Longitudinal Analysis of the Run-up to a Decision to Break-up (Fork) in a Community

Amirhosein Azarbakht, Carlos Jensen

Longitudinal Analysis of the Run-up to a Decision to Break-Up (Fork) in a Community

Amirhosein "Emerson" Azarbakht, Carlos Jensen

Oregon State University, School of Electrical Engineering & Computer Science
1148 Kelley Engineering Center, Corvallis OR 97331, USA
{azarbaka,carlos.jensen}@oregonstate.edu

Abstract. In this paper, we use a developer-oriented statistical approach to understand what causes people in complex software development networks to decide to fork (break away), and what changes a community goes through in the run-up to a decision to break-up. Developing complex software systems is complex. Software developers interact. They may have the same or different goals, communication styles, or values. Interactions can be healthy or troubled. Troubled interactions cause troubled communities, that face failure. Some of these failures manifest themselves as a community split (known as forking). These failures affects many people; developers and users. Can we save troubled projects? We statistically model the longitudinal socio-grams of software developers and present early indicators and warning signs that can be used to predict an imminent break-up decision.

1 Introduction

Social networks are a ubiquitous part of our social lives, and the creation of online social communities has been a natural extension of this phenomena. Social media plays an important role in software engineering, as software developers use them to communicate, learn, collaborate and coordinate with others [32]. Free and Open Source Software (FOSS) development efforts are prime examples of how community can be leveraged in software development, where groups are formed around shared interest, and depend on continued interest and involvement to stay alive [25].

Community splits in free and open source software development are referred to as forks, and are relatively common [28]. Robles et al. [28] define forking as "when a part of a development community (or a third party not related to the project) starts a completely independent line of development based on the source code basis of the project."

Although the bulk of collaboration and communication in FOSS communities occurs online and is publicly accessible for researchers, there are still many open questions about the social dynamics in FOSS communities. Projects may go through a metamorphosis when faced with an influx of new developers or the involvement of an outside organization. Conflicts between developers' divergent

visions about the future of the project may lead to forking of the project and dilution of the community. Forking, either as an acrimonious split when there is a conflict, or as a friendly divide when new features are experimentally added, affect the community [8].

Previous research on forking ranges from the study by Robles et al. [28] that identified 220 significant FOSS projects that have forked over the past three decades, and compiled a comprehensive list of the dates and reasons for forking to the study by Baishakhi et al. [7] on post-forking porting of new features or bug fixes from peer projects. It encompasses works of Nyman on developers' opinions about forking [27], developers motivations for performing forks [24], the necessity of code forking as tool for sustainability [26], and Syeed's work on sociotechnical dependencies in the BSD projects family [33].

Most existing research on forking, however, is post-hoc. It looks at the forking events in retrospect and tries to find the outcome of the fork; what happened after the fork happened. The run-up to the forking events are seldom studied. This leaves several questions unanswered: Was it a long-term trend? Was the community polarized, before forking happened? Was there a shift of influence? Did the center of gravity of the community change? What was the tipping point? Was it predictable? Is it ever predictable? We are missing that context.

Additionally, studies of FOSS communities tend to suffer from an important limitation. They treat community as a static structure rather than a dynamic process. Longitudinal studies on open source forking are rare. To better understand and measure the evolution, social dynamics of forked FOSS projects, and integral components to understanding their evolution and direction, we need new and better tools. Before making such new tools, we need to gain a better understanding of the context. With this knowledge and these tools, we could help projects reflect on their actions, and help community leaders make informed decisions about possible changes or interventions. It will also help potential sponsors make informed decisions when investing in a project, and throughout their involvement to ensure a sustainable engagement.

We use an actor-oriented longitudinal statistical model [30] to study the evolution and social dynamics of FOSS communities, and to investigate the driving forces in formation and dissolution of communities. This paper is a part of a larger study aiming to identify better measures for influence, shifts of influence, measures associated with unhealthy group dynamics, for example a simmering conflict, in addition to early indicators of major events in the lifespan of a community. One set of dynamics we are especially interested in, are those that lead FOSS projects to fork.

2 Related Work

The free and open source software development communities have been studied extensively. Researchers have studied the social structure and dynamics of team communications [9][16][17][18][23], identifying knowledge brokers and associated

activities [31], project sustainability [23][26], forking [25][3][4][5], requirement satisfaction [13], their topology [9], their demographic diversity [20], gender differences in the process of joining them [19], and the role of age and the core team in their communities [1][2][6][12][35]. Most of these studies have tended to look at community as a static structure rather than a dynamic process [11]. This makes it hard to determine cause and effect, or the exact impact of social changes.

