Article

Organizational Research Methods
2014, Vol. 17(1) 23-50

DOI: 10.1177/1094428113517007
orm.sagepub.com

Modeling Relational Events: A Case Study on an Open Source Software Project

Eric Quintane[1,2], Guido Conaldi[3], Marco Tonellato[4,5], and Alessandro Lomi[4]

Abstract

Sequences of relational events underlie much empirical research on organizational relations. Yet relational event data are typically aggregated and dichotomized to derive networks that can be analyzed with specialized statistical methods. Transforming sequences of relational events into binary network ties entails two main limitations: the loss of information about the order and number of events that compose each tie and the inability to account for compositional changes in the set of actors and/or recipients. In this article, we introduce a newly developed class of statistical models that enables researchers to exploit the full information contained in sequences of relational events. We propose an extension of the models to cater for sequences of relational events linking different sets of actors. We illustrate the empirical application of relational event models in the context of a free/open source software project with the aim to explain the level of effort produced by contributors to the project. We offer guidance in the interpretation of model parameters by characterizing the social processes underlying organizational problem solving. We discuss the applicability of relational events models in organizational research.

Keywords

relational event models, temporal dependence, two-mode networks, free/open source software

A considerable variety of data relevant to empirical organizational research present themselves in the form of sequences of relational events, or actions connecting a sender and a recipient unit at a specific point in time. For example, relational events may be determined by communication

[1]Institute of Management, University of Italian Switzerland, Lugano, Switzerland
[2]School of Management, University of Los Andes, Bogota, Colombia
[3]Center for Business Network Analysis, University of Greenwich, London, UK
[4]University of Italian Switzerland, Lugano, Switzerland
[5]Center for Organizational Learning, Innovation and Knowledge, Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

Corresponding Author:
Eric Quintane, School of Management, University of Los Andes, Calle 21 # 1 - 20, Bogota, Colombia.
Email: eric@quintane.net

instances (e.g., phone calls, emails, face-to-face conversation, instant messages) that connect individuals in organizations (Aral & Van Alstyne, 2011). When communication events are observed between two organizational members, they are typically represented as a tie in a network of communication relations (e.g., M. K. Ahuja & Carley, 1999). Relational events may also connect organizational members to organizational tasks, activities, or issues. For example, justices in the Supreme Court are frequently represented as connected to cases exemplifying various issues (Breiger, 2000). In this circumstance the aggregation of sequences of relational events gives rise to a two-mode network of individuals by issues (Breiger & Mohr, 2004).

As these motivating examples illustrate, sequences of relational events are rarely analyzed directly. Rather, they are typically aggregated and dichotomized into binary network ties connecting the units of interest (Conaldi, Lomi, & Tonellato, 2012; Grund, 2012; Quintane & Kleinbaum, 2011). Scholars interested in statistical modeling of relational events that link different sets of actors currently employ two main analytical strategies. The first is based on the direct statistical analysis of two-mode dyads, which represent the smallest components of any network. Each network tie is treated as an independent dyadic observation in the context of traditional statistical models for binary data, such as fixed or random effects logistic regressions. These models allow researchers to maintain the original structure of the observations and their order and frequency. However, they are incapable of taking into consideration extradyadic dependencies that are crucial when modeling any network data structure (Conaldi et al., 2012). Hence, models that assume dyadic independence are not well suited for network data structures in general, because accounting for dependence between dyads is key in network analysis (see Lusher, Koskinen, & Robins, 2012). An alternative strategy lies in the adoption of recently developed frameworks—Exponential Random Graph Models (ERGMs) and Stochastic Actor-Oriented Models (SAOMs) in which the endogenous dynamics of two-mode networks are directly modeled. However, in this case researchers have to aggregate and dichotomize their data sets to use the two main statistical frameworks.

Despite the widespread adoption of this practice, aggregating sequences of relational events into binary network ties connecting fixed sets of units has two major analytical disadvantages. First, aggregation of the sequence of relational events into binary ties results in the loss of information about the temporal order and the number of events that compose a tie. More specifically, aggregation and dichotomization of relational events into ties precludes the possibility of (a) distinguishing between the different sequences that could have led to a given network structure and (b) retrieving information about the number of events that compose each tie in the network, sometimes called the "weight" of the network tie. This is problematic whenever the ordering of events in time or the frequency of certain events contains relevant information about organizational processes. Second, the inability to represent compositional changes in the set of organizational members and/or tasks is generally problematic because network dependencies are weakened by demographic rotation in the set of relevant actors. More specifically, assuming a fixed set of actors makes it difficult to (a) account for the changes in the exposure to the risk of producing (or receiving) an event and (b) distinguish between instances where actors' presence in a setting occurs at the same time from instances where presence is sequential.

Against the backdrop of these general methodological and substantive problems, this article aims to make three contributions to the extant literature on statistical methods for organizational network research. First, we extend the toolset of organizational research methods with a newly derived class of statistical models specifically suited for relational event data structures. While sequences of relational events are common in organizational research and social sciences in general (Lazer et al., 2009), current modeling frameworks require the transformation of these data structures into binary ties measured at one or a discrete number of time points (Conaldi et al., 2012; Quintane & Kleinbaum, 2011). We introduce Butts's (2008) Relational Event Modeling framework (REM) as a way to model directly sequences of relational events without need for aggregation, as is required

by ERGM or SAOM frameworks. We present the rationale behind the REM and explain how it differs from two other recent frameworks for the modeling of relational events: de Nooy's (2011) multilevel event history framework and Stadtfeld and Geyer-Schulz's (2011) Markov actor-based framework.

