Noname manuscript No.
(will be inserted by the editor)

Stochastic Actor-Oriented Modeling for Studying Homophily and Social Influence in OSS Projects

David Kavaler · Vladimir Filkov

Received: date / Accepted: date

Abstract Open Source Software projects are communities in which people "learn the ropes" from each other. The social and technical activities of developers evolve together, and as they link to each other they get organized in a network of changing socio-technical connections. Traces of those activities, or behaviors, are typically visible to all, in project repositories and through communication between them. Thus, in principle it may be possible to study those traces to tell which of the observable socio-technical behaviors of developers in these projects are responsible for the forming of persistent links between them. It may also be possible to tell the extent to which links participate in the spread of potential behavioral influences.

Since OSS projects change in both social and technical activity over time, static approaches, that either ignore time or simplify it to a few slices, are frequently inadequate to study these networks. On the other hand, ad-hoc dynamic approaches are often only loosely supported by theory and can yield misleading findings. Here we adapt the *stochastic actor-oriented models* from social network analysis. These models enable the study of the interplay between behavior, influence and network architecture, for dynamic networks, in a statistically sound way.

We apply the stochastic actor-oriented models in case studies of two Apache Software Foundation projects, and study code ownership and developer productivity as behaviors. For those, we find evidence of significant social selection effects (homophily) in both projects, but in different directions. However, we find no evidence for the spread (social influence) of either code ownership or developer productivity behaviors through the networks.

D. Kavaler, V. Filkov
Computer Science Department
University of California, Davis
Davis, CA, 95616
E-mail: dmkavaler@ucdavis.edu, filkov@cs.ucdavis.edu

Data and scripts used in this work can be found at `http://web.cs.ucdavis.edu/~filkov/software/ASF-Siena/`.

1 Introduction

Open Source Software (OSS) projects are complex ecosystems of social interactions and technical activity [25,59]. They can be thought of as overlapping social networks of teams of contributors, and technical networks of artifact (code) dependencies. Figure 1 illustrates the temporal component and ecosystem in which the OSS social networks change. Complex networks are an appropriate formalism for modeling them as they can capture emergent properties relevant to software development, like accomplishing distributed tasks, team coordination, and social organization [12]. Information supplied by various people flows through these socio-technical networks. *E.g.*, new code is introduced, existing code is flagged for change, new features are discussed, and high priority bugs are assigned for debugging. To make sense of it all, a contributor might socially "link" to a small subset of people, *e.g.*, popular community members, as good central proxies to understand the dynamics of the system as a whole. One's social and technical activities in such a project change with the needs and expectations of tasks currently in focus, and evolve with the project. To increase their productivity, or quality, one may also borrow from others' either code [50], or desirable behavior. *E.g.*, to become a committer one has to emulate behavior, like patch submission and socialization amount [27], at levels practiced by the community, so as to establish trust.

Arguably, such linking is done to improve one's OSS experience. Explicitly or not, social linking among developers can affect the resulting software quality, which can gain by appropriate usage of social linking [8]. Software processes can also be improved by knowledge and experience reuse, or imitation [5]. And productivity, too, can be affected by judiciously establishing collaborations in OSS projects [84]. In fact, Software Engineering has benefited from social network analysis methods [75,8], especially in the field of distributed software development [46,18,35]. However, longitudinal studies of the dynamics of these networks have mostly been coarse-grained and not in keeping with modern statistical modeling formalisms. E.g., some studies gather data over a time period, only to reduce their analysis to a single cross-section [46, 9]. Others retain the longitudinal dimension of their data, but analyze at a relatively coarse-grained level [35]. Though the contributions of existing studies are palpable and cannot be ignored, complex and dynamic social networks such as those found in Open Source Software could greatly benefit from a more fine-grained analysis approach.

There is a dearth of studies, however, the above ones notwithstanding, that quantify the initiation of links, and the influence and spread of programmer behavior along the social network of developers. The culprit, in part, has likely been the absence of large-scale longitudinal data, which is necessary to provide sufficient variance among the behaviors for modeling. More critically, appro-

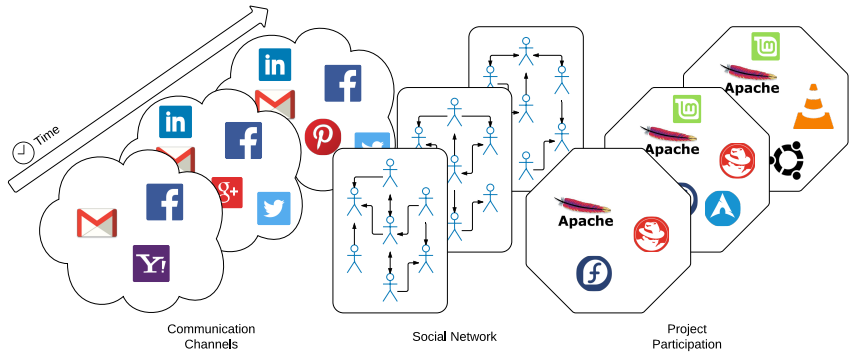


Fig. 1: Developers socialize through different channels over time, and correspondingly change their social ties. In addition, developers change their project interests and committing behavior over time. This forms a complex and dynamic longitudinal ecosystem of social interaction and productivity behavior. In this work, we consider emails as a communication channel between developers within individual projects.

priate statistical technology that can model longitudinal behavior over social networks has a high bar to digest and use properly. Without such methodology results could be misleading. So, *how* do we evaluate the analogue of the social phenomenon of making friends, *i.e.* linking to people we like, and the analogue of emulating behavior, *i.e.* mimicking others' attributes, in OSS socio-technical networks?

Borrowing a page from social network analysis, here we demonstrate how to adapt the statistical approach of *stochastic actor-oriented modeling* (SAOM) to study the effects of social links and neighbors' behavior on new network link formation and behavior influence in social networks of OSS developers. This allows us to conveniently test hypotheses within an environment where behaviors interact and feed into each other over time. To that end, we give a detailed description of the full pipeline for modeling using SAOMs, aimed at empirical software engineering practitioners. We illustrate this approach on case studies of two Apache Software Foundation (ASF) projects: Ant and Axis2/Java), on which we model three individual behaviors: one's commit count, high file ownership, and minor contributor count (or low file ownership), all over time.[1] We found that:

1. Modeling OSS temporal network data is intricate. Care must be taken to understand the networks and the amount of changes between time points, as well as the modeling parameters. We provide some guidance, based on our experience for best utilizing SAOMs in general.

[1] These behaviors have been studied before as important indicators of bugs, socio-technical structure, and overall software quality [22,26,82,11,55], and are discussed further in Sec. 5.

2. The evolution of OSS social networks can be modeled statistically once the above are understood, illustrating the potential of stochastic actor-oriented modeling to study complex, time-varying phenomena in software ecosystems.
3. There is either a significant or suggestive positive effect of behavior on network link formation for all behaviors tested in Axis2/Java, and a significant and negative effect for all behaviors tested in Ant. We find that there is no network structure influence effect on behavior evolution (social influence) for all behaviors tested in all projects studied.

The paper is structured as such: Section 2 covers relevant theory and related work; Section 3 briefly explains stochastic actor-oriented modeling; Section 4 covers general methods and advice for using SAOM; Section 5 presents details on applying SAOMs to our data. This is followed by a discussion in Section 6 and our conclusions and threats to validity in Section 7. Data and scripts used in this work can be found at `http://web.cs.ucdavis.edu/~filkov/software/ASF-Siena/`.

2 Theory and Related Work

Open Source Software projects exist and thrive in large part due to their dedicated community who feel connected and belonging to a team [21]. A number of studies have shown that human factors play a crucial role in the quality of software [11,10,16,51,54]. In addition, much work has been done in studying software systems through the use of various social, technical, and socio-technical networks [46,18,35,49,42,24]. Bird *et al.* examined socio-technical networks using classical social network analysis techniques to predict failure prone components in releases of Windows [10]. In addition, Bird *et al.* studied the email social networks of various OSS projects, finding that collaborative sub-communities within each project spontaneously arise as the projects evolve [12]. A question that has not been explored in a theoretically fundamental way, at least to our knowledge, is how do social interactions between developers affect the evolution of the software engineering process and vice-versa?

There has been work done in finding answers to this question, but in a less specialized fashion. Xuan *et al.* presented phenomenological quantitative methods to measure the effects of social activity on individual work behavior, finding that communication before and after committing activities plays a role in development of OSS projects [83]. Gharehyazie *et al.* examined email social networks related to various ASF projects and found that social activity alone can serve as a strong predictor of developer initiation [27]. Meneely *et al.* used social network measures to analyze the effect of focus and ownership on security errors, finding that lower focus is associated with more security errors [48]. Most closely related to our work is the work of Singh, who studied macro-level properties of developer collaboration networks, finding that they exhibit small-world properties as those found in "natural" social networks [64].

Though these works deal explicitly with social interaction between developers, they all utilize methodologies that differ from what we use here, and have different goals. We are focused on explicitly modeling the temporal interplay between social ties in the developer social networks, and their technical behavior while working on the artifacts.

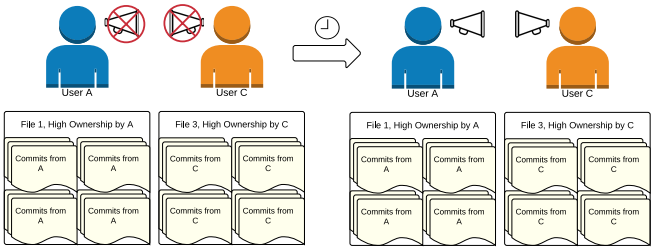
In the domain of health research, social networks have been used to study the spread of infectious diseases, innovations, knowledge, and other related phenomena [1,53,52,4,86,30]. Initially, very basic models were used to describe this spreading effect (also called *epidemiological processes*). Physicists became interested in these epidemiological processes when it was found that they can be mapped to the bond percolation problem [15]. Simple *compartmental models* were developed to describe these networks, with the main appeal of being described by tractable, closed-form mathematical formulae [13]. These simple models have since been extended far beyond their initial capacity [63]. However, it is not clear that the assumptions these simpler models rely on hold in real networks. Examples of these assumptions are homogeneity in degree distribution, fixed infectiveness time, uncorrelated probability of transmission between all pairs of individuals, and static network topology. These assumptions can be prohibitive or even misleading when studying real networks [52, 4,78]. An issue of primary importance in the general study of epidemiological and epidemiological-like processes is the separation of social influence and social selection (also called *contagion* and *homophily* in literature, respectively). This problem is more succinctly described as an issue of causal inference[2] or the *reflection problem* [47]; *e.g.* colloquially, do people become more like their friends (*i.e.* causally, through social influence), or do people make friends with those who are already like them (social selection)? These issues and other criticisms of existing methodologies have been discussed extensively within the research community [62,19,31,20].