Post-forking porting of new features or bug fixes from peer projects happens among forked projects [7]. A case study of the BSD family (i.e., FreeBSD, OpenBSD, and NetBSD, which evolved from the same code base) found that 10-15% of lines in BSD release patches consist of ported edits, and on average 26-58% of active developers take part in porting per release. Additionally, They found that over 50% of ported changes propagate to other projects within three releases [7]. This shows the amount of redundant work developers need to do to synchronize and keep up with development in parallel projects.

Visual exploration of the collaboration networks in FOSS communities was the focus of a study that aimed to observe how key events in the mobile-device industry affected the WebKit collaboration network over its lifetime. [34] They found that *coopetition* (both competition and collaboration) exists in the open source community; moreover, they observed that the "firms that played a more central role in the WebKit project such as Google, Apple and Samsung were by 2013 the leaders of the mobile-devices industry. Whereas more peripheral firms such as RIM and Nokia lost market-share" [34].

The study of communities has grown in popularity in part thanks to advances in social network analysis. From the earliest works by Zachary [36] to the more recent works of Leskovec et al. [21][22], there is a growing body of quantitative research on online communities. The earliest works on communities was done with a focus on information diffusion in a community [36]. The study by Zachary investigated the fission of a community; the process of communities splitting into two or more parts. They found that fission could be predicted by applying the Ford-Fulkerson min-cut algorithm [14] on the group's communication graph; "the unequal flow of sentiments across the ties" and discriminatory sharing of information lead to subcommunities with more internal stability than the community as a whole.[36]

3 Research Goals

Social interactions reflect the changes the community goes through, and so, it can be used to describe the context surrounding a forking event. Social interactions in FOSS can happen, for example, in the form of mailing list email correspondence, bug report issue follow-ups, and source code contributions and co-authoring. We consider some forking decisions [28] to be socially related, such that, they should have left traces in the developers' interactions data. Such traces may be identified using longitudinal modeling of the interactions, without

digging into the contents of the communications. These three reasons are (1) Personal differences among developer team, (2) The need for more community-driven development, and (3) Technical differences for addition of functionality. In this study, we analyzed, quantified and visualized how a community is structured, how it evolves, and the degree to which community involvement changes over time. Our over-arching research objective was to identify these traces/social patterns associated with different types of undesirable forking

R.G. 1: Do forks leave traces in the collaboration artifacts of open source projects in the period leading up to the fork? To study the properties of possible social patterns, we need to verify their existence. More specifically, we need to check whether the possible social patterns are manifested in the collaboration artifacts of open source projects, e.g., mailing list data, issue tracking systems data, source code data. This is accomplished by statistical modeling of developer interactions as explained in more detail in section 4.

R.G. 2: What are the traces that can explain longitudinal changes in sociograms in run-up to a forking event? What quantitative measure(s) can be used as an early warning sign of an inflection point (fork)? Are there metrics that can be used to monitor the odds of change, (e.g. forking-related patterns) ahead of time? This will be accomplished by statistical modeling of developer interactions as explained in more detail in section 4.

4 Methodology

Detecting change patterns, requires gathering relevant data, cleaning it, and analyzing it. In the following subsections, we describe the proposed process in detail. Figure 1 shows the overview of our methodology.

4.1 Data Collection

The data collected were developer mailing lists, where developers' interact by sending and receiving emails, and source-code repository contribution logs, where developers interact by modifying the code. The sociograms were formed based on interactions among developers in these settings. For the purpose of our larger study, not included in this paper, we gathered data for 13 projects, in three categories of forking, plus a control group. We have included the data for a project that forked in 2010. The name is left out for anonymity, to prevent defaming a project, and to prevent individuals from becoming target of blame, in case our findings may be misused. Mailing list data was cleaned such that the sender and receiver email ID case-sensitivity differences would be taken into account, to prevent duplicity. The Source Code repository version control logs were used to capture the source code activity levels of the developers who had contributed more than a few commits. The set of the developers who had both mailing list activity and source code repository activity formed the basis of the

