Population Dynamics in Open Source Communities: An Ecological Approach Applied to Github

Pablo Loyola
Industrial Engineering Department
University of Chile
Santiago, Chile
ployola@ing.uchile.cl

In-Young Ko
Deparment of Computer Science
Korea Advanced Institute of Science and Technology
Daejeon, South Korea
iko@kaist.ac.kr

ABSTRACT

Open Source Software (OSS) has gained high amount of popularity during the last few years. It is becoming used by public and private institutions, even companies release portions of their code to obtain feedback from the community of voluntary developers. As OSS is based on the voluntary contributions of developers, the number of participants represents one of the key elements that impact the quality of the software. In order to understand how the the population of contributors evolve over time, we propose a methodology that adapts Lotka-Volterra-based biological models used for describing host-parasite interactions. Experiments based on data from the Github collaborative platform showed that the proposed approach performs effectively in terms of providing an estimation of the population of developers for each project over time.

Categories and Subject Descriptors

J.4 [**Social and Behavioral Sciences**]: Economics; D.2.9 [**Software Engineering**]: Management—*Programming teams*

Keywords

Biological Mutualism; Ecological Models; Open Source Software Development

1. INTRODUCTION

The way in which software is developed has been changed considerably during the last decades. Firstly, passing from an individualistic and isolated programming paradigm to a social-based task solving procedures [2].

Secondly, as development tasks became more dynamic, allowing a more heterogeneous group of developers to take part into the programming issues, distributed collaborative systems, such as Git and Mercurial, began to gain popularity among several projects, many of them from the Open Source environment. This new change allowed the contribution from many non-core members to the projects, which produced more diversity and triggered innovation [20, 5].

The last observable change has come from the use of the Web as an interface for collaborative work in conjunction with the distributed subversion systems. This has generated a transformation that has contributed to understand software development as a social activity, generating ecosystems in which several stakeholders conform communities around software repositories[22].

Open Source Software (OSS) has embraced this paradigm shift, incorporating tools that boosts both transparency and visibility, elements that impact directly on the quality of the software developed [8]. One of the resulting artifacts of this change is Github, which is a Web-based hosting platform that provides the Git revision control system along with a solid social layer, which allows common functionalities such as *following*, *subscribing for updates* and *direct messaging* [10].

In OSS projects, one of the key elements related to the quality of the product is the group of contributors. Firstly, the number of people involved is relevant, because it provides a critical mass of contributors which will take care of bug fixing process and also the addition of new features. Additionally, a considerable amount of people working on the project could be useful to detect faults, following the assumption of the Linus's law [18].

Therefore, without people to contribute, the project dies as it remains inactive and it is not attractive for use. In that sense, having the possibility to get a notion of the evolution of the contributors could be useful to estimate the *health* of a project, in relation to the the probability that the project remain active for a certain period of time. This is relevant due to the increasing importance of OSS in both public and private sectors and thus it could be useful to provide a measure of the expected quality based on human factors rather than code metrics.

The new context of full access that the new social programming paradigm provides, generates more complexity and uncertainty to the dynamics of the projects, making more difficult to estimate the population of developers. Certain economic approaches based on the identification of incentives had focused on understanding the causes for the spontaneous collaborative communities [14]. Similarly, survey-based approaches, which have been useful for extracting best practices on the development process, only provides qualitative results.

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2578843.

In this work we provide a model to understand the population of contributors over time in OSS projects. Our approach is based on the use of biological mutualistic models in order to model the dynamics between contributors and repositories in an OSS ecosystem. The key element is the adaptation of the modified version of the Lotka-Volterra equations which are commonly used for the simulation of parasite-host interactions in biological fields. In this case, we propose an analogy based on the consideration of contributors as parasites and and repositories as hosts, both interacting under a mutualistic settings.

Our model not only considers the intrinsic grow rates of each actor, but also the inter-relationship between the participants and the effect of one on another. Previously, we explored the existence of a steady state in the dynamics of population in communities [16]. This work is a continuation which main goal is to understand the evolution of the population over time. The results showed that for a set of samples extracted from the Github platform, the overall dynamics of the population of contributors obtained from the mutualistic model fits in a reasonable way with the observed data.

It is important to note that the proposed approach is not intended to be used as a prediction tool. The main goal is to provide an initial way of understanding the population dynamics over time by means of a reduced number of parameters.