A second contribution of this article is to extend Butts's (2008) REM, originally derived for the analysis of events linking the same set of actors (one-mode networks), to the analysis of events linking different sets of actors (two-mode networks). The study of two-mode networks is common in the organizational literature. Examples include work done in the entertainment industry (e.g., Cattani & Ferriani, 2008; Zaheer & Soda, 2009) where actors are linked to movies or TV series. In coauthorship (e.g., Leydesdorff, 1995) and copatenting (e.g., Hess & Rothaermel, 2011) networks, actors are linked to the patent(s) or article(s) that they co-contributed to. In the study of board interlocks, board members are linked to companies (e.g., Robins & Alexander, 2004). Finally, studies in Free/Open Source Software (F/OSS) communities consider members of the community to be linked to issues (Stewart, 2005). Because the one-mode projection of two-mode networks is both common and problematic, the creation of specific statistics for the study of two-mode networks is important for the organizational literature (Conaldi et al., 2012). Hence, management scholars interested in understanding the interdependent processes that drive the patterns of affiliation in a two-mode network have extended the ERGM (Lusher et al., 2012) or SAOM (Koskinen & Edling, 2012) frameworks to two-mode network data. We propose a similar extension of the REM framework to sequences of relational events linking different sets of senders and recipients, and present a set of predictive statistics to that effect.

A third contribution of this article is to provide a real-world example of the use a relational event model to demonstrate the relevance and applicability of the REM approach to organizational research. Because of the novelty of the framework, very few empirical applications are available and little guidance is provided regarding specification, estimation, and interpretation of relational event models (for examples, see Brandes, Lerner, & Snijders, 2009; Butts, 2008; Quintane, Pattison, Robbins, & Mol, 2013). We didactically illustrate the empirical value of the model in the context of an F/OSS project. F/OSS projects involve virtual communities of contributors who take part voluntarily in the production of software that is then freely distributed. In this setting, the activity of fixing software bugs regularly carried out by contributors is ideally suited to illustrate the empirical relevance of the model we present. This is because organizational problem-solving activities—of which bug fixing is an example—unfold as temporally ordered sequences of relational events linking problem solvers (contributors) and problems (software bugs). Furthermore, this empirical context enables us to demonstrate various features of the modeling approach, such as taking into account large compositional changes in the node sets, including characteristics of the actors, but also modeling different types of events. More substantively, we show that by using a relational event model we are able to provide insights into a complex form of social organization of problem solving that cannot be fully reduced to individual self-interest.

We organize the article as follows. In the next section we provide a rationale for the development of models for sequences of relational events and describe our modeling approach. We then illustrate the application of the model through an empirical example in the context of an ongoing F/OSS project. In doing so, we specify statistics that permit the analysis of relational events linking different sets of actors. We offer interpretations of the results in our empirical context to illustrate how sequences of interdependent problem-solving actions can lead to specific self-organizing patterns that provide the social infrastructure underpinning the production of F/OSS. We conclude by discussing the general applicability of relational event models in organizational research.

Background and Motivation

Three statistical frameworks are available for the modeling of relational dependencies in network data: ERGM, SAOM, and REM. We provide a high-level comparison of these frameworks in Table 1,

Table 1. Comparison of the Exponential Random Graph Model (ERGM), Stochastic Actor-Oriented Model (SAOM), and Relational Event Framework (REM).

	ERGM	SAOM	REM (This Article)
Unit of analysis	One static network	Longitudinal networks (i.e., a network captured over several time points)	Continuously occurring relational events
Data structure	One binary matrix for each network	At least two binary matrices for each network	A sequence of events
Extensions	Longitudinal networks (several time points) Multiple networks (two networks) Two-mode networks Multilevel networks	Multiple Networks Two-mode networks Social settings	This article: Two-mode relational events
Key assumptions	Dependence structure between actors Global structure emerges out of local patterns	Dependence structure between actors Actor-oriented model: actors send ties to alters to maximize utility functions	Conditional independence of current events on their past history
Estimation	Markov chain Monte Carlo maximum likelihood estimation (MCMCMLE)	Simulation: Markov chain Monte Carlo (MCMC) Estimation: Logistic random utility maximization models (based either on MoM or MLE)	Conditional multinomial logistic regression
Key sources of information	Lusher, Koskinen, and Robins (2012)	Snijders, van de Bunt, and Steglich (2010); van de Bunt and Groenewegen (2007)	Butts (2008)

where we highlight the type of data structures that each framework is most suitable for, as well as the main extensions of these frameworks. In brief, the ERGM framework (Lusher et al., 2012) is best suited for cross sectional observations of social networks, usually taken at one point in time. ERGMs require binary ties of one type per network, which can be directed or undirected. Extensions of the ERGM framework have been proposed for multiple networks, two-mode networks, multilevel networks, and longitudinal networks (see Lusher et al., 2012). Important methodological advances have recently been made in modeling network evolution with the aim to understand of how social and organizational structures emerge and change over time (G. Ahuja, Soda, & Zaheer, 2011; Lazega, Mounier, Snijders, & Tubaro, 2012; Rivera, Soderstrom, & Uzzi, 2010; van de Bunt & Groenewegen, 2007). The SAOM framework (Snijders, van de Bunt, & Steglich, 2010; van de Bunt & Groenewegen, 2007) is best suited to the modeling of a network that evolves over time, captured using panel data at discrete points in time. SAOM has also been developed for binary ties, directed or undirected, of one type. The framework has been extended to cater for multiple networks and two-mode networks.