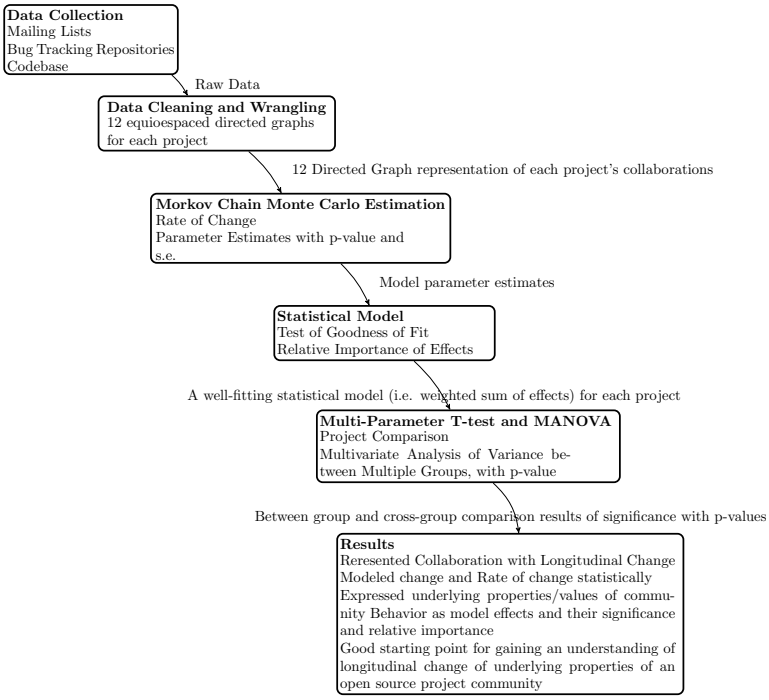

Fig. 1: The methodology in a glance

socio-grams we used in our analysis. The time period for which data was collected is one year leading to when the decision to break-up (fork) happened. This should capture the social context of the run-up to the forking event.

Social connections and non-connections can be represented as graphs, in which the nodes represent actors (developers) and the edges represent the interaction(s) between actors or lack thereof. Such graphs can be a snapshot of a network – a static sociogram – or a changing network, also called a dynamic sociogram. In this phase, we process interactions data to form a communication sociogram of the community. Two types of analysis can be done on sociograms: Either a *cross-sectional* study, in which only one snapshot of the network is looked at and analyzed; or a *longitudinal* study, in which several consecutive snapshots of the network are looked at and studied. We are interested in patterns in the run-up to forks, therefore, unlike most existing research on forking, we did a longitudinal study. We formed 10 equispaced consecutive time-window snapshots of the socio-grams for the community, using the mailing list interaction data and the source code repository commit activity data. These socio-grams were used to find a well-fitting statistical model that would explain how they changed from time-window t_1 through time-window t_{10}.

4.2 The Statistical Model

Longitudinal evolution of a network data is the result of many small atomic changes occurring between the consecutively observed networks. In our case, software developers are the actors in the networks, and they can form a connection with another developer, break off an existing connection, or maintain their status quo. These are the four possibilities of atomic change within our evolving networks: (1) forming a new tie; (2) breaking off an existing tie; (3) maintaining a non-connection; and (4) maintaining a connection. We assume a continuous-time network evolution, even though our observations are made at two or more discrete time points.

The state-of-the-art in studying longitudinal social networks, is the idea of *actor-oriented models* [30], based on a model of developers changing their outgoing ties as a consequence of a stochastic optimization of an *objective function*. This framework assumes that the observed networks at discrete times, are outcomes of a continuous-time Markov process. In the case of open source developers, the actor-oriented model, can be informally described as OpenSourceDeveloper-oriented model, in which, it is assumed that developers are in charge of their communication and collaboration choices. They choose to have interactions with certain other developers and/or they choose to stop having interactions with another developer. In short, they have autonomy in choosing their connections.

Let the data for our statistical developer-oriented model be M repeated observations on a network with g developers. The M observed networks (at least two) are represented as directed graphs with adjacency matrices $X(t_m) = (X_{ij}(t_m))$ for $m = 1, ..., M$, where i and j range from a to g. The variable X_{ij} shows whether at time t there exists a tie from i to j (value 1) or not (value 0). Be definition, $\forall i, X_{ii} = 0$ (i.e. the diagonal of the adjacency matrices).