2. BACKGROUND

Our approach is based on the adaptation of a Biological Mutualistic model to study the evolution of the population of contributors in Open Source projects.

Biological Mutualism is commonly defined as the relationship between species that results into reciprocal benefits. As stated by Ollerton [19], these benefits include trophic gain, physical protection and dispersal of gametes. These can be categorized as *resource benefits*, which are related to nutritional gain between species, or *service benefits*, which are related to transport and dispersal.

2.1 Lotka-Volterra equations for mutualism

Most approaches to model Biological Mutualism are based on the modification of the Lotka-Volterra system of differential equations [17]. Lotka-Volterra equations were not developed initially to model mutualistic interactions, but to model competition between biological species [1], as predator-prey dynamics. These equations describe how the birth rate of a species depends on each other over time.

The modified version of the Lokta-Volterra equations takes into account two main elements: the intraspecific competition and the per-capita effect between species. Following the approach by Bascompte et al. [1], they can be represented as follows, assuming a scenario where a set of n plants and m animal species interact mutualistically under a host-parasite relationship:

$$\frac{dP_i}{dt} = r_i P_i - S_i P_i^2 + \sum_{j=1}^{m} \alpha_{ij} P_i A_j \quad (1)$$

$$\frac{dA_j}{dt} = q_j A_j - T_j A_j^2 + \sum_{i=1}^{n} \beta_{ji} P_i A_j \quad (2)$$

where P_i represents the population of the plant i and A_j represents the population of animal j. The remaining parameters are presented in Table 1.

Table 1: Lokta-Volterra parameters

Parameter	Meaning
r_i	Growth rate of plant i
S_i	Intraspecific competition of plant i
q_j	Growth rate of animal j
T_j	Intraspecific competition of animal j
α_{ij}	Per-capita effect of animal j on plant i
β_{ji}	Per-capita effect of plant i on animal j

3. OVERVIEW OF THE APPROACH

In this section, we describe the main components of our approach. As it is based on the adaptation of the Lotka-Volterra equations, we first state the main characteristics of a biological mutualistic configuration present in nature and then compare them with the standard behavior observed in Open Source environments. In that sense, we provide literature-based evidence related to the similarities between the two fields. Subsequently, we show how each parameter of the original model for biological mutualism in nature can be interpreted under a Open Source environment.

3.1 Similarities between biological and Open Source environments

The key analogy used in this work is based on the idea that ***OSS developers behave as parasites and the repositories play the role of the hosts, based on a mutualistic configuration***.

In that sense, the developers contribute to the project repository, improving the code and generating a better quality software. In turn, the repository benefits the developers as they can access the code to their own purposes. Additionally, the repository serves as a place for interaction where developers can gain reputation and knowledge. The following list details the main characteristics of a host-parasite mutualistic configuration and links them with the observed behavior in OSS development:

Mutual benefit between species: Following the seminal work by von Hippel and von Krogh, to understand the behavior and incentives in an Open Source environment, it is necessary to combine both private and collective innovation models. In that sense, the developers are willing to spend their own time and resources contributing to a common project, hoping to receive benefits in terms of knowledge and expertise that they can use for their private work [12]. Given that, and using the analogy from the host-parasite mutualism, we state that the repository (host) benefits based on the flow of contributions from the set of developers (parasites) that improves the host along time. On the other hand, developers (parasites) benefit themselves by means of the knowledge obtained from the community and also due to they can reuse of the developed software for private purposes.

High degree of specialization: In host-parasite interactions, the parasite species is forced to develop sophisticated functionalities in order to adapt to the host [19]. Similarly, in an Open Source development environment contributors arrive with a heterogeneous level of expertise [24]. Initially, this leads to the generation of a higher number of bugs,

produced by the low understanding of the code and the lack of synchronization between the developers. Thus, developers need to learn from the code, sometimes concentrating on a small number of modules, where it has been shown that *ownership* plays an important role in the reduction of errors [3, 4].

Co-evolution between species: Following the work by Thompson [23], mutualism allows the process of co-evolution between species, due to the symbiotic relationship that boosts their behavior.

In the context of software development, we assume that the initial status of both the repository and the set of contributors changes according to the overall activity. Developers begin to contribute and interact, which leads to two kinds of evolution: 1) each contributor improves his knowledge, based on continuous feedback on his commits and 2) the structure of the development network takes form, allowing the generation of roles and spontaneous hierarchies. On the other hand, the repository receives the contributions which change its structure and behavior. The performance of the resulting software is tested by the community, which represents the main source of feedback for the group, generating a symbiotic cycle.