These statistical frameworks are appropriate for cross-sectional data—that is, data collected at one point in time—or for longitudinal data—that is, data collected through repeated waves of sociometric questionnaires administered at a limited number of discrete time points. This implies that continuous sequences of relational events need to be aggregated and dichotomized to be analyzed using existing statistical tools. Indeed, when data are collected as relational event sequences, they are usually aggregated over time and transformed to produce one or several cross-sectional networks representing some persisting connections among actors (e.g., Conaldi et al., 2012; Grund, 2012;

Quintane & Kleinbaum, 2011). The resulting longitudinal or cross-sectional data structures are then typically analyzed using a SAOM framework, or an ERGM. For example, Grund (2012) collected all events that occurred during English premier division soccer games in the 2006-2007 and 2007-2008 seasons. In this data set a pass between two players can be considered as a relational event that joins these players and events are distributed over time during each match, in a sequence. However, the sequences of passes were aggregated into cross sectional networks prior to analysis. Another example is the article by Quintane and Kleinbaum (2011) where electronic communications between all employees of a small nonprofit organization were collected over several months. This sequence of communication events was then collapsed into one cross section and analyzed using ERGM.

The REM (Butts, 2008) provides a way to directly model sequences of relational events without aggregating them. The framework was initially developed for directed events that join actors within the same node set (i.e., the equivalent of one-mode networks; see Quintane et al., 2013, for an empirical example), and this article proposes an extension to events that join actors that pertain to different node sets (i.e., the equivalent of two-mode networks). In this section, we first present the rationale for not transforming sequences of relational events into binary cross-sectional panels and then review the existing work in modeling of relational events.

Transforming Relational Event Sequences

The aggregation of sequences of relational events into binary cross-sectional network panels has two major analytical disadvantages: the loss of information regarding the order and number of events that compose a tie, and the inability to account for changes in the composition of the actor sets. We discuss these issues in turn. Aggregating relational events into ties results in the loss of information on the ordered sequence contained in the event data that underlie the creation of the tie. In other words, aggregating relational event sequences implies that the way in which events are distributed over time is irrelevant. This is important because different sequences of relational events, reflecting potentially distinct social processes, can lead to the same configuration of network ties. For example, in the study of board interlocks, a director who has joined and left the board of several companies sequentially over a given period of observation would be treated similarly to another director who has been on the board of the same companies during the full period of observation. Clearly the substantive implications of the rotation of the first director versus the stability of the second director for the companies are different. Similarly, the order of relational events is critical when events are considered as conduits that enable the flow of—for example—information, ideas, or goods. Again, in studies of board interlocks, the simultaneous presence of the same director on the board of two companies is usually assumed to enable the transfer of information and knowledge between the companies. Yet, if the presence of such director on the two boards is sequential, that is, the director left the board of the first company before joining the board of the second company, then the information or knowledge flow can account only for the transfer of information from the first to the second company—and not vice versa. More generally, preserving information on the order of events is key to providing causal explanations of the evolution of network structure.

In addition, aggregation of relational events over time leads to a frequency count where the number of times that a specific event occurs becomes an attribute (the ''strength'') of the tie between two actors (Borgatti & Halgin, 2011). Two analytical strategies may then be used to investigate aggregated relational event data: dichotomization or the use of statistical models that incorporate the weight of the tie. Dichotomization may be misleading because it enforces the choice of an arbitrary threshold that defines the existence of a tie. This obscures variations in the frequency of interactions, thus preventing considerations about changes in the level of attention that an actor dedicates to a given partner or issue. For example, in our empirical context we could stipulate a threshold of one action taken by a contributor on a software bug to define the existence of a tie. This implies that a

contributor undertaking one action and another contributor undertaking 10 actions on the same software bug would be undistinguishable in the aggregate data set. Dichotomization is also problematic because a change of threshold can alter greatly the structural properties of the network being studied (Grannis, 2010), both directly because it removes certain ties and indirectly because it affects network density (Anderson, Butts, & Carley, 1999).

Much progress has been made in the analysis of valued network (e.g., Opsahl, Agneessens, & Skvoretz, 2010), yet the majority of analytical procedures in standard social network software packages, such as UCINET (Borgatti, Everett, & Freeman, 2002), are tailored to binary network ties. Furthermore, statistical models for social network analysis such as ERGM or SAOM have been originally designed to consider ties as binary state variables. The analysis of valued ties remains both problematic and greatly needed within these major statistical frameworks.

A second reason suggests caution about research designs that aggregate streams of relational events into "states" that are then interpreted—and analyzed—as network ties. Aggregating sequences of relational events over time assumes that the actors involved in these events were present during the whole aggregation period. While this assumption of stability of the set of actors can be tenable in specific contexts, many current organizational environments are dynamic, with social actors entering and leaving a project or an organization at different points in time—a recurrent feature of organizations that Cohen, March, and Olsen (1972) described as "fluid participation." This issue is particularly relevant because assuming a fixed set of actors implies that actors have homogenous exposure to the risk of being a source or a target of an event over the aggregated period. An example from our empirical context can illustrate why this is important. F/OSS projects are characterized by high turnover of contributors and software bugs. This means that software bugs (i.e., the problems contributors need to solve) are discovered and disappear over time as software progresses and contributors address problems that are currently active. Each active problem may be considered as a locus of attention for existing contributors, while a resolved problem is a potential source of attention that has disappeared. The appearance and disappearance of problems affect the potential attention that contributors can dedicate to all of them and hence the probability that they will produce an action directed to any of them. A misrepresentation of this flow of problems will overestimate the sources of attention that a contributor is exposed to at a given point in time. The same logic applies to the joining and leaving of contributors.

Furthermore, assuming a fixed set of actors potentially creates artificial temporal overlap between actors in a given setting. More specifically, a fixed set of actors assumes that two actors are present at the same time in the same setting when their presence in the setting might in fact be sequential. This can be particularly problematic in substantive investigation where the simultaneous presence of actors in a setting is required for a process to unfold. For example, in the context of F/OSS projects, peer recognition is critical to understand the contributions of developers to a given project (Stewart, 2005). Yet peer recognition specifically requires the referent community of peers to be active in the project at the same time as the focal actor.

