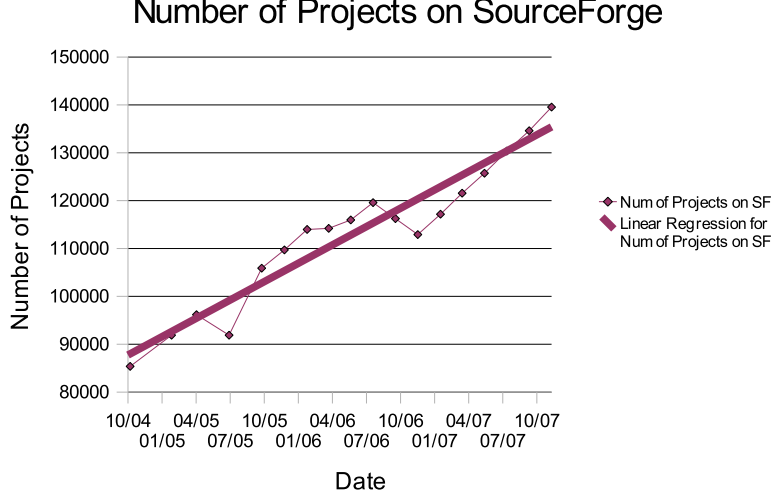


Figure 4: The number of projects on SourceForge with respect to time.

#### 7.4 Adding New Projects

New projects are created and added to the model at each time step, representing the new projects that are constantly being added to the open source community. The rate of creation of projects is based on empirical data from SourceForge, mined from the FLOSSmole database [5] and shown in Fig. 4.

The equation for the best-fit line is shown in (10):

$$y = 1242x + 88098 \quad (10)$$

where

$x$  is the number of months offset from October 2004 (i.e., 10/04 is 0, 11/04 is 1)

For the regression line,  $R^2 = 0.92$ , indicating that the line is a good fit and supporting that projects are added to SourceForge at a roughly linear rate, namely around 1242 projects per month. The occasional decrease in the number of projects appears to occur when SourceForge performs some level of housekeeping and removes dead projects, although SourceForge’s policy about the removal of projects does not appear to be publicly documented (i.e. frequency, criteria for removing a project, etc.).

FLOSSSim is normally run with a reduced set of projects, in which case the rate at which projects are added to the simulation is appropriately

scaled.

## References

- [1] V. Grimm, U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. K. Hienz, G. Huse, A. Huth, J. U. Jepsen, C. Jørgensen, W. M. Mooij, B. Müller, G. Pe'er, C. Piau, S. F. Railsback, A. M. Robbins, M. M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Souissi, R. A. Stillman, R. Vabø, U. Visser, and D. L. DeAngelis, “A standard protocol for describing individual-based and agent-based models,” *Ecological Modelling*, vol. 198, pp. 115–126, 2006.
- [2] V. Grimm, U. Berger, D. L. DeAngelis, J. G. Polhill, J. Giske, and S. F. Railsback, “The ODD protocol: A review and first update,” *Ecological Modelling*, vol. 221, pp. 2760–2768, Sep. 2010.
- [3] D. Weiss, “Quantitative analysis of open source projects on SourceForge,” in *Proceedings of the First International Conference on Open Source Systems (OSS 2005)*, M. Scotto and G. Succi, Eds., Genova, Italy, 2005, pp. 140–147.
- [4] R. A. Ghosh, B. Krieger, R. Glott, and G. Robles, “Part 4: Survey of developers,” in *Free/Libre and Open Source Software: Survey and Study*. Maastricht, The Netherlands: University of Maastricht, The Netherlands, Jun. 2002.
- [5] J. Howison, M. Conklin, and K. Crowston, “FLOSSmole: A collaborative repository for FLOSS research data and analyses,” *International Journal of Information Technology and Web Engineering*, vol. 1, pp. 17–26, Jul. 2006.
- [6] C. Garcia-Campos, *The CVSanaly Manual*, 2.0.0 ed., User Manual, LibreSoft, Apr. 2009. [Online]. Available: <http://gsyc.es/~carlosgc/files/cvsanaly.pdf>
- [7] K. Crowston and B. Scozzi, “Open source software projects as virtual organizations: Competency rallying for software development,” in *IEEE Proceedings Software*, vol. 149, no. 1, 2002, pp. 3–17.
- [8] E. S. Raymond, “The cathedral and the bazaar,” Thyrsus Enterprises, Tech. Rep. 3.0, Sep. 11, 2000.

- [9] R. A. Ghosh and V. V. Prakash, “The Orbiten free software survey,” *First Monday*, vol. 5, no. 7, Jul. 3, 2000. [Online]. Available: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/769/678>
- [10] S. Krishnamurthy, “Cave or community?: An empirical examination of 100 mature open source projects,” *First Monday*, vol. 7, no. 6, Jun. 2002. [Online]. Available: [http://www.firstmonday.org/issues/issue7\\_6/krishnamurthy/index.html](http://www.firstmonday.org/issues/issue7_6/krishnamurthy/index.html)