Constant interaction leads to stable relationship: In [19], Ollerton showed that as the mutualistic relationship increases its level of biological intimacy, the number of participants is reduced, reaching a stable point. Empirical studies show that at early stages, several Open Source projects perform in an erratic way and the heterogeneity of the changes is also considerable, specially when the project lacks a road map [9]. As time passes, some initial collaborators abandon the project, leaving it converging to a organizational equilibrium. This leads to a smaller but more cohesive team which has a more comprehensive knowledge about the program [6].

A parasite may evolve to become less harmful to its host: The analogy in this case comes directly from the learning process in which developers involve when contributing to a project. The rich interaction with a diversity of contributors and the specialization in determined parts of the code, lead a developer to gain adaptiveness which eventually will reduce the number of bugs [11]. Thus, the probability to *harm* the repository could decrease.

Transference of genetic material between species: Social development platforms like Github allow a developer to contribute to several projects simultaneously. This represents a rich source of knowledge from which best practices can be extracted. Once the developer learns and internalizes an optimal way to generate a module or functionality, this expertise can be reused in several other projects that share the same requirements [13]. Thus, we use the concept of transference of genetic material to make an analogy with the transference and reuse of innovation and best practices between software projects.

3.2 Adaptation of the mutualism model to an Open Source environment

Given the model based on the modification of the Lotka-Volterra equations showed in Subsection 2.1, we present the way in which we relate it to an Open Source environment, adapting each parameter described in Table 1:

Growth rate: Under biological settings, the growth rate is related to the reproductive capabilities of the species involved. For the Open Source development environment, we assume that the growth rates of the developers and project repositories can be extracted simply by using past data and calculating the number of given elements per time unit.

Intraspecific competition: The concept of intraspecific competition deals with the inherent mechanism that rules the distribution of the resources within a given species. From an Open Source development perspective, we identify the two competitions as follows:

(a) Developer intraspecific competition: Although the concept of *competition* inside an Open Source project does not sound logical, we can get an estimation using a reputation-based metric. In that sense, we assume that although developers behave in a cooperative way, they search for the recognition of their peers. Given that, the number of followers that each developer has can give us a quantitative value of the reputation. Thus, we state that the degree of competition is high if the number of followers per developer is homogeneous. Otherwise, we assume the degree of competition is low (i.e., dominant configuration).

(b) Repository intraspecific competition: Project repositories need developers to survive. In that sense, we model this intraspecific interaction as a *struggle* to catch developers. Thus, we assume that there is a high degree of competition when the number of contributors per project repository is more homogeneous. On the other hand, the competition coefficient will be lower if the number of contributors per project repository is more heterogeneous (i.e., following a Power law distribution).

Per-capita effect: The per-capita effect refers to the unitarian contribution of one species to the behavior of the other. In an Open Source environment, we identify and define this kind of effects as follows:

(a) Developer to repository: It is calculated as the number of contributions (as commits) produced by a developer. We assume that adding or changing the code is the way in which the developer can influence the repository (although in reality there are other ways such as bug reporting and community management).

(a) Repository to developer: As we cannot easily get data about the private use of the knowledge and resources obtained by participating in the project, we focus on the *social* dimension of benefit. In that sense, we quantify the number of *followers* that a given developer obtained by contributing a determined project. We assume that the number of followers is directly related with the *reputation* that each developer has.

4. EVALUATION

In this section we provide a detailed overview of the evaluation strategy for the proposed approach. In general terms, it is based on the comparison of distribution of the population of developers and repositories over time obtained from the model, against the data observed empirically.

4.1 Test Environments

Given the nature of the proposed model, we cannot constrain the data to individual projects or developers and treat them as isolated artifacts. Our approach required as input *environments*, which means several repositories and developers interacting in a networked way.

Github by itself is an OSS environment, but the amount of data generated everyday makes it difficult to perform a

complete analysis. As a way to simplify the process, we decided to extract samples of the Github environment and test our approach on them. We called these samples as *sub-environments* and the way of extracting them considers the identification of a seed repository and the extraction of its contributors, then, a recursive method extracts all the other repositories in which the identified contributors are working on. The details are summarized in the following steps:

Seed repository extraction: We chose an initial set of repositories based on well known popularity and forking activity. This ensures a good starting point with a considerable level of heterogeneity.