Since OSS projects are complex, dynamic systems, the study of their details at fine granularity (*i.e.* at the developer level) calls for methods that can elucidate potential confounds, isolating important factors for software engineering in the presence of complex interactions (*e.g.* the potential confounding of degree with transitivity). The *stochastic actor-oriented models* of Snijders *et al.* [67,65] are one such family of modeling methods. They explicitly model longitudinal network and behavioral data simultaneously (*i.e.* evolving networks and evolving behavior) at the *per-node level*, avoiding many limiting assumptions of alternative models of network evolution [37,3,14][3] Though well-founded criticisms regarding some problems, like reflection and causal inference, still exist with stochastic actor-oriented models, many constraints due to the assumptions of previous (simpler) models are greatly reduced, increasing flexibility as is necessary to study such complex systems.

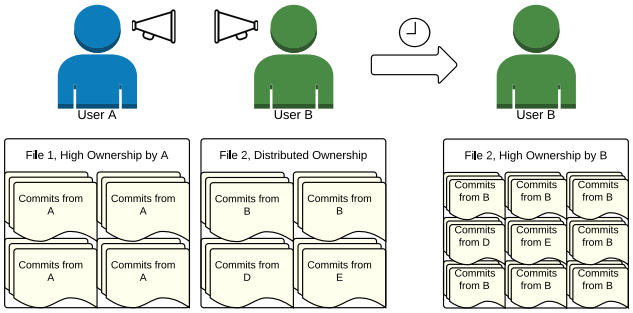
Given the technical details of SAOM modeling, and being motivated by prior work using social network analysis techniques applied to Software Engi-

[2] The problem of causal inference is not limited to the study of epidemiological processes.

[3] Also, these alternative models generally lack fundamental statistical data fitting ability.



(a) Social selection (homophily). Two users who are not initially socially connected have the same behaviors. Some time later, they connect socially due to their similar behaviors.



(b) Social influence (contagion). Two users are initially socially connected, but with different behaviors. Some time later, user B mimics the behavior of user A – in this case, high file ownership behavior.

Fig. 2: Illustration of potential mimicking behavior among developers.

neering, we sought to address applied problems related to social selection and influence in OSS developer social networks through case studies: (1) to what extent do we find homophily, *i.e.* linking of people to others with like behavior, in OSS projects? And (2) can we find evidence for behavioral influence among the nodes in our networks, *i.e.* does some technical behavior spread *vs.* arise spontaneously as a person spends more time with the project? Figure 2 illustrates this process graphically.

Next, we present the SAOM formalism and modeling techniques, followed by our illustration of its use, on cases studies of data from two OSS projects.

3 Stochastic Actor-Oriented Models

Stochastic Actor-Oriented Models (SAOMs) were developed for the analysis of longitudinal social network data, collected by taking two or more "snapshots" (also called "panels" or "waves") of a network as it evolves over time (often called a *network panel study*) [56]. There is an extensive body of work utilizing SAOMs to test hypotheses on network and behavior co-evolution, as well as much work describing its statistical basis [67,65,68,69,66,72,30,74]. A defining aspect of SAOMs is their focus on the "actors", e.g., people, firms, *etc.* SAOMs model change from the perspective of an actor within a (potentially) changing network; a major difference compared to other techniques used for modeling longitudinal network data.

Ties between actors are generally directed relationships such as trust or friendship, where ties go from "ego" (self) to "alter" (other). These models view network evolution as a series of "choices" by actors to create, maintain, or terminate ties to other actors. "Choice" does not necessarily imply conscious control; it refers to an actor's outgoing ties and behavior as explained by aspects of that particular actor and their context within the network [71]. This does not conform or restrict to any particular belief or theory in the exact overarching mechanisms of network and behavioral change [56], allowing researchers to further develop their theories behind the reasons of action (*e.g.* social selection *vs.* influence). This fact is a primary reason behind the usefulness of this method; we may hypothesize that particular software engineering behaviors are either influential or lend themselves to homophilous relations, and this methodology allows us to test these hypotheses in the presence of complex, dynamic OSS systems. Since its introduction, SAOMs have been applied to a wide variety of domains, including the study of selection patterns in school classrooms [2], the evolution of communication networks in high-risk social-ecological systems [7], the role of teen drinking behavior in friendship selection [17], and the spread of norms among judges in the French court system [41].

Informally, the model's objective function describes how likely it is for an actor to change their local network in a particular way. At each time step, the actors move in a direction that maximizes their particular objective function, given constraints on current network topology, exogenous covariates, endogenous covariates, and random influences. The process is analogous to behavioral evolution; at each step, an actor can increase their behavior ($+1$), decrease their behavior (-1), or keep their behavior the same (± 0) [66,74,71,77]. The modeled evolution is determined from (observed) wave to wave using a very large number of ministeps. Each ministep changes the underlying network state; as a result, the actors are always affecting each other throughout this evolution [56,70,85]. This allows the models to reflect the constant feedback process that exists in network dynamics.

Formally, the mathematical specification of the network objective function has the form

$$f_i^{net}(x) = \sum_k \beta_k^{net} s_{ik}^{net}(x) \tag{1}$$

where $f_i^{net}(x)$ is the value of the network objective function for actor i depending on the state x of the network. Functions $s_{ik}^{net}(x)$ are effects based on the network state (*e.g* outdegree), and the β_k^{net} are parameters to be estimated. The behavioral objective function is defined analogously, with an additional dependence on behavior value z.

$$f_i^{beh}(x, z) = \sum_k \beta_k^{beh} s_{ik}^{beh}(x, z) \tag{2}$$

The overall estimation procedure uses repeated simulations of network evolution from each wave to the next. This provides a large set of networks that "could have" brought the observed networks from one to the other. The outcome of the simulations is a *log-linked* objective function with estimated parameters β_k^{net} (or β_k^{beh}).

4 Modeling OSS Networks With SAOMs

In this work, we primarily address practical matters of implementation and assumptions that pertain to our work; much more detail can be found in the algorithm descriptions [70] and associated documents [67,65,68,69,66,72,74, 73,60,61,43,45,44].

SIENA is an R package that implements the SAOM simulation-based fitting procedures for directed or undirected one-mode networks [68]; the evolution of a two-mode network [39]; the evolution of an individual behavior; and the co-evolution of one-mode networks, two-mode networks, and individual behaviors [74,73]. The fourth model type is utilized in this work. In SIENA, in addition to network data and behavioral data, exogenous covariates may be defined which are used to guide simulation but are not modeled themselves.

To ensure that the SAOM methodology could be properly applied to our data, we follow guidelines provided by the creators of SIENA from the official SIENA manual [56], as well as the SIENA introductory tutorial [71]. In addition, we follow advice as given by users and developers of SIENA according to discussions on the official SIENA user group[4]. The assumptions of SAOM relevant to our work are: the changing network is the outcome of a Markov process; parameter estimates remain constant in time; network change exists but is not too drastic (described in more detail below); all nodes are potential network partners; and all nodes are tie senders or tie receivers at all points in time, unless specified as a "joiner" or a "leaver".

The assumption of a Markov process has been made in virtually all models for social network dynamics, as it is exceedingly difficult to devise manageable models that do not follow this assumption [71]. The assumption of parameter

[4] http://groups.yahoo.com/groups/stocnet/

estimates remaining constant in time can be relaxed through time-dependent dummy variables. However, the introduction of these variables can increase model complexity substantially. This topic has been discussed at length [43, 45] [5] The assumption that all nodes are tie senders or tie receivers at all points in time is very unlikely to affect our results, as few nodes violated this assumption, and those nodes that did violate this assumption were removed (described below and in Table 1). Here, we describe how we process our network data to meet those assumptions.

4.1 SIENA Performance and Practical Application

The SIENA package provides numerous parameters that may be specified to fine-tune the estimation process. One of the core practical features that SIENA provides is support for parallel computation. Table 2 shows computation and convergence times for our case study projects running on $2, 4, 6$, and 8 cores, with a step size of 0.02 and 3000 phase 3 iterations [70]. Computation time is the amount of elapsed time to complete one full model estimation; convergence time is the total elapsed time to run all estimations to convergence, using prior parameter estimates as initial values. The authors did not run into any issues with being memory-bounded when estimating models. As shown, the amount of time to estimate our case study models for Ant slightly increased when moving from 6 to 8 cores, which merits more study. However, the general trend is a reduction in estimation time.

As the models are stochastic, the number of estimations until convergence will likely vary between projects, and may vary between full model runs. The length of estimation depends heavily on both the size and structure of the input data. As shown in Table 1, Ant and Axis2/Java have a comparable number of nodes, but Axis2/Java experiences more frequent changes (lower average link lifetime) and has a larger number of ties between nodes, while having two less modeled waves. Though similar in size, the structure of Axis2/Java is more varied. This may explain why the estimation for Axis2/Java took much longer than the estimation for Ant.

Reducing step size will likely increase the amount of time to convergence, but may be necessary to reach convergence. As referenced in the manual [56], phase 3 is used to estimate the covariance matrix and matrix of derivatives used to calculate standard errors. A lower number of phase 3 iterations can result in inaccurate standard errors. The authors of the package recommend 3000 iterations for "publication quality" standard errors, but note that this phase takes a lot of computing time.

In addition, one can provide the results of a previous model (on the same data) as initial values for subsequent model estimations; SIENA provides this

[5] We built separate models for each consecutive wave and estimated parameters for each separate model to test this assumption. These parameter estimates were then compared across models. There were no significant departures in these tests to indicate non-constant parameter estimates across time.

functionality out-of-the-box. When building and exploring models, the authors recommend using this feature as they found this greatly reduced convergence time in practice.