In order to model the network evolution from $X(t_1)$ to $X(t_2)$, and so on, it is natural to treat the network dynamics as the result of a series of small atomic changes, and not bound to the observation moment, but rather as a more of less continuous process. In this way, the current network structure is a determinant of the likelihood of the changes that might happen next [10].

For each change, the model focuses on the developer whose tie is changing. We assume that developer i has control over the set of outgoing tie variables $(X_{i1}, ..., X_{ig})$ (i.e. the i^{th} row of the adjacency matrix). The network changes one tie at a time. We call such an atomic change a *ministep*. The moment at which developer i changes one of his ties, and the kind of change that he makes, can depend on attributes represented by observed covariates, and the network structure. The moment is stochastically determined by the *rate function*, and the particular change to make, is determined by the *objective function* and the *gratification function*. We cannot calculate this complex model exactly. Rather than calculating exactly, we estimate it using a Monte Carlo Markov Chain method. The estimated model is used to test hypotheses about the forked FOSS

communities. These above three functions and their definitions taken from [29] are explained in detail the following subsections.

4.2.1 Rate Function The *rate function* $\lambda_i(x)$ for developer i is the rate at which developer i's outgoing connections changes occur. It models how frequently the developers make ministeps. The rate function is formally defined [29] by

$$\lambda_i(x) = \lim_{dt \to 0} \frac{1}{dt} \mathrm{P}\big(X_{ij}(t+dt) \neq X_{ij}(t) \quad for \quad some \quad j \in \{i, ..., g\} | X(t) = x)\big). \tag{1}$$

The simplest specification of the rate of change is that all developers have the same rate of change of their ties.

4.2.2 Objective Function The *objective function* $f_i(s)$ for developer i is the value attached to the network configuration x. The idea is that, given the opportunity to make a change in his outgoing tie variables $(X_{i1}, ..., X_{ig})$, developer i selects the change that gives the greatest increase in the objective function. We assume that if there is difference between developers in their objective functions, these differences can be represented based on the model covariates [29]. For more details, please refer to [29]. The following weighted sum represents the objective function (2):

$$f_i(\beta, x) = \sum_{k=1}^{L} \beta_k s_{ik}(x) \tag{2}$$

Parameters $\beta = (\beta_1, ..., \beta_L)$ is to be estimated. Functions $s_{ik}(x)$ can be the following [29]:

4.2.2.1 Structural Effects For the structural effects, the following were used in the objective function.

1. The reciprocity effect, which reflects the tendency toward reciprocation of connections. A high value for its model parameter will indicate a high tendency of developers for reciprocated interactions.
2. The closure effects (e.g. in friendship networks, it means, friends of friends tend to become friends) In our case, Transitive triplets effect, which models the tendency toward network closure. It reflects the preference of developers to be connected to developers with similar outgoing ties.
3. Three-cycles, may be interpreted as the tendency toward local hierarchy. It is similar to reciprocity defined for three developers, and is the opposite of hierarchy.
4. Activity, which reflects the tendency of developers with high in-degree/out-degrees to send out more outgoing connections because of their current high in-degree/out-degree.

5. Covariate effects: Developers' covariates may influence the formation or termination of ties. For example: (a) Covariate V-related activity, which reflects the developer i's out-degree multiplied by his covariate V value. (b) Covariate V-related dissimilarity, which reflects the sum of differences in covariate V values' between developer i and all developers to whom developer i is connected. We use the following developer attributes as covariates:
   – (Covariate V1) Developer's level of activity (i.e. mailing list posts per month)
   – (Covariate V1) Developer's level of contribution (i.e. code commits per month) as shown in Table 2.
   – (Covariate V4) Developer's seniority as a development community member (i.e. how many total contributions they have had in the lifetime of the project)
6. out-out degree assortativity, which reflects which reflects the tendency of developers with high out-degree to be connected to other developers with high out-degrees

4.2.3 Markov Chain Transition Rate Matrix The components of the developers-oriented model, described above, define a continuous-time Markov chain on the space χ of all directed graphs on this set of g developers. This Markov chain is used to estimate the model parameters stochastically, instead of calculating them exactly, which is not possible for us. This Markov chain has a transition rate matrix. The transition rate matrix (also called intensity matrix), for this model is given by expression (3):

$$q_{ij}(x) = \lim_{dt \to 0} \frac{1}{dt} \mathrm{P}\big(X(t+dt) = X(i \mapsto j) | X(t) = x\big) = \lambda_i(x) p_{ij}(x) \quad (3)$$

Expression (3) shows the rate at which developer i makes ministeps, multiplied by the probability that he changes the arc variable X_{ij}, if he makes a ministep. Our Markov chain can be simulated by following the steps explained in [29].