Involved contributors extraction: For each seed repository extracted, the second step is to collect all its contributors. A contributor is defined as a developer that has had at least one commit on a given time period.

Second layer repository extraction: For each contributor identified in the previous step, all the other repositories in which he/she has been working on have to be retrieved.

Second layer contributors extraction: For all the repositories identified in the previous step, all of their contributors have to be collected.

At the end of the sampling extraction process, a set of *sub-environments* is obtained, each of them with a subset of contributors.

4.2 Experiment Design Overview

4.2.1 Data extraction

Having obtained the set of *sub-environments* from the previous step, we need to use each of these as sources of empirical data for validating the current approach.

As this experiment is inherently a time-dependent validation, the elements that conform each sample may vary over time. The only fixed item is the seed repository, from which the sample is generated. Thus, for each time instance selected (i.e. days) all the data has to be collected.

The first step is to extract the data from the Github platform. To do that, a set of scripts that use the API the Github provides, were written.

The time period for the extraction of the data was set up from January to June 2013, comprising six months of activity. The extracted data consists on two main elements: commit activity (from which rates of contributors can be extracted) and social activity (from which inter-developer interaction can be extracted).

All the data grouping was set on a daily basis. That means that all the parameters shown in Subsection 3.2 were calculated filtering by day. The number of selected seed repositories was 26, based on the most forked projects on the platform.

4.2.2 Population calculation

After collecting all the data for each sub-environment and for each day during the time interval of study, the process of population obtaining has to be performed. This process is done in two independent ways:

(a) Based on the Lotka-Volterra mutualistic model: Given all the parameters calculated for each sub-environment, they are passed to the model. The analytical solution of the of the system of equations showed in 2.1 provides a quantitative value for the population of contributors for each day.

(b) Direct calculation: This process is based on directly calculating the frequency of emergence of contributors, based on the identification of accepted commits.

ID	Seed Repository	ID	Seed Repository
1	bootstrap	14	CodeIgniter
2	homebrew	15	linux
3	rails	16	phonegap-plugins
4	html5-boilerplate	17	Diaspora
5	hw3rottenpotatos	18	three.js
6	oh-my-zh	19	zf2
7	node	20	jquery-mobile
8	jquery	21	django
9	phonegap-start	22	jekyll
10	impress.js	23	less.js
11	backbone	24	devise
12	d3	25	textmate
13	jquery-ui	26	redis

Table 2: Sub-environments with seed repository

4.2.3 Statistical Comparison

Having both ways of calculating the population of contributors over time, the next step is to perform a comparison between them. An common way to provide a statistical comparison that serves as a way to validate the quality of an estimator is to use the Root Mean Squared Error (RMSE) [25], which is defined as $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2}$, where $\hat{Y}$ represents the the vector of n estimated values, and Y is the vector of the observed values. In order to visualize in a better way the results, we will use the normalized version of RMSE, $NRMSE = \frac{RMSE}{\Delta Y}$, where ΔY represents the range of observed values of the variable under study.

With the results of this comparison it is possible to decide if the population for a given sub-environment behaves under a mutualistic configuration, based on how low the NRMSE value is.

4.3 Threats to Validity

External Validity: Our study is limited to a sample of existing repositories on the Github collaborative platform. There are other platforms in which Open Source artifacts are currently stored, such as SourceForge or GoogleCode. Nevertheless, we believe the this platform is representative of the current status of Open Source [8].

Internal Validity: Our main concern is related to the data extraction tools developed for conducting the experiments. Although Github provides a rich API, the time limitation on the requests forced us to make some modifications, such as restarting the extraction process several times.

Construct Validity: In Subsection 3.2 we defined the way in which the parameters of the standard biological model should be adapted to a Open Source environment. It is possible that this adaptation can be achieved in a different way, obtaining different results. However, adding a social component to the metrics, in terms of relating it with the reputation, represent an intuitive attempt, considering that this is an initial study.

5. RESULTS AND ANALYSIS

For each sub-environment, we solve the differential equations system using the Scipy[1] mathematical library . The respective value of the parameters were calculated a-priori for each instance.

Figure 1 summarizes the results showing the values of the NRMSE for each sub-environment based on the the selected set of seed repositories. As it can be seen, the majority of the sub-environments present a low NRMSE value , as almost 90% of them present a value under or equal to 0.2. On the other hand, few sub-environments present very high values, which means that the mutualistic behavior cannot be inferred.