4.2 Social Interaction Decay

One of the primary obstacles in applying the SAOM method to various types of network data is that there is often no clear definition of what is an ongoing social relationship. This is a problem in any type of network data in which the primary form of interaction is through *aggregated contacts*. In the social world, friendships (or more generally, social relations) are created, maintained, and lost. Similarly, knowledge can be gained, maintained, and lost. However, when using emails as a proxy for social relation or knowledge flow (*i.e.* information exchange between developers), it is unclear how to identify the *loss* of such a relation. The problem then is to identify the distribution of relational durations in order to model the loss of social relations in aggregated contact data (*e.g.* email data).

We leverage prior work by Holme arguing, through a number of case studies, that the distribution of relation durations is exponentially decaying [34]. The probability density function of an exponential distribution is

$$\lambda e^{-\lambda x}, x \geq 0$$

where λ is called the *rate parameter* [6]. For clarity, we define the inverse of λ as $\beta = \lambda^{-1}$. β can be thought of as a sort of *survival parameter*. As the expected value of an exponential distribution is $\lambda^{-1} = \beta$, β is then the expected lifetime of a unit within the system. Thus, in building our networks, we decay ties by sampling from an exponential distribution, resampling when an existing link is reinforced. A visualization of this process can be seen in Figure 3. The parameter λ that we choose is the largest value such that the Jaccard similarity [36] of consecutive network snapshots is at least 0.3 for 3 or more consecutive waves[7]. The Jaccard similarity here is used as a measure of network stability [71, 6], as suggested by the creators of SIENA. We choose λ values in intervals of 30 days, starting from $\lambda = 1/30$ ($\lambda = 0$ corresponds to no decay). Our choice of at least 3 consecutive waves is due to the algorithm that SIENA uses to estimate our models, which we refer to for a thorough description [70]. In essence, having only two consecutive waves limits the amount of variability available in the observed data, which can affect parameter estimation. Our models are then built using only those consecutive waves determined by the above logic[8].

[6] Not to be confused with the SIENA model rate parameter, described in Section 5.1.4.

[7] Note that this is equivalently the smallest such β.

[8] Axis2/Java had high fluctuations in activity (both social and technical) towards the beginning its lifetime. As a result, model estimations which included these time periods proved difficult to estimate; in particular, the number of ministeps required before arriving at the time for the next wave became too large. As we are interested in the average social and technical behaviors of projects, the offending waves were removed from the analysis for this project.

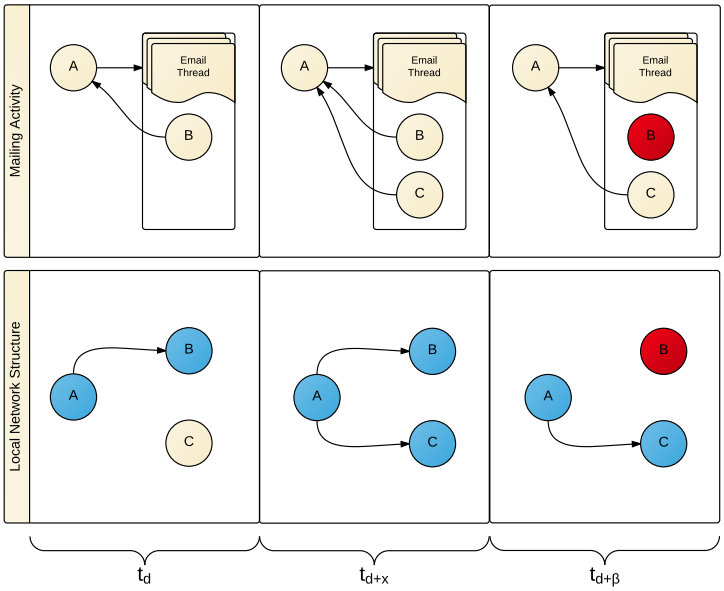


Fig. 3: Our network construction process. Evolution of an email thread (top) and the corresponding network (bottom). User A starts an email thread. B's response results in a network link $A \rightarrow B$, indicating knowledge flows from A to B. Later, C's response to A results in another link. $A \rightarrow B$ is removed from the network when β days have passed since the last response by B.

To check that our decay method indeed generates networks with typical social network properties, we calculated network *control profiles* [58] for each wave in each project. All generated networks are heavily source dominated, typical of social networks.

4.3 Baseline Wave Choice

Recall that in SAOMs, we consider "network panel waves" as sample points within our longitudinal network and behavioral data. For each wave, we have a snapshot of both the state of the network and the state of actors' behaviors. In more standard sociological data (*e.g.* survey data), waves are chosen as the points at which surveys are administered. However, with emails, this delimitation is unclear. Thus, we must decide the points in time that we wish to take snapshots of the data, as taking data snapshots at each and every point in time that an email is sent would make model simulation computationally infeasible.

With more waves comes potentially more accurate models, at the trade-off of including more noise in the data. In addition, as the amount of time between waves approaches zero, the assumption of time-constant parameter estimates becomes less correct as noise "spikes" in the data will increase which may overly influence parameter estimates. The effect of choosing time points that are farther apart is analogous to smoothing a noisy signal over time. *E.g.* in trade data, aggregating by day will introduce many more "spikes" between days as day-to-day variation may be large compared to month-to-month or year-to-year variation.

Using fewer waves can cut down on noise, at the trade-off of potentially reducing accuracy. For guidance on this matter, we consider prior work using SIENA, which is often applied to data with many fewer than 10 waves [71]. These studies also often utilize data over a much smaller period of time (*e.g.* on the scale of a few years) compared to our data. Hence, we chose to use a baseline of 8 waves, where the wave dates are chosen such that the number of emails between waves is the same. *E.g.* if there are 120 total emails, wave 1 is chosen to be the point at which a total of 15 emails have been sent, wave 2 is at the point of 30 emails, *etc*[9]. We found that using more than 8 waves generally led to instability of the estimation procedure as well as departures from the constant parameter estimate assumption (as using more baseline waves means less time between consecutive waves). Using fewer than 8 waves as a baseline also sometimes led to some instability in the estimation procedure[10,11]. Thus, we found that the choice of 8 waves as a *baseline* provided a good balance between the want of a larger sample size and the necessity of fulfilling the model assumptions.

4.4 Joiners and Leavers

Longitudinal network data that spans a long period of time, as ours does, is bound to have nodes that join and leave the network over time. We consider a node as having joined the network the moment that they acquire their first tie and as having left the network when their total tie count (total degree) drops to zero. In addition, we remove all nodes with zero total degree for all time, indicating no emails sent or received for the entirety of the project. Note that this can happen if a developer does not communicate on the email mailing list even if they contribute work (*i.e.* commits) to the project. This step is performed to adhere to the SIENA guideline that all nodes in the network must be senders or receivers (or both), unless specified as a "joiner"

[9] The authors also attempted to use an equal number of days as a separator of waves. However, this led to extreme skew and imbalance in network size (nodes and ties) as projects tend to have "burst" activity behavior; earlier waves were much less varied compared to later waves.

[10] In particular, there were instabilities in estimating the network rate parameters – the number of "chances" an actor has to change its ties.

[11] The choice of 8 waves is likely specific to our data – SIENA supports any number of waves, though time complexity increases with more waves.

or a "leaver" at a particular wave. In addition, nodes are not permitted to drop to zero total tie count more than once - we call these nodes "socially transient" and remove them. We are interested in only those nodes who are *socially* active for the entirety of their time in the project, as our focus is to study the role of ties on developer behavior (and vice-versa), and socially transient nodes may introduce noise that overly influences our model. The total number of nodes filtered by the above process can be found in Table 1. As shown, the number of "socially transient" (*i.e.* filtered) nodes is low. The number of nodes modeled in each network adheres to SIENA guidelines, as described in referenced work [71] suggesting that the number of actors should likely be more than 20.

4.5 Model Interpretation

The result of each modeling analysis is a log-linked objective function. From it we can calculate predicted probabilities by exponentiating the value of the objective function for a particular scenario, and dividing it by the sum of exponentiated objective values for all possible scenarios. An additional way to interpret this is as follows. If an actor has the chance to make a change in their outgoing ties, and the two states x_a and x_b are possible results of this change, then $f_i(x_b) - f_i(x_a)$ is the log-odds ratio for choosing between these alternatives. This means that the ratio of the probability of x_a and x_b as next states is:

$$e^{f_i(x_b)-f_i(x_a)} = \frac{e^{f_i(x_b)}}{e^{f_i(x_a)}}$$

The objective functions are typically used to *compare* how "attractive" different potential tie changes are [71].

As a simple example, consider a hypothetical model with an estimated density effect of -2.0 and an estimated reciprocity of 0.8. The associated network objective function is thus:

$$f_i^{net}(x) = \sum_j (-2.0x_{ij} + 0.8x_{ij}x_{ji})$$

Comparing between the choices of (1) adding a reciprocated tie or (2) adding a non-reciprocated tie, (1) yields: $-2.0 + 0.8 = -1.2$ while (2) (*i.e.* $x_{ji} = 0$) yields: -2.0. Thus, when comparing these two alternatives, adding a reciprocated tie is preferable to adding a non-reciprocated tie.

5 Case Studies: Applying SAOMs to ASF Project Networks

We theorize (with support from literature) that it is more likely for individuals' behavior to affect others and spread through a network if it is readily observable [57], and if the behavior is desirable and easy to imitate. The technical behaviors we examine in this work are *commit rate* (a light-weight proxy for

productivity), *high file ownership*, and *minor contributor count*. These behaviors have been studied before as important indicators of bugs, socio-technical structure, and overall software quality [22,26,82,11,55]. Bird *et al.* found that measures of ownership (such as proportion of ownership for the top owner and the number of low-expertise developers) have a relationship with both pre- and post-release failures [11]. Rahman and Devanbu studied authorship at a fine-grained level, looking particularly at specific implicated buggy code segments, finding that higher levels of ownership by a single author is associated with buggy code [55]. Additionally, they suggest that changes made by developers with little experience in a particular file (*i.e.* minor contributors) are strongly associated with defects.

On the other hand, some important technical observables, like defects in code, *e.g.*, are not clearly associated to developer actions, as demonstrated by the challenge they present in identifying [38]. They are also clearly undesirable to imitate, even if easy to find. Though a study of bug spread through a social network would perhaps be interesting, since it is not clear through which mechanisms that would happen, we think it unlikely for software bugs to spread through a socially-defined network so we chose not to study them. We contrast that to the measures of commit rate, high file ownership, and minor contributor count, which are much more readily observed through version control history and developer mailing lists.