Models for Relational Event Sequences

REMs have emerged as a way to avoid having to transform sequences of relational event into binary cross sections. Three recent contributions have proposed specialized models focused on the analysis of sequences of relational events (Butts, 2008; de Nooy, 2011; Stadtfeld & Geyer-Schulz, 2011). These models are specifically suited to estimate sequences of *relational events*, where a relational event is defined as "a discrete event generated by a social actor (the 'sender') and directed toward one or more targets (the 'receivers,' who may or may not be actors themselves)" (Butts, 2008, p. 159). REMs predict the occurrence of an event at a certain point in time or in a sequence.

Therefore, they do not require prior aggregation of the sequence of events and can handle changes in the node set for each event.

Butts's (2008) REM for social action is a flexible modeling framework for relational events in a social setting based on event history analysis. The framework assumes that each event is independent of all other events, conditional on the realized history of events. The conditional independence assumption implies that "past history creates the context for present (inter)action, forming differential propensities for relational events to occur" (Butts, 2008, p. 160). Then, once an event occurs, "this alters the context of action, and the process begins anew" (p. 160). The framework enables the estimation of the likelihood that a given event is associated with specific configurations of past relational events sequences, without assuming that the event is determined by these configurations. Further assumptions made within Butts's (2008) framework are the following: (a) the statistics that capture past sequences of events in the model completely represent the process at hand, (b) future events do not influence past event, and (c) nonevents do not influence events. Butts (2008) proposes that these assumptions are most likely to be satisfied in a context where past behavior strongly influences future behavior and where actors do not have the time or the disposition to engage in strategic behavior that would include adapting one's actions because of the nonactions or potential future actions of other actors.

Because of the novelty of the model, empirical applications are still limited. Butts (2008) provides an empirical illustration of the model using the sequence of radio communications between responders to the World Trade Center disaster. Brandes et al. (2009) propose an extension of Butts's (2008) model to weighted relational events, where the weight of the event represents the degree of friendliness (or hostility) of an interaction from one country to another. Brandes et al. (2009) show that a past negative action from one country to another tends to lead to another negative action from the recipient to the sender, while this is much less common for a positive action.

De Nooy (2011) proposes a multilevel event history model that predicts when a relational event will happen in response to a specific initial event. The model assumes that actors become at risk of sending relational events once an initial event has occurred, which marks the beginning of a period. While the event-history framework accounts for temporal dependencies between actions, the multilevel aspect of the model accounts for dependencies among observations created by repeated actions by the same actors. We highlight a distinction between this approach and Butts's (2008) framework. For Butts, each prior relational event creates the condition for the next event to emerge (i.e., each event marks the end of the previous period and the beginning of the new period), while for de Nooy (2011) relational events (book reviews) arise in reaction to a specific event of a different kind (book publication) that occurred in the past. De Nooy's (2011) framework is best suited for sequences of events of one type that follow the occurrence of an event of a distinct type (e.g., book reviews follow book publication). It is not well suited for sequences of relational events of the same type, which we argue are common in management research

Stadtfeld and Geyer-Schulz (2011) propose an actor-oriented Markov process framework to analyze sequences of relational events. In the general version of their framework, actors are assumed to make two decisions: (a) when to generate a relational event and (b) to whom this relational event is directed. Because the model is based on a Markov process, all information required to model each event is assumed to be contained in the previous state of the network. In other words, each dyadic event is modeled as the result of actor's decisions that are based on the realized state of a Markov process. Hence, each state is the result of aggregation of past events that constitute the network structure on which probabilities of present events are inferred, with the implication that the information pertaining to the order of the sequence of relational events that lead to a particular configuration is not retained. However, Stadtfeld and Geyer-Schulz suggest that their framework could incorporate this information by extending the state space (i.e., including information about prior states). Furthermore, Stadtfeld and Geyer-Schulz's framework is actor oriented, in the sense that events are

products of decisions based on individual utility functions, very much like the SAOM framework. By contrast, Butts's (2008) framework can be considered as action or behavior oriented (p. 167) and does not require the assumption that events are the product of actor's decision to maximize a utility function. An implication is that in Stadtfeld and Geyer-Schulz's framework actors are assumed to make the decision of whom to send an event to prior to and independently from when to send this event. This may be problematic because it decouples social from temporal interdependencies, which are likely to be related in a variety of contexts. For example, Butts shows that radio operators exhibit a tendency to preferentially communicate with other radio operators who have most recently contacted them.

Model Development

We propose an extension of Butts (2008) framework to events linking actors, or social entities, in different node sets (i.e., the equivalent of two-mode networks). The model presented here is based on the sequential form of Butts's REM, which we propose can be estimated using a conditional logistic regression. Butts presents both a continuous and a sequential form of the model. The continuous form of the model assumes that each relational event has a piecewise constant hazard of occurrence (λ) given a particular history of prior relational events. The piecewise constant rate function λ is parameterized in the form $\lambda = \exp\left(\sum_h \theta_h (u_h)_{i(m)j(m)}\right)$. Each statistic $(u_h)_{i(m)j(m)}$ depends on actor attributes, past configurations of relational events relevant to the m_{th} event, $a_m = (i(m), j(m))$ and/or exogenous covariates; and the θ_h are corresponding parameters.

In our empirical context, the information about the order of events is more important than their exact timing. In this case, Butts (2008) demonstrates that the probability that the m_{th} event is the next event is equal to the occurrence rate for that event divided by the sum of the rates for events $(k, l) \in \Omega(m)$ that might occur, including the m_{th} event itself. The next event in a sequence has a probability: $p(m) = \lambda_m / \left(\sum_{m'} \lambda_{m'}\right)$. λ_{m} is the rate parameter of event m and the events m′ are all potential events that could occur instead of m in the sequence. The probability for an event to be next in a sequence is expressed by the following function,

$$\exp\left(\sum_h \theta_h (u_h)_{i(m)j(m)}\right) \Big/ \sum_{(k,l)\in\Omega(m)} exp\left(\sum_h \theta_h (u_h)_{kl}\right) \tag{1}$$

The probability for the full sequence of events is given by the function,

$$p(A_t) = \prod_m \left[exp\left(\sum_h \theta_h (u_h)_{i(m)j(m)}\right) \Big/ \sum_{(k,l)\in\Omega(m)} exp\left(\sum_h \theta_h (u_h)_{kl}\right) \right] \tag{2}$$

where m is an index that goes through all the events, h is an index that goes through all the statistics, k is an index that goes through all the software bugs, and l is an index that goes through all the contributors.