In that sense, it could be said that, on average, the mutualistic behavior is present on the overall subset of the Github sub-environment, which means that the level of fitness between the computed population value and the observed is considerable.

If we take a look at the set of low NRMSE sub-environments, a common characteristic between them is that the seed repository from which the sub-environment was generated, is one of the commonly called *successful* OSS projects, such as Rails and Bootstrap. On the other side, high values of NRMSE are associated with less known projects, such as Diaspora. This is an interesting fact that can be related to the rate of new contributions that the studied environments have: as a project is important within the community, people could tend to focus on trying to become a contributor to that project, in the sense that 1) she can find a broad number of problems to work on or 2) a big and successful project represents a better showcase for showing herself to the community (reputation-based decision).

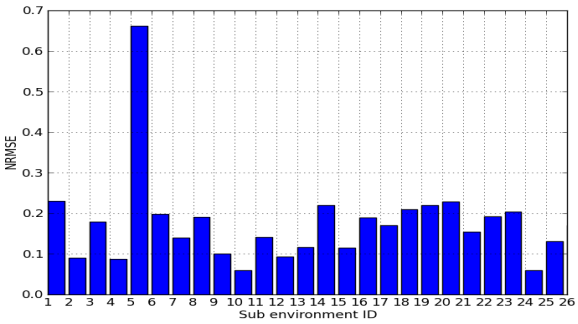

Figure 1: NRMSE values for each sub-environment

6. DISCUSSION

6.1 Impact of the programming language under use

We performed a variation in which the sub-environments are generated by selecting a seed repository that represents a specific programming language. From it, contributors and other repositories were extracted, under the condition that they share the same programming language.

Given the above, five languages were chosen, which led to five new sub-environments where the approach was conducted. Table 3 summarizes the results for this variation.

[1]http://www.scipy.org

ID	Seed Language	No of Projects	NRMSE
1	Javascript	67	0.11
2	PHP	51	0.31
3	Python	56	0.179
4	Ruby	33	0.101
5	Shell scripting	11	0.521

Table 3: Normalized Root Mean squared error for each language-based sub-environment

Interestingly, the condition for mutualism, a low value of NRMSE, is more accentuated. We theorize that this is because sharing the same programming language generates a higher fitness between the contributors and the repositories, which boots the level of specialization and subsequently reduces the number of failed contributions (bugs), producing a better result in terms of quality. Additionally, the fitness between developers should be stronger under this configuration, which eventually increases the value of the parameters related to the intraspecific competition (reputation).

In terms of the values obtained, Ruby presents the lowest one, closely followed by Javascript, which means the population values obtained from the model are closer to the real observed ones. On the other hand, the value for the Shell scripting sample shows a value in which it is difficult to state if the mutualistic behavior is present or not. We relate this result to the fact the both Ruby and Javascript are languages widely used and on the Github environment, on the contrary, Shell scripting is not popular. In that sense, the initial number of projects that share the same programming language could impact the performance of the proposed model, in the sense that having a critical mass with a considerable amount of heterogeneity improves the coexistence under mutualistic settings.

One interesting extension for this analysis could be the study other similarities between the repositories, in order to get an estimation on how the preferences of the developers impact the growth of the projects. Besides a language similarity, it could be relevant to inspect the domains in which the software has been used. Additionally, the geographical location of the contributors could represent another criteria, as it has been noted in other works [10].

7. RELATED WORK

From the economic perspective, several attempts had been proposed to understand the nature of the incentives and the reasons that drive people to contribute in Open Source projects. Lerner and Tirole noticed the relevance of the problem, stating that for the traditional economic theory, the behavior of Open Source contributors was *initially startling* [15]. They explored the phenomena from the point of view of labor economics, trying to identify the main motivations behind it, using a qualitative approach. In [14], the same authors deepen by providing more evidence and a hypothesis, specifically focused on trying to understand why private companies released the code from their commercial products.

Another relevant approach was introduced by von Hippel and von Krogh [12], where they proposed a combination between the *private investment* model of innovation and the *collective action* model of innovation to understand the underlying rationality of the phenomena. Similar approaches

focus on the discovery of the motivations of the developers [7, 21], using mainly survey-based analysis.

Regarding a more quantitative analysis, the majority of the research is focused on using Open Source development platforms to study the reliability of the software. In that sense, several approaches have been proposed focusing on the prediction of bug generation and its relation with the social structure of the community [26, 6] and also the level of communication [3].