5.1 Trace Data

The Apache Software Foundation (ASF) is a collection of hundreds of OSS projects. The primary consideration for potential case study projects is whether or not we have enough nodes (*i.e.* developers) with enough emails over time to warrant a study using SAOM. Of the 31 projects for which we had email data, we performed an initial examination of evolutionary network size, and narrowed the field to 9 potential projects.

To manage this project in terms of human and computational resources, and for the purposes of presenting the findings in a typically sized paper, we chose to model 2 projects[12], Ant, and Axis2/Java, from different subject domains with sufficient number of developers (translating to nodes) and emails over time to meet the assumptions of the SAOM. These projects were selected as they vary in network size, growth and decay rates, and are cross-domain.

For each project, data was collected from the developer mailing lists and version control systems. As we aim to model the co-evolution of social activity and technical activity, we combine both developer mailing list data and version control data to build our models. Table 1 shows descriptive statistics for the projects under study, and Figure 4 provides a diagram of our data gathering

[12] We initially built models on 3 projects: Ant, Axis2/Java, and Derby. However, Derby results were similar to Axis2/Java results (*e.g.* positive behavioral selection). As we are interested in presenting case studies of the application of the SAOM method in OSS, we only discuss results for Ant and Axis2/Java.

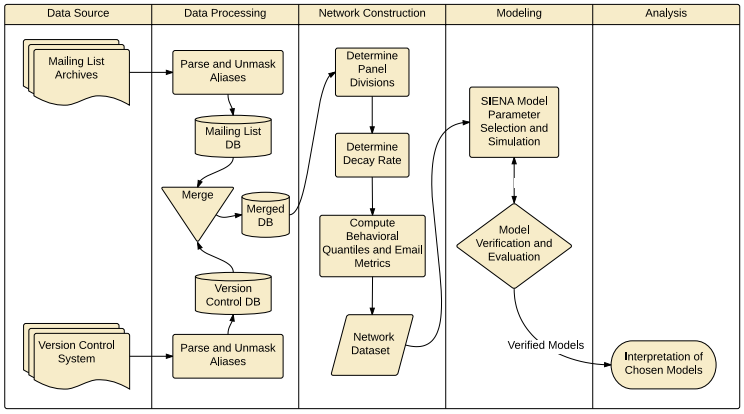


Fig. 4: Process diagram of data gathering, processing, network construction, modeling, and analysis.

processing, merging, and analysis pipeline. In this section, we describe the processes we followed to extract data from the above sources and an overview of network construction.

5.1.1 Alias Unmasking

OSS participants often use different aliases (combinations of names and email addresses) even within the same project, which can lead to problems. In addition, as we are mining data from two data sources (mailing lists and version control history), the problem can become even more difficult as participants may use different aliases across data sources. For example, a developer named John Smith may have multiple aliases (*e.g.* `<John Smith, jsmith@gmail.com>`, `<John, john.smith@gmail.com>`, `<J. Smith, jsmith@jsmith.com>`). These aliases all represent a single person, and must be treated as such in order to accurately represent an individual's activity in a project. Unmasking aliases has been addressed many times in the literature [9,76,40,27], without any perfect solution [29]. We use the method described by Gharehyazie *et al.*, which employs an enhanced version of the technique developed by Bird *et al.* [9] based on heuristics and string similarity measures. The heuristics involve "guessing" likely email prefixes based on first and last names (*e.g.* John Smith might use prefixes such as `john`, `johnsmith`, `jsmith`).

In summary, the process consists of the following steps. First, all suffixes, prefixes, and generic names *e.g.* Dr., Mr., Admin, are removed. Then, a similarity score based on Levenshtein (edit) distance is calculated for each pair of email addresses as described Bird *et al.* [9]. Perfect matches (having score

of 1) are merged automatically, and less-than-perfect matches that achieve an empirically defined threshold (based on other Apache projects to offer a good trade-off between false positives and false negatives) are studied manually, and reviewed by another researcher for accuracy.

5.1.2 Constructing Email Networks

In OSS projects, it is often the policy to direct as much communication as possible through project-specific mailing lists so all participants can benefit from the exchange of information [9]. All messages sent to this list will be broadcast to all subscribed participants. These broadcast messages differ from standard point-to-point email communication as there is no clear and distinct recipient. To generate networks using this data, we use the method described by Gharehyazie *et al.* [27] and Bird *et al.* [9] as follows. If person A is the initial creator of an email thread (broadcasting their message to all participants), and person B responds to this thread, we can say with high confidence that person B has read and acknowledged the information provided by person A. Thus, we create a network link from A to B, representing this information flow. Although this method precludes the flow of information as a result of two responses in the same thread (*e.g.* person B and person C may have knowledge transferred $B \rightarrow C$ due to a response from B to A's thread), we believe the method presented here is appropriate for the questions we wish to answer. In addition, our method provides a high level of confidence that information has indeed been passed in the direction indicated by the network.

The primary guiding principle behind our network construction logic is that we do not believe that raw emails themselves should be considered an *exact* proxy for social relations. We construct our networks to reflect an actual interaction between individuals, rather than merely reflecting that an email has been sent from one individual to another; we are not treating the emails themselves as edges in our network. Instead, the edges represent events of knowledge being passed between individuals in the network, as proxied by responses and ongoing back and forth emails. There is a theoretical "knowledge threshold" required to create an edge in our network. The initial broadcast from A to all participants does not pass this threshold; we cannot be certain that anyone who did not reply to A actually received information. Similarly, responses from B and C to post A (described above) do not create a link $B \rightarrow C$, as we cannot be certain that C has received information from B. Constructing networks in this manner provides higher confidence that we are capturing meaningful social linking between developers. The process of constructing email networks can be seen in Figure 3.

In this work we are interested in *knowledge* or *information* flow, as exhibited by developers actively discussing on the mailing lists different aspects of a project, *e.g.*, the existence of bugs, collaboration opportunities, social or code structure changes, *etc.* Studying knowledge flow does not a priori invalidate or contradict the assumptions of the SAOM modeling framework, or the SIENA tool; the emphasis of knowledge flow is merely provided to give an intuition

Table 1: Project descriptive statistics. Average values are per wave.

	Ant	Axis2/Java
Number of Unfiltered Nodes	46	78
Number of Filtered Nodes	44	69
Date Range	7/10/2001 3/16/2012	6/21/2005 3/18/2012
Average # Observed Ties	129	314
Average Link Lifetime (days)	330	120
Wave range modeled	1-7	2-6

Table 2: Computation times (seconds). Step size set to 0.02, 3000 phase 3 iterations.

	Computation Time				Convergence Time			
Number of cores	2	4	6	8	2	4	6	8
Ant	318.53	246.04	248.52	235.33	2693.43	1429.66	840.13	939.347
Axis2/Java	391.65	252.48	253.77	281.77	10344.92	6665.393	3390.189	1392.62

of how to interpret model results. Moreover, studying knowledge flow is akin to the use of SAOMs in studying the spread of *social norms*, *e.g.* norms in communication between judges in the French court system [41]. Here, the behaviors under study are analogous social norms for software, *e.g.* how much should a developer commit over time?

5.1.3 Mining and Computing Technical Behaviors

Source code repositories paired with version control systems (*e.g.* Git, SVN) allow (potentially) distributed developers to collaborate by maintaining a full history of changes and associated logs for each change. The projects studied here (Ant, Axis2/Java) all currently use Git[13] for version control, though some Apache projects migrated from using SVN to Git at some point in time.

Current Git logs include the history of changes made while the project used SVN, allowing us to reach all the way to the beginning of the project's history. This allows us to gather information for each developer at all points in time regarding *e.g.* their commits and what files these commits have touched. This information is used to compute our dependent behavioral time series for each developer for each project in our models. Note that the technical data is gathered up to a particular point in time; the "end" of our data is the temporal end of the data that we have available for study. This does not necessarily imply that work on a given project has ceased.

For our three chosen behaviors (Section 5), we defined the thresholds as follows:

1. High File Ownership: total count of files per developer where their number of commits is $\geq 70\%$ of total commits to the file
2. Minor Contributor: total count of files per developer where their number of commits is $\leq 5\%$ of total commits to the file

[13] http://git-scm.com/

Table 3: Description of ordinal transformation categories for behavioral data. All percent ranges are *exclusive-inclusive*.

Ordinal value	Description
0	No behavioral value or the same behavioral value for the rest of time (implying cessation of technical activity)
1	$0 - 25\%$ quantile raw value for a wave
2	$25 - 50\%$ quantile raw value for a wave
3	$50 - 75\%$ quantile raw value for a wave
4	$75 - 100\%$ quantile raw value for a wave

3. Commit Rate: number of commits in a time window equal to the expected link lifetime; see Section 4.2

We found no appreciable difference between high file ownership at $\geq 70\%$, $\geq 80\%$, and $\geq 90\%$, and chose $\geq 70\%$ as we believed it to be an adequate level at which to consider high ownership according to prior work [11,55]. The level at which we consider a minor contributor is also established by prior work [11]. Recall that our SAOM simulate behaviors in single steps (*i.e.* at each step, a behavior can increase by 1, decrease by 1, or stay the same). To ensure that our simulations are tractable, we transform our raw behavioral data (*e.g.* number of commits in a time window) into an ordinal form as per SIENA guidelines, described in Table 3.

5.1.4 Model Building

The model building procedure is split into multiple parts. Behavioral selection parameters are tested to be operationalized by *similarity*, *sameness*, *alter value*, *ego value*, or the *interaction between ego value and alter value*. Influence is operationalized by average alter in all cases (except in the high file ownership model for Axis2/Java; see Section 5.2). Descriptions of a set of parameters can be found in Table 4. The set of tested behavioral selection parameters chosen are the most basic representations of social selection provided by the SIENA package.