4.2.4 Markov Chain Monte Carlo (MCMC) Estimation The described statistical model for longitudinal analysis of open source software development communities is a complex model and cannot be exactly calculated, but it can be stochastically estimated. We can simulate the longitudinal evolution, and estimate the model based on the simulations. Then we can choose an estimated model that has a good fit to the network data. For details of the simulation and estimation procedures please refer to [29]. The desirable outcome for the estimation is the vector parameter $\hat{\beta}$ for which the expected and the observed vectors are the same.

Table 1: Parameter estimates

Effect	par.	(s.e.)
Rate 1	1.419	(0.402)
Rate 2	2.633	(0.919)
Rate 3	3.231	(1.222)
Rate 4	11.656	(7.158)
Rate 5	5.238	(1.871)
Rate 6	5.431	(1.901)
Rate 7	1.863	(0.520)
Rate 8	0.791	(0.258)
Rate 9	0.671	(0.206)
outdegree (density)*	–5.389	(0.300)
reciprocity	–6.448	(31.754)
transitive triplets	–0.582	(0.875)
3-cycles	–2.680	(8.084)
out-out degree$\hat{}$(1/2) assortativity*	1.123	(0.291)
devScAct alter*	–0.021	(0.009)
devScAct ego*	0.011	(0.003)
devScAct ego x devScAct alter	–0.000	(0.000)
devMlAct alter	0.141	(0.010)
devMlAct ego	–0.037	(0.051)
devMlAct ego x devMlAct alter	0.002	(0.003)
int. devMlAct ego x devScAct ego*	0.003	(0.002)

5 Results

The results of parameter estimation are listed in Table 1. The parameter estimates that are statistically significant are marked with an asterisk (*) in Table 1. Recall that the weighted sum in expression (2) represents our objective function, and the effects listed in Table 1 are the parameter estimates of β_k's in expression (2).

The rate parameters represent the rate of change for the period between t_1 to t_2 for developers (i.e. how likely developers were to change ties in that time period). There's a clear trend in the rates 1-9, with a peak of 11.65 for the t_4 to t_5 time period. This suggests a significantly higher "preference" by developers for (a) forming new ties and interacting with previously non-connected developers and (b) terminating a previously connected tie. This peak value dies down as to less than 1, for the the t_8 to t_9 time period at 0.79 which can be used as an early warning sign of an imminent change decision.

6 Conclusion

In this study, we used a developer-oriented approach to statistically model the changes a FOSS community goes through in the run-up to a fork. The model represented tie formation, breakage, and maintenance between developers. We use 10 snapshots of the graph as observed data to estimate the influence of several effects on formation of the observed networks. We used a stochastic estimation method to estimate several model parameters of the model and used

a Wald-type t-test to estimate the significance of these parameters on this longitudinal change.

The results show that the out-out degree assortativity and the outdegree (density) effects are statistically significant, which can be interpreted that developers maintained a "preference" for interacting with developers who had similar outdegree levels. For example, core developers with high levels of mailing list activity responding to messages, were more likely to be connected to other similarly behaving high-outdegree developers. Also, that top answerer/repliers on the mailing list were more likely to contact other top developers, and the community shows a preference for inter-stratum ties.

The developers' source code repository contribution level (devScAct ego) was also statistically significant, which implies developers with higher levels of source code contributions increase their outdegree more rapidly. The developers' source code repository contribution level (devScAct alter) is also statistically significant, which implies developers with higher levels of source code contributions increase their indegree more rapidly.

Perhaps, an interesting observation is the existence of significance for high activity/contribution to the source code repository, however, in contrast, there's a lack of significance for high activity on the mailing list. In summary, high levels of contribution to the source code brings you connections more rapidly, while high levels of contributions to the mailing list is not suggestive of this. This can be interpreted as a sign of meritocracy based on code, rather than talk, which captures a healthy dynamic in this project, that was forked because of addition of functionality, and was classified as a healthy fork.

7 Threats to Validity

The study findings may not be generalized. First, one reason is that the projects is this research study were selected from a pool of candidate projects, based on a filtering criteria that included availability of their data. Given access, a larger number of projects as the sample size could result in a more robust investigation.