The probability that an event is next in a sequence of events is calculated by comparing its occurrence rate in the past (coming from the statistics in the model) to the occurrence rate of all the potential events (Equation 1). The probability of the sequence itself is the product of the probabilities of each event (Equation 2). A large positive (negative) value of θ_h indicates that an event m is more (less) likely if the statistic u_h summarizing relevant prior events is high. The model is equivalent to a conditional logistic regression (e.g., Agresti, 2002), where each event is specified as a stratum that

can—for example—be fitted using the COXREG procedure in SPSS (SPSS 20 was used in our study). In the application of the model presented here we distinguish between two different types of events (problem-solving action) observed in the sequence on the basis of the underlying level of effort needed to produce them.

Empirical Illustration

Setting

We illustrate the application of this model empirically in the context of an F/OSS project. F/OSS projects are virtual communities of contributors who participate to the production of software that are freely redistributable and modifiable (O'Mahony, 2003).[1] The specific F/OSS project we analyze is "Epiphany," now rebranded "Web," the default web browser of the GNOME graphical desktop environment. The Epiphany project has been active since November 2002, and to this date has undergone 18 release cycles, each resulting in a stable new version of the software.

This empirical setting is ideal for illustrating our modeling approach because all actions taken by contributors on software bugs are recorded, time-stamped, and publicly available.[2] Furthermore, the voluntary nature of F/OSS projects implies that contributors join and leave the project over time, and the activity of the contributors means that software bugs are "created" and "fixed" continuously as well. As such, the composition of the sets of contributors and software bugs change continuously throughout the duration of the project. Also, we have information about each contributor, software bug, and the type of action that links them. Finally, the sociotemporal interdependencies that underlie the sequences of actions performed by contributors on software bugs provide critical information to explore the internal organizational dynamics of F/OSS projects. In one of the most important contribution to research on F/OSS production, Lakhani and von Hippel (2003) conclude that it is necessary "to analyze the micro-level functioning of successful open source projects to really understand how and why they work" (p. 940).

Following this broad research direction in the specification and interpretation of our model, we attempt to investigate whether micro-level sequences of (inter)action can give rise to self-organizing patterns that provide the social infrastructure underpinning F/OSS production. Existing literature on F/OSS has endeavored to provide answers to the "paradox of participation." The survival of F/OSS projects depends crucially on the commitment of voluntary participants and their willingness to engage in decentralized activities (Crowston & Scozzi, 2008; Mockus, Fielding, & Herbsleb, 2002), with no obvious incentive to contribute to the production of what effectively constitutes a shared public good (Lee & Cole, 2003; Markus, 2007; Murray & O'Mahony, 2007; O'Mahony, 2003; von Hippel & von Krogh, 2003). The prevalent explanation holds that visibility and peer recognition act as the main motivating factors in the absence of formal incentive systems. According to this view, developers contribute publicly to the addressing of problems whose solution grants visibility and recognition—that is, signaling incentives either in the form of peer recognition or future job offers (Lerner & Tirole, 2002). Yet this explanation does not necessarily account for the engagement of contributors in more mundane, maintenance-based tasks, which cannot be directly associated with visibility gains, but constitute the bulk of contributions made by participants to F/OSS projects (Dempsey, Weiss, Jones, & Greenberg, 1999; Lakhani & von Hippel, 2003) and have been shown to be essential to the survival of F/OSS projects (David & Rullani, 2008; von Krogh, Spaeth, & Lakhani, 2003).

We adopt two empirical strategies to demonstrate how a relational event model can be used to address these substantive questions. First, we use attributes of the contributors and software bugs to investigate whether contributors tend to prioritize actions that could potentially give them more visibility in their community. We propose two potential indicators of visibility. The *popularity* of a

software bug, that is, the extent to which a software bug has attracted actions from contributors in the past, acts as an informal signal of the interest that the community of contributors attaches to the resolution of a given software bug. Contributors may therefore be driven to contribute to popular software bugs because of their popularity, as a self-reinforcing mechanism. The *severity* of a software bug captures the level of priority that contributors need to allocate to a software bug. It acts as a more formal signal of the interest that the community of contributors attaches to the resolution of a given software bug. Again, contributors may be driven to direct a disproportionate amount of their actions to more severe software bugs to demonstrate their skills and gain visibility.

Second, we exemplify the possibility offered by the model to cater for different types of events by distinguishing between actions depending on the amount of energy—or "effort"—that contributors need to devote when engaging a software bug. We consider high-effort actions those involving the direct production or review of software code. We chose to focus on code contribution as high-effort actions because code contribution is as highly valued in F/OSS projects as it is rare. This is testified by famous representative of the F/OSS ecosystem—see, for example, Raymond's (1999) most famous illustration of the shared values among F/OSS contributors in the case of Fetchmail project or the well-known quote from Linus Torvalds, the initiator of Linux, "Talk is cheap. Show me the code." By contrast, low-effort actions are those addressed at more mundane tasks contributing to the description, general classification, and maintenance of software bugs (Lakhani & von Hippel, 2003).[3] Here, we are interested in understanding (a) the potential differences in self-organizing processes that emerge in high-effort and low-effort actions and (b) whether the signals of visibility (popularity and severity) operate in the same way to sustain the engagement of contributors in high-effort and low-effort actions. More generally, distinguishing between these different types of actions enables us to explore specifically the emergence of self-organizing processes in the more mundane maintenance actions that compose a large part of the actions taken on software bugs.