Both alter's cumulative email count (log) and alter's age (log) are used as controls for general social activity in our models, which may otherwise be confounded with social selection. We view these as necessary and important controls for all models due to the structure of the network itself. Since a link is formed between a thread creator and a thread responder when a response is made, if an actor is either a) generally socially active (represented by cumulative email count) or b) generally "old" in terms of project participation, we hypothesize that it is more likely for this actor to respond when compared to an actor who has low values of these variables. Thus, these control effects are included in all models regardless of significance. Also included in the models are the *network rate* and *behavioral rate* estimates (not shown). These values

indicate the average number of "chances" an actor has to change their outdegree or behavior, respectively, by -1, ± 0, or $+1$. The general model building procedure is guided by recommendations given by Snijders *et al.* [71] and is as follows:

1. Fit a preliminary base model consisting only of network and behavioral rate parameters, density, reciprocity, alter's cumulative email count (log), alter's age (log), linear shape, and quadratic shape.
2. Use the score-type test (STT) of Schweinberger [60] to test for significance of transitive triplets, transitive ties, and 3-cycles without estimation. Of those with significance $p < 0.05$, perform stepwise selection of parameters with highest significance and estimate the resulting models until the score-type test no longer detects significance.
3. Use the STT on indegree popularity (sqrt), outdegree popularity (sqrt), indegree activity (sqrt), outdegree activity (sqrt), number of actors at distance 2, ego's cumulative email count (log), and ego's age (log). Perform stepwise selection of these parameters as in 2) above, estimating resulting models.
4. Use the STT to test for behavioral selection effects *without estimating influence parameters*, operationalized as stated above. Keep note of all potentially significant parameters ($p < 0.05$).
5. Use the STT to test for behavioral selection effects while *controlling for influence* (*i.e.* under the model with influence estimated). Estimate the *single*[14] selection parameter that is most significant under tests from 4) and 5).

Note again that in estimating models, network dynamics and behavioral dynamics are treated simultaneously as dependent variables; they are modeled together. The SIENA model analysis yields parameter estimates for each β_k^{net} and β_k^{beh} in Equations 1 and 2.

As a concrete example, consider the model results for Axis2/Java in Table 6. The specification of the network objective function is:

$$\begin{aligned} f_i^{net}(x) = \quad & \beta_{dens}^{net} x_{i+} + \beta_{rec}^{net} \sum_j x_{ij} x_{ji} + \beta_{outa}^{net} x_{i+}^{1.5} + \beta_{inp}^{net} \sum_j x_{ij} \sqrt{\sum_h x_{hj}} \\ & + \beta_{tties}^{net} \sum_j x_{ij} max_h(x_{ih} x_{hj}) + \beta_{3cyc}^{net} \sum_{j,h} x_{ij} x_{jh} x_{hi} \\ & + \beta_{altb}^{net} \sum_j x_{ij} I\{z_i = z_j\} + \beta_{alte}^{net} \sum_j x_{ij} e_j + \beta_{alta}^{net} \sum_j x_{ij} a_j \end{aligned}$$

[14] An exception to this rule exists if the *ego X alter* interaction selection effect is most significant. In this case, one must control for the lower level structures of ego value and alter value when estimating the interaction effect. This is standard practice in general statistical modeling.

Table 4: A subset of model parameters usable in SIENA analysis. x_{ij} is an indicator variable where $x_{ij} = 1$ indicates the presence of a tie from i to j. All covariates are *mean-centered* internally by SIENA.

Parameter	Description
Transitive Triplets	The number of pairs of actors (j, h) to both of whom i is tied, while j is tied to h $s_{ik}^{net} = \sum_{j,h} x_{ij} x_{ih} x_{jh}$
Outdegree Activity (sqrt)	Reflects tendencies for actors with high outdegrees to send extra outgoing ties "because" of their high outdegrees; leads to dispersion in outdegrees $s_{ik}^{net} = x_{i+}^{1.5} = x_{i+}\sqrt{x_{i+}}$
Indegree Popularity (sqrt)	Reflects tendencies for actors with high indegrees to attract extra incoming ties "because" of their high indegrees; leads to dispersion in indegrees $s_{ik}^{net} = \sum_j x_{ij} \sqrt{\sum_h x_{hj}}$
Alter's Cumulative Email Count (log)	The log sum of cumulative email count for all actors to whom i has a tie $s_{ik}^{net} = \sum_j x_{ij} e_j$
Behavior Similarity	The sum of centered behavior similarity scores between i and the other actors j to whom he is tied where $\hat{sim}$ is the mean of all similarity scores, defined by $sim_{ij} = \frac{\Delta - \lvert z_i - z_j \rvert}{\Delta}$, with $\Delta = max_{ij} \lvert z_i - z_j \rvert$ being the observed range of the behavioral value z $s_{ik}^{net} = \sum_j x_{ij} (sim_{ij} - \hat{sim})$
Behavior Sameness	The number of ties of i to all other actors j who have *exactly the same value* of the behavioral value z $s_{ik}^{net} = \sum_j x_{ij} I\{z_i = z_j\}$
Average Alter	The product of i's behavior multiplied by the average behavior of his alters (similar to an ego-alter covariance) $s_{ik}^{beh} = z_i (\sum_j x_{ij} z_j)(\sum_j x_{ij})$

where z_j is actor j's behavioral value, e_j is actor j's cumulative email count, and a_j is actor j's age[15]. The behavioral objective function is constructed in the analogous way.

5.2 Case Study Results

We built models for each of the three behaviors for each project (Section 5.1), resulting in 6 estimated models. Tables 5 and 6 show results for tested behaviors in Ant and Axis2/Java, respectively. We omit some in-text parameter estimate reiteration, though all estimates and their significance can be seen in the respective aforementioned tables[16].

[15] Note that z_j is used here to represent actor j's behavioral value, while the z parameter is missing from the function signature. This is to emphasize that z_j here is treated as a *covariate* and is not modeled by the network objective function.

[16] For file ownership behavior in Axis2/Java (Table 6), addition of the influence effect of average alter caused high instability in estimation of the model. As a result, the model for

Table 5: SIENA Model Analysis for Ant. Bold sections indicate estimated selection and influence effects.

	Ownership		Commit Rate		Minor. Contrib. Count	
	Est.	SE	Est	SE	Est	SE
Network effects						
Density	-2.280***	0.284	-3.082***	0.244	-2.713***	0.280
Reciprocity	0.864***	0.170	0.954***	0.157	0.826***	0.187
Outdegree Activity (sqrt)	0.134*	0.062	0.161**	0.058	0.232***	0.063
Number of Actors at Distance 2	-0.143**	0.045	-0.105*	0.041	-0.114*	0.042
Ego's Cumulative Email Count (log)	-0.128**	0.042	-0.146***	0.039	-0.144**	0.046
Ego's Age in Project (log)	-0.094***	0.023	-0.058**	0.019	-0.106***	0.028
Transitivity effects						
Transitive Triplets	0.212***	0.034	0.213***	0.031	0.216***	0.044
Transitive Ties	1.175***	0.199	1.184***	0.188	1.273***	0.216
3-cycles	-0.203***	0.048	-0.189***	0.044	-0.189***	0.051
Behavioral selection effects						
Selection Parameter						
(Same, Similar, Same)	**-1.835*** **	**0.544**	**-1.150*** **	**0.296**	**-2.305*** **	**0.682**
Alter Controls						
Alter's Cumulative Email Count (log)	-0.017	0.030	-0.049+	0.027	-0.020	0.029
Alter's Age in Project (log)	-0.115***	0.022	-0.080***	0.016	-0.139***	0.026
Behavioral Effects						
Linear Shape	-2.400**	0.900	-1.194***	0.154	-1.377***	0.251
Quadratic Shape	0.478***	0.133	0.330***	0.045	0.364***	0.061
Influence Effects						
Average Alter Behavior	**1.087**	**0.691**	**0.230**	**0.168**	**0.336**	**0.235**

+ p <0.1; * p <0.05; ** p <0.01; *** p <0.001;

Table 6: SIENA Model Analysis for Axis2/Java. Bold sections indicate estimated selection and influence effects.

	Ownership		Commit Rate		Minor. Contrib. Count	
	Est.	SE	Est	SE	Est	SE
Network effects						
Density	-3.616***	0.162	-3.597***	0.190	-3.664***	0.243
Reciprocity	0.637***	0.072	0.603***	0.071	0.628***	0.076
Outdegree Activity (sqrt)	0.258***	0.019	0.247***	0.020	0.256***	0.024
Indegree Popularity (sqrt)	0.205***	0.054	0.194**	0.066	0.209*	0.104
Transitivity effects						
Transitive Ties	0.708***	0.113	0.680***	0.107	0.703***	0.152
3-cycles	0.062***	0.016	0.076***	0.016	0.060***	0.016
Behavioral selection effects						
Selection Parameter						
(Same, Same, Same)	**0.131+**	**0.079**	**0.266***	**0.107**	**0.236*** **	**0.083**
Alter Controls						
Alter's Cumulative Email Count (log)	-0.013	0.018	-0.014	0.019	-0.013	0.025
Alter's Age in Project (log)	-0.018*	0.007	-0.011	0.007	-0.013+	0.007
Behavioral Effects						
Linear Shape	-1.288***	0.135	-0.944***	0.096	-1.031***	0.297
Quadratic Shape	0.427***	0.054	0.372***	0.041	0.417***	0.078
Influence Effects						
Average Alter Behavior	—	—	**0.230**	**0.168**	**0.429**	**1.702**

+ p <0.1; * p <0.05; ** p <0.01; *** p <0.001;

5.2.1 Goodness of Fit

To assess goodness of fit, we compare simulated networks to observed networks with respect to various statistics of the networks themselves *i.e.* indegree distribution, outdegree distribution, geodesic distribution, and behavior

high file ownership behavior only includes the linear shape and quadratic shape parameters. The exclusion of this parameter in the model should not appreciably affect our outcomes or goodness of fit as the score-type test of this parameter suggested insignificance.

distribution, as proposed and elaborated by Lospinoso [44]. In essence, the method operates by comparing observed values at the ends of periods with simulated values for the ends of periods. The simplest method of presenting this comparison is through the use of violin plots [32], which use kernel density estimates to present the distribution of the simulated statistic with the observed values overlaid. The aim is to have simulated statistics which follow the observed statistics closely *i.e.* overlaid observed values lie within the confidence bands of simulated statistics. Figures 5, 6, 7, and 8 show these plots for high file ownership behavior for projects in our case study. Of those pictured, all models provide satisfactory goodness of fit except geodesic distribution, which is slightly overestimated by the models. This could be due to multiple measures of transitivity for each model. However, the fit is still satisfactory for our analysis, as we focus primarily on local structures *i.e.* at geodesic distance of 2, meaning an overestimated geodesic distribution beyond a distance of 2 should not affect our discussion. All other tested behaviors showed very similar goodness of fit to those pictured.