Second, we used data from online communications. The assumption that all the communication can be captured by mining repositories is intuitively imperfect, but inevitable. Third, social interactions data is noisy, and our statistical approach might be affected because of this.

Third, the statistical model we use to model the longitudinal evolution of collaboration networks is estimated stochastically, rather than being calculated exactly. The stochastic process might not always arrive at the same results. To counter this issue, we run the algorithm several times to double-check for such irregularities.

References

1. Azarbakht, A. and C. Jensen, "*Drawing the Big Picture: Temporal Visualization of Dynamic Collaboration Graphs of OSS Software Forks,*" Proc. 10th Int'l. Conf. Open Source Systems, 2014.
2. Azarbakht, A. and C. Jensen, "*Temporal Visualization of Dynamic Collaboration Graphs of OSS Software Forks,*" Proc. Int'l. Network for Social Network Analysis Sunbelt XXXIV Conf., 2014.
3. Azarbakht, A., "*Drawing the Big Picture: Analyzing FLOSS Collaboration with Temporal Social Network Analysis,*" Proc. 9th Int'l. Symp. Open Collaboration, ACM, 2013.
4. Azarbakht, A. and C. Jensen, "*Analyzing FOSS Collaboration & Social Dynamics with Temporal Social Networks,*" Proc. 9th Int'l. Conf. Open Source Systems Doct. Cons., 2013.
5. Azarbakht, A., "*Temporal Visualization of Collaborative Software Development in FOSS Forks,*" Proc. IEEE Symp. Visual Languages and Human-Centric Computing, 2014.
6. Azarbakht, E. A., "*Longitudinal Analysis of Collaboration Graphs of Forked Open Source Software Development Projects Using An Actor-oriented Social Network Analysis,*" Proc. Int'l. Network for Social Network Analysis Sunbelt conf., 2016.
7. Baishakhi R., C. Wiley, and M. Kim, "*REPERTOIRE: a cross-system porting analysis tool for forked software projects,*" Proc. ACM SIGSOFT 20th Int'l. Symp. Foundations of Software Engineering, ACM, 2012.
8. Bezrukova, K,, C. S. Spell, J. L. Perry, "*Violent Splits Or Healthy Divides? Coping With Injustice Through Faultlines,*" Personnel Psychology, Vol 63, Issue 3. 2010.
9. Bird, C., D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "*Latent social structure in open source projects,*" Proc. 16th ACM SIGSOFT Int'l. Symposium on Foundations of software engineering, ACM, 2008.
10. Coleman, J.S. "*Introduction to Mathematical Sociology,*" New York etc.: The Free Press of Glencoe. 1964.
11. Crowston, K., K. Wei, J. Howison, and A. Wiggins. "*Free/Libre open-source software development: What we know and what we do not know,*" ACM Computing Surveys, 44, 2, Article 7, 2012.
12. Davidson, J, R. Naik, A. Mannan, A. Azarbakht, C. Jensen, "*On older adults in free/open source software: reflections of contributors and community leaders,*" Proc. IEEE Symp. Visual Languages and Human-Centric Computing, 2014.
13. Ernst, N., S. Easterbrook, and J. Mylopoulos, "*Code forking in open-source software: a requirements perspective,*" arXiv preprint arXiv:1004.2889, 2010.
14. Ford, L. R. and D. R. Folkerson, "*A simple algorithm for finding maximal network flows and an application to the Hitchcock problem,*" Canadian Journal of Mathematics, vol. 9, pp. 210-218, 1957.
15. Forrest, D., C. Jensen, N. Mohan, and J. Davidson, "*Exploring the Role of Outside Organizations in Free/ Open Source Software Projects,*" Proc. 8th Int'l. Conf. Open Source Systems, 2012.
16. Guzzi, A., A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen. "*Communication in open source software development mailing lists,*" Proc. 10th Conf. on Mining Software Repositories, IEEE Press, 2013.
17. Howison, J., K. Inoue, and K. Crowston, "*Social dynamics of free and open source team communications,*" Proc. Int'l. Conf. Open Source Systems, 2006.