Our aim is to use these substantive questions to illustrate the specification and interpretation of the results of a relational event model.

Data Collection and Preparation

We collected all the 4,347 actions undertaken by any active contributor on any active software bug throughout two release cycles of Epiphany,[4] which lasted for approximately one year, from September 8, 2005, to September 6, 2006. All actions carried detailed temporal information that allowed us to reconstruct the action sequence.[5] The two releases had an average of 119 active contributors (103 for the first one and 135 for the second one) against an average of 117 ($SD = 45$) for the whole project. The two releases had a lower number of software bugs (686 on average) than the whole project (1,305 on average), yet the release cycles are not unusual given the strong variations both in terms of active contributors and active software bugs throughout the life of the project. For robustness, we checked our results using a release cycle that had a higher number of active software bugs than the average for the whole project and the results are substantively similar to those presented here.

As contributors joined and left the project and new software bugs were reported during the release cycles we had to adjust continuously the sets of active software bugs and contributors, that is, those who were at risk of emitting or receiving an action. We considered contributors and software bugs as active when the first action in which they were involved was recorded. Similarly, we considered contributors and software bugs as inactive after the last action involving them was recorded during the observation period, if no other actions involving them had been reported in any of the release cycles following the ones under investigation. Some contributors (57) and software bugs (309) were already active before the beginning of the first cycle (i.e., they had either acted on at least a software bug or received an action from at least one contributor before the beginning of the first release cycle). Active contributors and software bugs constitute the risk set used in the model estimation.

Our data set is structured as a discrete ordered sequence of relational events (actions from a contributor on a software bug), each of which can be expressed as $a_{\mathrm{m}} = (i(m),j(m))$ where the m_{th} event is an action from contributor $i(m)$ to bug $j(m)$. The contributors $i(m)$ are members of a known set $N(m)$ and software bugs $j(m)$ are members of a known set $P(m)$ relevant at each event (m). The sequence of events is assumed to be of length M and may be written in the form $(a_1, a_2, \ldots, a_M)$. The sequence of events up to and including the m_{th} event may be written as $A_m = \{a_h: h \leq m\}$. $a_m = (i(m),j(m))$ denotes the m_{th} event and $\Omega(m) = \{(k, l): k \neq l \text{ and } k \in N(m), l \in P(m)\}$ is the set of potential events for the m_{th} event. $\Omega(m)$ includes the actual m_{th} event and all possible events that could have occurred as the m_{th} event. More specifically, for each event we established a risk set composed of active contributors and active software bugs at the time of the event. We computed all possible dyadic combinations of contributors and software bugs that may have occurred instead of the actual event, which we call potential events. We developed a Java application to compute potential events and their relevant preconfigurations based on the actual events contained in the data extracted from the bug repository.[6] In the appendix, we present to the interested reader a high-level road map of how the data set has been prepared and transformed with the Java program for estimation using multinomial conditional logistic regression in a standard statistical package.

Dependent Variable

Within a sequence of events, the dependent variable is the next event (action) in the sequence. More specifically, for each action on a software bug, the dependent variable is a binary variable containing information about possible dyadic combinations of contributors and software bugs. It takes the value of 1 if this action has occurred in the dyad and the value 0 if it has not occurred.

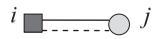
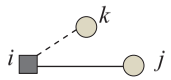
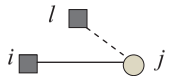
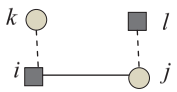
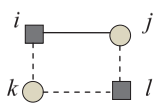
Explanatory Variables

The main task in specifying a relational event model is the identification of relevant statistics, which characterize the recurrence of certain configurations in the past sequence of relational events. Note here that the statistics proposed by Butts (2008) apply in the case where both the sender and the recipient of an action belong to the same set of actors, even though the framework is not restricted to this case. In our empirical setting, actions can be generated only by a contributor and directed toward a software bug. A software bug cannot act on a contributor or on another software bug and a contributor cannot act on another contributor. In a cross-sectional perspective, this type of data structure is named a two-mode network and is characterized by the existence of two sets of actors with ties between the sets but not within the sets. We draw from ERGM (Lusher et al., 2012) and SAOM frameworks (Snijders et al., 2010) to propose a complete list of statistics that reflect sequences of events between different sets of actors. In the following paragraphs, we describe the statistics and provide information about their scaling (see Table 2).

Given a set of actors A and a set of recipients B, *prior edge* captures the extent to which past actions from a specific actor i from set A toward a specific recipient j from Set B predict future actions from i toward j. This statistic is equivalent to the persistence statistic proposed by Butts (2008, p. 169) and captures the tendency for inertia in social relations (Rivera et al., 2010). Specifically, in our context a positive and significant parameter estimate indicates that the same contributors tend to act repetitively on the same software bugs, while a negative parameter estimate indicates that a contributor tends not to repeat actions on the same software bug over time.

Prior activity provides information about the impact of an actor's past activity on the probability of that same actor to send another action in the future. It is similar to the out-activity statistic in the SAOM framework that captures a self-reinforcing effect of past activity (Snijders et al., 2010). Substantively, a positive and significant parameter estimate should be interpreted as meaning that the

Table 2. Statistics and Scaling.