5.2.2 Behavioral Selection Effects

Ant: Commit rate behavior selection is best represented by negative behavioral similarity. This suggests that knowledge is less likely to flow (or be maintained) between actors with similar commit rate behavior. For high file ownership behavior (-1.835^{***}) and minor contributor behavior, behavioral selection is best represented by negative behavior sameness. This suggests that knowledge is less likely to flow (or be maintained) between actors with the same high file ownership or minor contributor behavior.

Axis2/Java: Behavioral selection is best represented by positive behavioral sameness, though this effect is only suggestive for high file ownership behavior (0.131, $p < 0.1$).

Of the two projects, though behavioral selection may be operationalized in different ways, we see positive behavioral selection effects in Axis2/Java while Ant has negative effects. This suggests that knowledge has a propensity to flow towards alters with similar or the same behavior as the ego in Axis2/Java, while knowledge tends to flow towards alters with dissimilar behavior as the ego in Ant. Behavioral selection is best operationalized by either behavioral similarity (Ant commit rate) or behavioral sameness (Ant high file ownership and minor contributor count, Axis2/Java all three behaviors). Note that behavioral similarity and behavioral sameness are very similar - similarity is calculated using a similarity score (see Table 4), while sameness is given a value only when two individuals have the exact same behavioral value. In the case of Axis2/Java, the existence of behavioral selection in the form of positive behavioral sameness suggests that there may be "knowledge circles" in Axis2/Java, based on behavior - those who exhibit the same behaviors generally have knowledge flow between them. This is discussed further in Section 6.

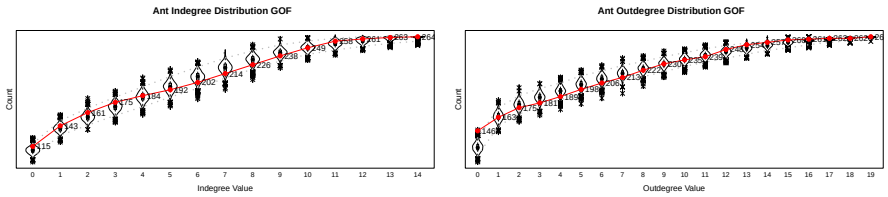


Fig. 5: Indegree (left) and outdegree (right) distribution goodness of fit plots for Ant high file ownership. Dotted lines are 95% confidence bands. Observed values at ends of periods are pictured in red. Violin plots show simulated values at ends of periods.

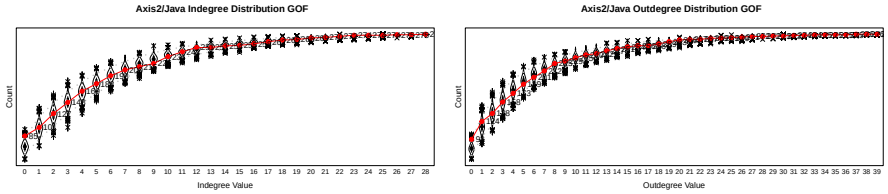


Fig. 6: Indegree (left) and outdegree (right) distribution goodness of fit plots for Axis2/Java high file ownership. Dotted lines are 95% confidence bands. Observed values at ends of periods are pictured in red. Violin plots show simulated values at ends of periods.

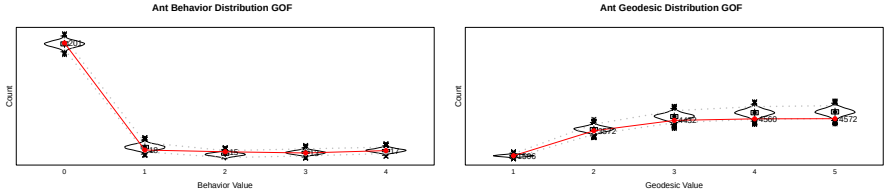


Fig. 7: Behavior (left) and geodesic (right) distribution goodness of fit plots for Ant high file ownership. Dotted lines are 95% confidence bands. Observed values at ends of periods are pictured in red. Violin plots show simulated values at ends of periods.

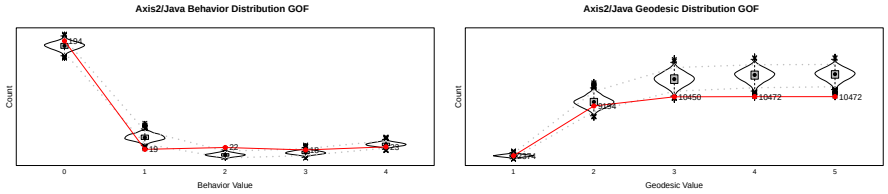


Fig. 8: Behavior (left) and geodesic (right) distribution goodness of fit plots for Axis2/Java high file ownership. Dotted lines are 95% confidence bands. Observed values at ends of periods are pictured in red. Violin plots show simulated values at ends of periods.

5.2.3 Network, Transitivity, and Influence Effects

Reciprocity is significant and positive for all models, and density is significant and negative for all models. Reciprocity is considered a basic feature of most social networks [80]. Thus, it is not surprising that reciprocity is highly significant in our models. Abbreviations used below are for transitive triplets (ttrip), transitive ties (tties), and 3-cycles (3c).

For all behaviors, we see a net push towards higher transitivity (Ant high file ownership ttrip 0.212***, tties 1.175***, 3c −0.203***; Axis2/Java high file ownership tties 0.708***, 3c 0.062***). In addition, for all behaviors and all projects, we see positive and significant outdegree activity effects. This suggests that knowledge sources with high outdegree have a tendency to increase their outdegree. Additionally, in Axis2/Java, we see a positive effect for indegree popularity, suggesting high dispersion in both outdegree and indegree. In all cases, this indicates the existence of clustering. Generally speaking, most social networks have a tendency towards transitivity or clustering [71]. In graph theoretical terms, two-paths tend to become closed; colloquially, friends of friends often become friends themselves [33].

The above results together with behavioral selection results show that the high-level social structure of these projects is relatively constant even across multiple behavior dependent variables (*i.e.* transitivity, reciprocity, clustering of degree)[17]. Yet, in the presence of these similarities, nuanced differences (*i.e.* different social selection tendencies) shine through. This shows the power of the SAOM method.

We see no significant effect of network structure on behavior evolution in Ant (average alter Est. 1.087 SE 0.691) or for Axis2/Java[18] In other words, we find no behavioral influence effect in all projects for all behaviors tested.

5.2.4 Ego Email, Ego Age, and Alter Control Effects

Ant: For email measures, we see a negative effect of ego's cumulative email count (high file ownership −0.128**) with an insignificant alter effect. A positive effect for outdegree activity (as found in this project) suggests that those who create many email threads or incite discussion are likely to continue that trend. However, a negative effect of ego's cumulative email count suggests that this self-enforcing growth mechanism may not be too strong. We also see significant and negative ego (high file ownership −0.094***) and alter (high file ownership −0.115***) age effects for all behaviors, suggesting that actors who are older are less likely to create or maintain ties, and receivers of ties are less likely to be older. This can be interpreted as an eventual "social isolation" of older users.

[17] Evidence of clustering initially raised a concern with the authors that the constructed networks had extreme levels of clustering. Further analysis showed that this was not the case; the clustering is at an acceptable level according to prior work in these social networks.

[18] Recall that we did not estimate average alter influence for the high file ownership model in this case.

Axis2/Java: We see insignificant alter email effects for all three behaviors, and a significant negative alter age parameter only for high file ownership behavior.

For those projects in which alter age is significant (Ant in all three behaviors, Axis2/Java in high file ownership), the effect is negative. This suggests that knowledge is more likely to flow in the direction of lower age while controlling for the mentioned behaviors.

5.2.5 Behavioral Effects

For all models in Ant and Axis2/Java, we see a significant and negative linear shape, indicating that developers are more likely to score below the mean for the respective behavior; in other words, developers tend to drift towards lower values of respective behaviors. In addition, we see a significant and positive quadratic shape for all models, suggesting that changes in behavioral value are self-reinforcing; those who are high in their behavior are more likely to further increase their behavior, and those who are low in their behavior are more likely to further decrease their behavior. This can be an indication of addictive behavior (when it is possible for such a behavior to be addictive) [56].

6 Discussion

The findings in this work show details of deeply embedded social phenomena in open source projects that have not been adequately or robustly identifiable in prior work. As shown above, the high-level social structure (*i.e.* transitivity, reciprocity, clustering of degree) of the projects under study is relatively constant even across multiple behavior dependent variables. However, though the projects may all have similar high-level social structure, each has fine-grained attributes (*e.g.* specific social selection dynamics discussed in Section 5.2.2) that are different from one another. Our ability to identify this is in large part due to the flexibility provided by the SAOM method. Previous statistical dynamic network models – specifically those of Wasserman [79] and Wasserman and Iacobucci [81] – do not allow complicated dependencies between ties such as those generated by transitive closure [71]. SIENA has *e.g.* transitivity in its library of measures, allowing one to control for it while testing for other tendencies (*e.g.* homophily) that may be generically confounded [62].

An example of this flexibility is presented through evidence of "knowledge circles" in some projects. We define a "knowledge circle" as a group of clustered individuals who share knowledge with one another. In Axis2/Java, this is evidenced by positive behavioral sameness and high clustering (outdegree activity, indegree popularity, and transitivity). This indicates that knowledge flows within clustered groups between those with the same behavior (in Axis2/Java). On the other hand, for Ant, selection effects are negative for all three behaviors, with evidence of clustering. This indicates that the "knowledge circles" in Ant are composed of individuals with dissimilar behaviors. The

existence of "knowledge circles" is important as they are a unique attribute that can be used to describe differences in organizational structure and the social dynamic within a project. This information can be used to determine whether particular goals of a project are being met.

The results discussed in Section 5.2.3 (*i.e.* clustering) are typical of "standard" social networks, such as friendship networks [33, 23]. We expect a negative effect for density as our networks have a bounded size; a positive and significant effect would suggest unbounded growth in degree. As a result, we can safely say that our networks obey the general guidelines of what is considered a "social network", and can be analyzed as such.