18. Howison, J., M. Conklin, and K. Crowston, "*FLOSSmole: A collaborative repository for FLOSS research data and analyses,*" Int'l. Journal of Information Technology and Web Engineering, 1(3), 17-26. 2006.
19. Kuechler, V., C. Gilbertson, and C. Jensen, "*Gender Differences in Early Free and Open Source Software Joining Process,*" Open Source Systems: Long-Term Sustainability, 2012.
20. Kunegis, J., S. Sizov, F. Schwagereit, and D. Fay, "*Diversity dynamics in online networks,*" Proc. 23rd ACM Conf. on Hypertext and Social Media, 2012.
21. Leskovec, J., Kleinberg, J., and Faloutsos, C.: "*Graphs over time: densification laws, shrinking diameters and possible explanations,*" Proc. SIGKDD Int'l. Conf. Knowledge Discovery and data Mining, 2005.
22. Leskovec, J., K. J. Lang, A. Dasgupta, and M. W. Mahoney, "*Statistical properties of community structure in large social and information networks,*" Proc. 17th Int'l. Conf. World Wide Web, ACM, 2008.
23. Nakakoji, K., Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. "*Evolution patterns of open-source software systems and communities,*" Proc. Int'l. Workshop Principles of Software Evolution, ACM, 2002.
24. Mikkonen, T., L. Nyman, "*To Fork or Not to Fork: Fork Motivations in SourceForge Projects,*" Int'l. J. Open Source Softw. Process. 3, 3. July, 2011.
25. Nyman, L. , "*Understanding code forking in open source software,*" Proc. 7th Int'l. Conf. Open Source Systems Doct. Cons., 2011.
26. Nyman, L., T. Mikkonen, J. Lindman, and M. Fougère, "*Forking: the invisible hand of sustainability in open source software,*" Proc. SOS 2011: Towards Sustainable Open Source, 2011.
27. Nyman, L., "*Hackers on Forking,*" Proc. Int'l. Symp. on Open Collaboration, 2014.
28. Robles, G. and J. M. Gonzalez-Barahona, "*A comprehensive study of software forks: Dates, reasons and outcomes,*" Proc. 8th Int'l. Conf. Open Source Systems, 2012.
29. Snijders, Tom AB. "*Models for longitudinal network data,*" Models and methods in social network analysis 1: 215-247. 2005.
30. Snijders, Tom AB., GG Van de Bunt, CEG Steglich, "*Introduction to stochastic actor-based models for network dynamics,*" Social networks 32 (1), 44-60. 2010.
31. Sowe, S., L. Stamelos, and L. Angelis, "*Identifying knowledge brokers that yield software engineering knowledge in OSS projects,*" Information and Software Technology, vol. 48, pp. 1025-1033, Nov 2006.
32. Storey, M., L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky, "*The (R) Evolution of social media in software engineering,*" Proc. Future of Software Engineering, ACM, 2014.
33. Syeed, M. M., "*Socio-Technical Dependencies in Forked OSS Projects: Evidence from the BSD Family,*" Journal of Software 9.11 (2014): 2895-2909. 2014.
34. Teixeira, J., and T. Lin, "*Collaboration in the open-source arena: the webkit case,*" Proc. 52nd ACM conf. Computers and people research (SIGSIM-CPR '14). ACM, 2014.
35. Torres, M. R. M., S. L. Toral, M. Perales, and F. Barrero, "*Analysis of the Core Team Role in Open Source Communities,*" Int. Conf. on Complex, Intelligent and Software Intensive Systems, IEEE, 2011.
36. Zachary, W., "*An information flow model for conflict and fission in small groups,*" Journal of Anthropological Research, vol. 33, no. 4, pp. 452-473, 1977.

Table 2: The list of developers source code contributions in the 10 months run-up to the forking event, sorted by total number of commits.