8. CONCLUSIONS AND FUTURE WORK

The proposed approach brings the benefits of explicitly providing a notion of how the community behaves over time using a flexible approach based on biological mutualism. To the best of our knowledge, this is one of the first attempts to provide a quantitative notion of how the population of developers evolves over time. This represents a contribution to the general understanding of the dynamics of collaborative work on OSS projects. It is necessary to improve the data extraction processes to be able to perform additional analysis that provides more statistical confidence.

Regarding future work, we propose to improve the the approach to generate a prediction model from which reliability metrics for Open Source projects could be extracted. Additionally, it is necessary to perform more analysis on the interaction dynamics, such the per-capita effect and the *competition* between contributors.

9. ACKNOWLEDGMENTS

This work was partially supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract (UD060048AD).

10. REFERENCES

[1] J. Bascompte, P. Jordano, and J. M. Olesen. Asymmetric Coevolutionary Networks Facilitate Biodiversity Maintenance. *Science*, 312(5772):431–433, Apr. 2006.

[2] A. Begel, J. D. Herbsleb, and M.-A. Storey. The future of collaborative software development. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work Companion*, CSCW '12, pages 17–18, New York, NY, USA, 2012. ACM.

[3] M. Bernardi, G. Canfora, G. Di Lucca, M. Di Penta, and D. Distante. Do developers introduce bugs when they do not communicate? the case of eclipse and mozilla. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 139 –148, march 2012.

[4] C. Bird. Sociotechnical coordination and collaboration in open source software. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 568 –573, sept. 2011.

[5] C. Bird and N. Nagappan. Who? where? what? examining distributed development in two large open source projects. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 237 –246, june 2012.

[6] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent Social Structure in Open Source Projects. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 24–35. ACM, 2008.

[7] J. Bitzer, W. Schrettl, and P. J. SchrÃűder. Intrinsic motivation in open source software development. *Journal of Comparative Economics*, 35(1):160 – 169, 2007.

[8] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1277–1286, New York, NY, USA, 2012. ACM.

[9] P. A. David and J. S. Shapiro. Community-based production of open-source software: What do we know about the developers who participate? *Information Economics and Policy*, 20(4):364 – 398, 2008.

[10] B. Heller, E. Marschner, E. Rosenfeld, and J. Heer. Visualizing collaboration and influence in the open-source software community. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 223–226, New York, NY, USA, 2011. ACM.

[11] A. Hemetsberger and C. Reinhardt. Learning and knowledge-building in open-source communities. *Management Learning*, 37(2):187–214, 2006.

[12] E. v. Hippel and G. v. Krogh. Open source software and the "private-collective" innovation model: Issues for organization science. *Organization Science*, 14(2):209–223, Mar. 2003.

[13] B. Kogut and A. Metiu. Open source software development and distributed innovation. *Oxford Review of Economic Policy*, 17(2):248–264, 2001.

[14] J. Lerner and J. Tirole. The economics of technology sharing: Open source and beyond. Working Paper 10956, National Bureau of Economic Research, December 2004.

[15] J. Lerner and J. Triole. The simple economics of open source. Working Paper 7600, National Bureau of Economic Research, March 2000.

[16] P. Loyola and I.-Y. Ko. Biological mutualistic models applied to study open source software development. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conference on*, 2012.

[17] R. M. May. Mutualistic interactions among species. *Nature*, 296(5860):803–804, 1982.

[18] A. Meneely and L. Williams. Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 453–462, New York, NY, USA, 2009. ACM.

[19] J. Ollerton. *Biological Barter: patterns of specialization compared across different mutualisms.*, pages 411–435. University of Chicago Press, 2006.

[20] C. Rodriguez-Bustos and J. Aponte. How distributed version control systems impact open source software projects. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 36 –39, june 2012.

[21] S. K. Shah. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52(7):pp. 1000–1014, 2006.

[22] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 359–364, New York, NY, USA, 2010. ACM.

[23] J. N. Thompson. *The geographic mosaic of coevolution.* University Of Chicago Press, 1 edition, June 2005.

[24] G. von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217 – 1241, 2003.

[25] W. W.-S. Wei. *Time series analysis.* Addison-Wesley Redwood City, California, 1994.

[26] T. Zimmermann and C. Bird. Collaborative Software Development in Ten Years: Diversity, Tools, and Remix Culture. In *Proceedings of the Workshop on The Future of Collaborative Software Development*, 2012.