6.1 SAOM in a Software Engineering Context

The direction of knowledge flow combined with information regarding overall network structure (*e.g.* "knowledge circles") can be an important indicator of overall "health" of an open source project. Whether or not this structure is an indicator of positive or negative "health" of a project depends on one's philosophical standpoint on the "correct" structure of open source projects. In a project that aims to have an equitable dissemination of information to all participants, high clustering and knowledge flows between those with the same behavior (as is the case in Axis2/Java) could be an indicator of negative health; in a project that aims to have a hierarchical structure (as in the standard description of the onion model of team structure), this can be an indicator of positive health. In contrast to the case in Axis2/Java, one may believe that a structure which promotes interaction between those with differing behaviors is desirable (as in Ant, evidenced by negative behavioral selection estimates for all behaviors). These organizational structures all have their merits, and whether or not the structure of a given project is "healthy" depends on the goals of the project itself. SAOMs and SIENA allow for the identification of such complex structures in the presence of confounds and are thus useful in the realm of software engineering.

In our analysis of age effects (Section 5.2.4), we found that when alter age is significant, the effect is negative; this means that knowledge is less likely to flow in the direction of higher age. Due to the way that our networks are structured, this result implies that those who respond to email threads are younger in project age. It may be that these individuals are eager to convey their social or technical prowess, so they are more motivated than those who are older in project age to engage in social activity. This idea is corroborated by work showing that social activity plays a major role in advancing one's status in OSS [28]. As is the case with behavioral selection and clustering, whether or not this is a "healthy" structure depends on one's philosophical standpoint and the goals of a given project. In addition, in Ant we see evidence of eventual "social isolation" of developers over time, evidenced by significant negative ego and alter age effects for all behaviors. The exact cause of this phenomenon could be a focus of future work.

As shown in Table 2, the computation time for model estimation on real projects is reasonable. The authors believe that this method can be applied to even larger data sets with relative ease, as the computation time barrier is relatively low. OSS projects are extremely complex, dynamic systems, which evolve at varying rates with differences at a very detailed level. The ability to extract such fine-grained results (*e.g.* evidence of social selection and possible social isolation) in the presence of potential confounds (*e.g.* transitivity) in a reasonable amount of time illustrates the usefulness of SIENA and general SAOM methods in the field of software engineering.

7 Conclusion and Threats to Validity

The SAOM methodology is a useful tool in general: it allows for the study of network architecture and behavior co-evolution within complex, dynamic networks, where identifying potential confounds is otherwise a very difficult task. In particular, it allows researchers to explicitly model behavioral influence and behavioral selection, two effects which are generically confounded in simpler models.

We found no evidence of behavioral influence for all projects and all behaviors tested. However, except for high file ownership behavior in Axis2/Java where there was a suggestive effect, we found significant effects of selection (homophily) in all projects for all behaviors, all positive for Axis2/Java and all negative for Ant. Note that the existence of behavioral selection in software projects shouldn't be interpreted as "good news" or "bad news"; whether or not its effect is beneficial or detrimental depends heavily on the dynamics of the particular project and if its effects are desired. We believe the methods used in this work can be applied to the analysis of process models and organizational structure within OSS projects.

In addition, we found that all projects share high-level features (*e.g.* a push towards higher clustering), while differing in fine-grained details such as the best selection parameter, the sign of selection parameter, *etc.* This is important as it characterizes some of the differences between project ecosystems which may otherwise be difficult to isolate due to the great complexity of network and behavioral co-evolution. This also shows the power of the SAOM methodology – separation of complex features is an extremely useful feature when studying a system as varied and complicated as OSS. We believe that the methodology here would be useful to a software engineering researcher as it explicitly models effects that may otherwise be generically confounded, allowing direct testing of fine-grained hypotheses in the presence of complex interactions over time. We have shown that the SAOM approach is applicable to longitudinal software engineering data, and look forward to future applications of this methodology in software engineering.

We note a number of threats to validity to our conclusions. Even though we chose our case study projects so as to be as diverse as possible they are all from ASF, a tightly knit community with well defined rules, and consequently we are

likely capturing less variance than that in the full OSS project space. Choosing a small number of projects was deliberate, imposed by the computationally intensive SIENA algorithms and manually intensive model building procedure.

We also acknowledge issues with the way we assemble our email networks based on the discussion boards posts. Developers use a varied number of communication methods, with email being one of them. The inclusion of other means of communication *e.g.* IRC chat logs could strengthen our work. Additionally, our particular method of network construction (Figure 3), though following prior work, is just one realization of the possible constructions. Another potential construction is to have a tie go from the thread creator to every mailing list subscriber. These threats, while not trivial, have been noted before [27,9].

Though SAOMs were theoretically created to be adaptable to a multitude evolving network situations, applications have been mostly limited to the realm of sociology (*e.g.* friendship networks) where data is often gathered using *e.g* surveys. By nature, sociological data is a different type of data than that generally found in OSS *e.g.* Git history. For example, surveys generally have a fixed time point of gathering, which sets the point at which a "wave" is defined. Differences in the nature of the data should not be an issue theoretically, and we have gone through great lengths to ensure our data meets the theoretical assumptions of SAOM. However, as with any technology applied in a novel domain one must remain vigilant and perhaps even skeptical about initial results. Whereas modeling with SIENA has been around long enough for most kinks to have been ironed out, we had to make a few choices that were imposed by the specifics of our data. In particular, our choice for the appropriate length and distribution of panel waves could use a more objective analysis and validation. Likewise for the social interaction decay function, which even though is literature supported, may depend on the project or even period in the project. And, finally, the modeling and goodness of fit approaches can benefit from more variance and larger data sets.

Acknowledgements The authors want to thank Mohammad Gharehyazie for sharing his ASF project data. We thank Saheel Godhane for fruitful discussion during the early stages of the project. We thank anonymous reviewers for their helpful suggestions on prior versions of manuscript. The authors gratefully acknowledge support from the Air Force Office of Scientific Research, award FA955-11-1-0246, and a faculty grant from UC Davis. The authors are thankful for generous support from UC Davis.

References

1. Anderson, R.M., May, R.M., Anderson, B.: Infectious diseases of humans: dynamics and control, vol. 28. Wiley Online Library (1992)
2. Baerveldt, C., de la Rúa, F., Van de Bunt, G.G., et al.: Why and how selection patterns in classroom networks differ between students. the potential influence of networks size preferences, level of information, and group membership. In: Redes: revista hispana para el análisis de redes sociales, vol. 19, pp. 0273–298 (2010)
3. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. science **286**(5439), 509–512 (1999)

4. Barthélemy, M., Barrat, A., Pastor-Satorras, R., Vespignani, A.: Dynamical patterns of epidemic outbreaks in complex heterogeneous networks. Journal of theoretical biology **235**(2), 275–288 (2005)
5. Basili, V.R., Caldiera, G.: Improve software quality by reusing knowledge and experience. Sloan Management Review pp. 55–64 (1995)
6. Batagelj, V., Bren, M.: Comparing resemblance measures. Journal of classification **12**(1), 73–90 (1995)
7. Berardo, R.: The evolution of self-organizing communication networks in high-risk social-ecological systems. International Journal of the Commons **8**(1), 236–258 (2014)
8. Bettenburg, N., Hassan, A.E.: Studying the impact of social structures on software quality. In: Program Comprehension (ICPC), 2010 IEEE 18th International Conference on, pp. 124–133. IEEE (2010)
9. Bird, C., Gourley, A., Devanbu, P., Gertz, M., Swaminathan, A.: Mining email social networks. In: Proceedings of the 2006 international workshop on Mining software repositories, pp. 137–143. ACM (2006)
10. Bird, C., Nagappan, N., Gall, H., Murphy, B., Devanbu, P.: Putting it all together: Using socio-technical networks to predict failures. In: Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on, pp. 109–119. IEEE (2009)
11. Bird, C., Nagappan, N., Murphy, B., Gall, H., Devanbu, P.: Don't touch my code!: examining the effects of ownership on software quality. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, pp. 4–14. ACM (2011)
12. Bird, C., Pattison, D., D'Souza, R., Filkov, V., Devanbu, P.: Latent social structure in open source projects. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pp. 24–35. ACM (2008)
13. Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.U.: Complex networks: Structure and dynamics. Physics reports **424**(4), 175–308 (2006)
14. CAD, C.A.D.: A generai theory of bibiiometric and other cumulative advantage processes. Journal of the American society for Information science p. 293 (1976)
15. Cardy, J.L., Grassberger, P.: Epidemic models and percolation. Journal of Physics A: Mathematical and General **18**(6), L267 (1985)
16. Cataldo, M., Wagstrom, P.A., Herbsleb, J.D., Carley, K.M.: Identification of coordination requirements: implications for the design of collaboration and awareness tools. In: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work, pp. 353–362. ACM (2006)
17. Cheadle, J.E., Stevens, M., Williams, D.T., Goosby, B.J.: The differential contributions of teen drinking homophily to new and existing friendships: an empirical assessment of assortative and proximity selection mechanisms. Social science research **42**(5), 1297–1310 (2013)
18. Cherry, S., Robillard, P.N.: The social side of software engineeringa real ad hoc collaboration network. International Journal of Human-Computer Studies **66**(7), 495–505 (2008)
19. Cohen-Cole, E., Fletcher, J.M.: Detecting implausible social network effects in acne, height, and headaches: longitudinal analysis. Bmj **337** (2008)
20. Cohen-Cole, E., Fletcher, J.M.: Is obesity contagious? social networks vs. environmental factors in the obesity epidemic. Journal of health economics **27**(5), 1382–1387 (2008)
21. Crowston, K., Howison, J.: The social structure of free and open source software development. First Monday **10**(2) (2005)
22. Curtis, B., Krasner, H., Iscoe, N.: A field study of the software design process for large systems. Communications of the ACM **31**(11), 1268–1287 (1988)
23. Davis, J.A.: Clustering and hierarchy in interpersonal relations: Testing two graph theoretical models on 742 sociomatrices. American Sociological Review pp. 843–851 (1970)
24. De Souza, C., Froehlich, J., Dourish, P.: Seeking the source: software source code as a social and technical artifact. In: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work, pp. 197–206. ACM (2005)
25. Ducheneaut, N.: Socialization in an open source software community: A socio-technical analysis. Computer Supported Cooperative Work (CSCW) **14**(4), 323–368 (2005)
26. Fong Boh, W., Slaughter, S.A., Espinosa, J.A.: Learning from experience in software development: A multilevel analysis. Management Science **53**(8), 1315–1331 (2007)