	Developer	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	Sum $t_1..t_{10}$
1	Anonymized Developer #1	17	54	48	22	86	298	238	154	136	210	1263
2	Anonymized Developer #2	55	100	42	58	74	156	120	16	44	4	669
3	Anonymized Developer #3	7	34	12	70	64	70	8	38	146	118	567
4	Anonymized Developer #4	21	163	54	138	64	46	38	36	0	4	564
5	Anonymized Developer #5	38	190	6	26	40	14	10	30	34	36	424
6	Anonymized Developer #6	21	0	20	58	59	35	48	41	24	80	386
7	Anonymized Developer #7	0	0	0	0	0	36	42	47	143	15	283
8	Anonymized Developer #8	23	22	9	87	72	1	1	0	0	0	215
9	Anonymized Developer #9	8	60	53	55	3	1	0	0	12	0	192
10	Anonymized Developer #10	0	0	3	81	39	12	4	8	2	4	153
11	Anonymized Developer #11	0	0	0	0	8	60	1	6	14	23	112
12	Anonymized Developer #12	2	47	30	1	7	2	0	8	0	0	97
13	Anonymized Developer #13	0	0	3	0	0	0	11	13	1	63	91
14	Anonymized Developer #14	0	0	0	0	0	0	0	8	38	40	86
15	Anonymized Developer #15	3	35	33	1	0	0	0	0	0	0	72
16	Anonymized Developer #16	3	0	0	0	0	0	0	4	17	46	70
17	Anonymized Developer #17	0	3	0	25	40	0	0	0	0	0	68
18	Anonymized Developer #18	0	0	0	55	0	9	0	1	0	1	66
19	Anonymized Developer #19	0	0	0	0	0	0	4	21	17	23	65
20	Anonymized Developer #20	0	0	0	9	15	14	11	2	6	0	57
21	Anonymized Developer #21	13	1	3	3	0	12	17	2	0	0	51
22	Anonymized Developer #22	8	18	12	0	0	0	0	0	3	4	45
23	Anonymized Developer #23	0	0	9	6	0	0	1	3	1	24	44
24	Anonymized Developer #24	0	0	0	0	0	0	13	16	3	4	36
25	Anonymized Developer #25	5	20	10	0	0	0	0	0	0	0	35
26	Anonymized Developer #26	1	0	11	2	14	7	0	0	0	0	35
27	Anonymized Developer #27	0	0	0	0	0	0	4	14	3	13	34
28	Anonymized Developer #28	3	12	4	1	5	0	0	1	1	4	31
29	Anonymized Developer #29	0	0	0	26	1	0	0	0	0	0	27
30	Anonymized Developer #30	0	0	0	0	0	0	0	0	0	26	26
31	Anonymized Developer #31	0	0	0	0	3	8	7	0	0	8	26
32	Anonymized Developer #32	0	0	0	0	10	13	0	1	0	0	24
33	Anonymized Developer #33	0	0	0	0	0	0	19	2	2	0	23
34	Anonymized Developer #34	0	0	0	16	7	0	0	0	0	0	23
35	Anonymized Developer #35	0	0	0	0	0	0	2	19	0	0	21
36	Anonymized Developer #36	0	8	11	0	0	0	0	0	0	0	19
37	Anonymized Developer #37	0	0	0	18	0	0	0	0	0	0	18
38	Anonymized Developer #38	0	0	0	0	0	0	17	0	0	0	17
39	Anonymized Developer #39	0	0	0	11	6	0	0	0	0	0	17
40	Anonymized Developer #40	0	0	0	0	0	0	0	2	0	12	14
41	Anonymized Developer #41	3	0	1	0	0	0	0	0	0	9	13
42	Anonymized Developer #42	0	0	0	0	0	0	7	2	2	2	13
43	Anonymized Developer #43	2	0	0	3	0	1	1	0	1	5	13
44	Anonymized Developer #44	0	0	0	0	0	0	0	0	8	5	13
45	Anonymized Developer #45	1	2	0	5	1	0	0	1	3	0	13
46	Anonymized Developer #46	0	0	0	0	4	5	3	0	1	0	13
47	Anonymized Developer #47	0	0	0	0	0	0	0	0	3	9	12
48	Anonymized Developer #48	0	0	0	0	0	0	10	1	0	0	11
49	Anonymized Developer #49	0	0	0	0	0	0	1	10	0	0	11
50	Anonymized Developer #50	0	6	5	0	0	0	0	0	0	0	11
51	Anonymized Developer #51	0	0	2	1	0	0	0	0	0	8	11
52	Anonymized Developer #52	0	1	0	0	0	0	0	6	2	0	9
53	Anonymized Developer #53	0	0	0	4	4	0	0	0	0	1	9
54	Anonymized Developer #54	0	0	0	0	0	0	0	0	0	9	9
55	Anonymized Developer #55	1	0	0	0	1	6	0	0	0	0	8
56	Anonymized Developer #56	1	6	1	0	0	0	0	0	0	0	8
57	Anonymized Developer #57	1	7	0	0	0	0	0	0	0	0	8
58	Anonymized Developer #58	0	0	0	0	0	0	0	0	0	8	8
59	Anonymized Developer #59	0	0	0	2	4	0	0	0	0	0	6