Name	Visual Representation	Probabilistic Mechanism	Statistic	Scaling
Prior edge		The tendency for contributor *i* to engage software bug *j* as function of the extent to which *i* has engaged software bug *j* in the past	$\sum_{r=2}^{t-1} Y_{ijr}$	$\sum_{k=1}^{m}\sum_{r=2}^{t-1} Y_{ikr}$
Prior activity		The tendency for contributor *i* to engage software bug *j* as function of the extent to which *i* has engaged software bug(s) *k* in the past	$\sum_{k=1}^{m}\sum_{r=2}^{t-1} Y_{ikr}$	$\sum_{l=1}^{n}\left(\sum_{k=1}^{m}\left(\sum_{r=2}^{t-1} Y_{lkr}\right)\right)$
Prior popularity		The tendency for contributor *i* to engage software bug *j* as function of the extent to which j has attracted contributor(s) *l* in the past	$\sum_{l=1}^{n}\sum_{r=2}^{t-1} Y_{ljr}$	$\sum_{l=1}^{n}\left(\sum_{k=1}^{m}\left(\sum_{r=2}^{t-1} Y_{lkr}\right)\right)$
Out-in assortativity		The tendency for contributor *i* to engage software bug *j* as function of the extent to which *i* has engaged software bug(s) *k* in the past and *j* has attracted contributor(s) *l* in the past	$\sum_{k=1}^{m}\sum_{r=2}^{t-1} Y_{ikr} \times \sum_{l=1}^{n}\sum_{r=2}^{t-1} Y_{ljr}$	$\sum_{l=1}^{n}\left(\sum_{k=1}^{m}\left(\sum_{r=2}^{t-1} Y_{lkr}\right)\right)$
Four cycle		The tendency for contributor *i* to engage software bug *j* as function of the extent to which contributors *i* and *l* have engaged in software bug *k* and contributor *l* has engaged in *j*	$\sum_{l=1}^{n}\left(\sum_{k=1}^{m}\left(\sum_{r=2}^{t-1}\left(Y_{ikr} \times Y_{lkr} \times Y_{ljr}\right)\right)\right)$	$\sum_{k=1}^{m}\sum_{r=2}^{t-1} Y_{ikr} \times \sum_{l=1}^{n}\sum_{r=2}^{t-1} Y_{ljr}$
Core developers		The tendency for core developers to engage software bugs	$Y_{ijr} = 1$ *if i is core* $Y_{ijr} = 0$ *if i is not core*	
Severe software bug		The tendency for contributors to engage more severe software bugs	$Y_{ijr} = severity\ j$	
Core severe		The tendency for core developers to engage in more severe software bugs	$Y_{ijr} = severity\ j$ if *i iscore* $Y_{ijr} = 0$ *if i is not core*	

Note: ■ contributor ● software bug ⟶ current event ⋯⋯▸ past events ■ core developer. Let $Y_{ijt} = 1$ if *i* engages *j* at time *t*; $Y_{ijt} = 0$ otherwise. *i* denotes the contributor, *j* the software bug, *t* is the time of the event, *r* is the period of activity, *n* is the number of contributors active at time *t*, *m* is the number of software bugs active at time *t*.

more a contributor has acted on (any) software bugs in the past, the more likely he or she is to send another action in the future (which would indicate a general tendency toward active contributors becoming more active over time). A negative and significant parameter estimate indicates that the more a contributor has acted on any software bug in the past, the less likely he or she is to send actions again in the future (a reduction of activity).

Prior popularity refers to the extent to which past actions received by recipient *j* affect the probability that recipient *j* will receive a further action. This statistic is similar to the in-star statistic in the ERGM framework and equivalent to the preferential attachment statistic proposed by Butts (2008, p. 170). Its interpretation parallels the prior activity statistics in that a positive and significant parameter estimate indicates a higher propensity for software bugs that have received many actions in the past to receive more in the future, while a significant negative parameter estimate would indicate that a software bug that has received many actions in the past is less likely to receive one again in the future.

In specifying the popularity and activity effects, we assume that prior activity and prior popularity affect future activity or popularity in a linear fashion.[7] However, past research suggests that it may be more likely that the effect of prior activity (popularity) on future activity (popularity) is nonlinear (Ducheneaut, 2005; Fang & Neufeld, 2009; Qureshi & Fang, 2011). Indeed, a contributor who is already contributing to a high level may not be able to further increase his or her level of contribution and is more likely to either maintain or decrease it. This would occur because contributors may reach a saturation level given limited time and/or cognitive abilities. We expect a similar nonlinear relationship for the popularity of software bugs. It is conceivable that software bugs can receive only so many actions at the same time because of a limited number of different actions can be taken, but also because some actions require time to be completed and induce a pacing of activities. Hence, software bugs that receive many actions in the past are also likely to reach a saturation level and see their popularity decrease in the future. Consequently, we specify a squared term for prior activity and for prior popularity.

Out-in assortativity refers to the extent to which past actions sent by sender *i* and past actions received by recipient *j* affect the probability that sender *i* will emit a further action toward recipient *j*. A positive and significant parameter estimate indicates a higher propensity for a contributor who has sent many actions in the past to send a further action to a software bug that have received many actions in the past, while a significant negative parameter estimate would indicate that a contributor who has sent many actions in the past is less likely to send a further action to a software bug that has received many actions in the past.

Four cycle characterizes the extent to which prior actions of two actors *i* and *l* (from set A) toward recipient *k* (from set B) combined with actor *l*'s actions toward recipient *j* (from set B) will lead actor *i* to send an action toward recipient *j* in the future. It is similar in spirit to the four-cycle statistic presented in the ERGM framework (Wang, Pattison, & Robins, 2013). For example, in our context four cycle captures the extent to which the joint work of two contributors on software bugs in the past increases their propensity to work jointly on another software bug in the future, which can be conceived as a form of collaboration. A positive and significant parameter estimate indicates that past partners tend to partner again in the future, while a negative and significant parameter estimate can be interpreted as a tendency for contributors who have collaborated in the past not to collaborate again in the future.

Scaling

We follow Butts (2008) in choosing the scaling values for each statistic (see Table 2). Scaling is necessary because a raw frequency count would give more weight to events that happen later in the release cycle. All statistics (including high effort, low effort, and generic) are placed in the same metric using the scaling procedures described below to make their effect sizes comparable. All

scaled statistics have also been standardized, across all values for observed and potential events (using the descriptives command in SPSS), to make their effects more comparable across different model specifications.