27. Gharehyazie, M., Posnett, D., Filkov, V.: Social activities rival patch submission for prediction of developer initiation in oss projects. In: Software Maintenance (ICSM), 2013 29th IEEE International Conference on, pp. 340–349. IEEE (2013)
28. Gharehyazie, M., Posnett, D., Vasilescu, B., Filkov, V.: Developer initiation and social interactions in oss: A case study of the apache software foundation. Empirical Software Engineering pp. 1–36 (2014)
29. Goeminne, M., Mens, T.: A comparison of identity merge algorithms for software repositories. Science of Computer Programming **78**(8), 971–986 (2013)
30. Greenan, C.C.: Diffusion of innovations in dynamic networks. Journal of the Royal Statistical Society: Series A (Statistics in Society) (2014)
31. Halliday, T.J., Kwak, S.: Weight gain in adolescents and their peers. Economics & Human Biology **7**(2), 181–190 (2009)
32. Hintze, J.L., Nelson, R.D.: Violin plots: a box plot-density trace synergism. The American Statistician **52**(2), 181–184 (1998)
33. Holland, P.W., Leinhardt, S.: Transitivity in structural models of small groups. Comparative Group Studies (1971)
34. Holme, P.: Network dynamics of ongoing social relationships. EPL (Europhysics Letters) **64**(3), 427 (2003)
35. Hong, Q., Kim, S., Cheung, S., Bird, C.: Understanding a developer social network and its evolution. In: Software Maintenance (ICSM), 2011 27th IEEE International Conference on, pp. 323–332. IEEE (2011)
36. Jaccard, P.: The distribution of the flora in the alpine zone. 1. New phytologist **11**(2), 37–50 (1912)
37. Jackson, M.O., Rogers, B.W.: Meeting strangers and friends of friends: How random are social networks? The American economic review pp. 890–915 (2007)
38. Johnson, B., Song, Y., Murphy-Hill, E., Bowdidge, R.: Why don't software developers use static analysis tools to find bugs? In: Software Engineering (ICSE), 2013 35th International Conference on, pp. 672–681. IEEE (2013)
39. Koskinen, J., Edling, C.: Modelling the evolution of a bipartite networkpeer referral in interlocking directorates. Social Networks **34**(3), 309–322 (2012)
40. Kouters, E., Vasilescu, B., Serebrenik, A., van den Brand, M.G.: Who's who in gnome: Using lsa to merge software repository identities. In: Software Maintenance (ICSM), 2012 28th IEEE International Conference on, pp. 592–595. IEEE (2012)
41. Lazega, E., Mounier, L., Tubaro, P., et al.: Norms, advice networks and joint economic governance: the case of conflicts among shareholders at the commercial court of paris. Does economic governance matter pp. 46–70 (2011)
42. Lopez-Fernandez, L., Robles, G., Gonzalez-Barahona, J.M., et al.: Applying social network analysis to the information in cvs repositories. In: International Workshop on Mining Software Repositories, pp. 101–105. IET (2004)
43. Lospinoso, J.: Testing and modeling time heterogeneity in longitudinal studies of social networks: A tutorial in rsiena (2010)
44. Lospinoso, J.: Statistical models for social network dynamics. Ph.D. thesis, Oxford University (2012)
45. Lospinoso, J.A., Schweinberger, M., Snijders, T.A., Ripley, R.M.: Assessing and accounting for time heterogeneity in stochastic actor oriented models. Advances in data analysis and classification **5**(2), 147–176 (2011)
46. Madey, G., Freeh, V., Tynan, R.: The open source software development phenomenon: An analysis based on social network theory. AMCIS 2002 Proceedings p. 247 (2002)
47. Manski, C.F.: Identification of endogenous social effects: The reflection problem. The review of economic studies **60**(3), 531–542 (1993)
48. Meneely, A., Williams, L.: Secure open source collaboration: an empirical study of linus' law. In: Proceedings of the 16th ACM conference on Computer and communications security, pp. 453–462. ACM (2009)
49. Meneely, A., Williams, L., Snipes, W., Osborne, J.: Predicting failures with developer networks and social network analysis. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pp. 13–23. ACM (2008)
50. Mockus, A.: Large-scale code reuse in open source software. In: Emerging Trends in FLOSS Research and Development, 2007. FLOSS'07. First International Workshop on, pp. 7–7. IEEE (2007)

51. Nagappan, N., Murphy, B., Basili, V.: The influence of organizational structure on software quality: an empirical case study. In: Proceedings of the 30th international conference on Software engineering, pp. 521–530. ACM (2008)
52. Newman, M.E.: Spread of epidemic disease on networks. Physical review E **66**(1), 016,128 (2002)
53. Pastor-Satorras, R., Vespignani, A.: Epidemic spreading in scale-free networks. Physical review letters **86**(14), 3200 (2001)
54. Pinzger, M., Nagappan, N., Murphy, B.: Can developer-module networks predict failures? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pp. 2–12. ACM (2008)
55. Rahman, F., Devanbu, P.: Ownership, experience and defects: a fine-grained study of authorship. In: Proceedings of the 33rd International Conference on Software Engineering, pp. 491–500. ACM (2011)
56. Ripley, R.M., Snijders, T.A., Boda, Z., Vörös, A., Preciado, P.: Manual for siena version 4.0. University of Oxford (2014)
57. Rogers, E.M.: Diffusion of innovations. Simon and Schuster (2010)
58. Ruths, J., Ruths, D.: Control profiles of complex networks. Science **343**(6177), 1373–1376 (2014)
59. Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K.: Understanding free/open source software development processes. Software Process: Improvement and Practice **11**(2), 95–105 (2006)
60. Schweinberger, M.: Statistical modelling of network panel data: Goodness of fit. British Journal of Mathematical and Statistical Psychology **65**(2), 263–281 (2012)
61. Schweinberger, M., Snijders, T.A.: Markov models for digraph panel data: Monte carlo-based derivative estimation. Computational statistics & data analysis **51**(9), 4465–4483 (2007)
62. Shalizi, C.R., Thomas, A.C.: Homophily and contagion are generically confounded in observational social network studies. Sociological Methods & Research **40**(2), 211–239 (2011)
63. Shi, H., Duan, Z., Chen, G.: An sis model with infective medium on complex networks. Physica A: Statistical Mechanics and its Applications **387**(8), 2133–2144 (2008)
64. Singh, P.V.: The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success. ACM Transactions on Software Engineering and Methodology (TOSEM) **20**(2), 6 (2010)
65. Snijders, T., van Duijn, M.: Simulation for statistical inference in dynamic network models. In: Simulating social phenomena, pp. 493–512. Springer (1997)
66. Snijders, T., Steglich, C., Schweinberger, M.: Modeling the coevolution of networks and behavior. na (2007)
67. Snijders, T.A.: Stochastic actor-oriented models for network change. Journal of mathematical sociology **21**(1-2), 149–172 (1996)
68. Snijders, T.A.: The statistical evaluation of social network dynamics. Sociological methodology **31**(1), 361–395 (2001)
69. Snijders, T.A.: Models for longitudinal network data. Models and methods in social network analysis **1**, 215–247 (2005)
70. Snijders, T.A.: Siena algorithms (2014)
71. Snijders, T.A., Van de Bunt, G.G., Steglich, C.E.: Introduction to stochastic actor-based models for network dynamics. Social networks **32**(1), 44–60 (2010)
72. Snijders, T.A., Koskinen, J., Schweinberger, M., et al.: Maximum likelihood estimation for social network dynamics. The Annals of Applied Statistics **4**(2), 567–588 (2010)
73. Snijders, T.A., Lomi, A., Torló, V.J.: A model for the multiplex dynamics of two-mode and one-mode networks, with an application to employment preference, friendship, and advice. Social networks **35**(2), 265–276 (2013)
74. Steglich, C., Snijders, T.A., Pearson, M.: Dynamic networks and behavior: Separating selection from influence. Sociological methodology **40**(1), 329–393 (2010)
75. Storey, M.A., Treude, C., van Deursen, A., Cheng, L.T.: The impact of social media on software engineering practices and tools. In: Proceedings of the FSE/SDP workshop on Future of software engineering research, pp. 359–364. ACM (2010)

76. Vasilescu, B., Serebrenik, A., Goeminne, M., Mens, T.: On the variation and specialisation of workloada case study of the gnome ecosystem community. Empirical Software Engineering **19**(4), 955–1008 (2014)
77. Veenstra, R., Dijkstra, J.K., Steglich, C., Van Zalk, M.H.: Network–behavior dynamics. Journal of Research on Adolescence **23**(3), 399–412 (2013)
78. Vespignani, A.: Modelling dynamical processes in complex socio-technical systems. Nature Physics **8**(1), 32–39 (2012)
79. Wasserman, S.: A stochastic model for directed graphs with transition rates determined by reciprocity. Sociological methodology **11**, 392–412 (1980)
80. Wasserman, S.: Social network analysis: Methods and applications, vol. 8. Cambridge university press (1994)
81. Wasserman, S., Iacobucci, D.: Sequential social network data. Psychometrika **53**(2), 261–282 (1988)
82. Weyuker, E.J., Ostrand, T.J., Bell, R.M.: Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models. Empirical Software Engineering **13**(5), 539–559 (2008)
83. Xuan, Q., Devanbu, P.T., Filkov, V.: Converging work-talk patterns in online task-oriented communities. arXiv preprint arXiv:1404.5708 (2014)
84. Xuan, Q., Filkov, V.: Building it together: synchronous development in oss. In: Proceedings of the 36th International Conference on Software Engineering, pp. 222–233. ACM (2014)
85. Zeggelink, E.: Dynamics of structure: an individual oriented approach. Social Networks **16**(4), 295–333 (1994)
86. Zhang, H., Fu, X.: Spreading of epidemics on scale-free networks with nonlinear infectivity. Nonlinear Analysis: Theory, Methods & Applications **70**(9), 3273–3278 (2009)