The prior edge statistic is scaled using the total number of actions emitted by the contributor until the current action. Hence, the resulting scaled statistic should be interpreted as a proportion of the actions dedicated by contributor *i* to software bug *j* out of all the actions that contributor *i* has performed in the past. A positive parameter estimate means that the higher the proportion of actions contributor *i* has performed on software bug *j* in the past, the more likely is contributor *i* to perform one more action on software bug *j*. By contrast, a negative effect would indicate that the higher the proportion of actions that contributor *i* has performed on software bug *j* in the past, the less likely is contributor *i* to perform one more action on software bug *j*.

Both prior activity, prior popularity and out-in assortativity statistics are scaled using the total number of actions that have occurred until the current action. Hence, a positive prior activity (prior popularity) parameter estimate indicates that the higher the proportion of actions taken by contributor *i* out of all actions taken by all contributors in the past (actions received by software bug *j* out of all actions received by all software bugs in the past) the higher the probability of a further action from contributor *i* (to software bug *j*).

The four-cycle statistic is scaled by the prior activity of the contributor and the prior popularity of the software bug involved in the action. Hence, it is a proportion of the actions taken by contributor *i* on software bug *j* out of all the actions taken by contributor *i* and all the actions received by software bug *j*. The interpretation of the statistic is similar to those presented above. A positive parameter estimate indicates that the higher the proportion of actions taken by contributor *i* on a set of software bugs *k* out of all the actions taken by *i,* and the more a set of contributors *l* have also taken actions on software bugs *k*, and the more contributors *l* have taken actions on software bug *j*, out of all the actions received by *j*, the higher the probability that contributor *i* will take an action on software bug *j*. A negative effect will indicate a reduction of this probability.

Actor Attributes

We created three variables linked to the attributes of the contributors and of the software bugs: *core developer*, *severe software bug*, and *core severe*.

Core developer is included to control for the well-documented tendency of "core developers" to display higher levels of engagement (Crowston & Howison, 2005; Xu, Gao, Christley, & Madey, 2005). Core developers are contributors "who contribute most of the code and oversee the design and evolution of the project" (Crowston & Howison, 2005, p. 7), including the initiator and maintainer of the project. In the release cycle under study, four developers were publicly identified by the community of contributors in the Epiphany's official mailing list as core developers.[8] The binary variable takes the value of 1 if the contributor taking the action is a core developer and 0 otherwise. A positive parameter estimate denotes that being a core developer increases the probability of taking an action on a software bug, while a negative parameter estimate indicates a reduced probability that this action will occur.

Severe software bug is included to control for the propensity for contributors to engage software bugs that are considered as more urgent or important by the community and therefore may provide a higher level of visibility. Software bugs are assigned a severity level on a 7-point scale, from *trivial* to *blocker*, when they are opened. A positive parameter estimate indicates that severe software bugs have a higher probability to receive actions, while a negative parameter estimate indicates that severe software bugs have a reduced probability to receive actions. We also conducted robustness checks using a binary variable taking the value of 1 if the software bug has been classified as either

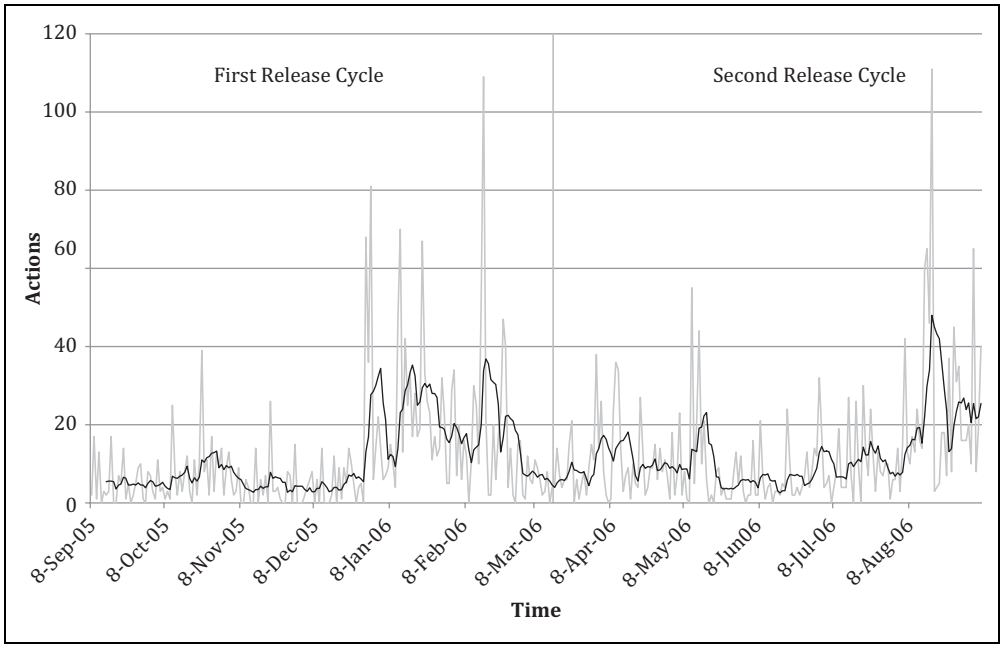


Figure 1. Daily time series of actions undertaken by contributors on software bugs (gray line) with weekly moving average (black line).

6 (*critical*) or 7 (*blocker*) and 0 otherwise. The results using a binary version of the variable are substantively similar to those presented here.

Finally, *core severe* reflects the extent to which core developers tend to target more severe software bugs. A positive parameter estimate indicates that core developers have a higher probability to act on more severe software bugs, while a negative parameter estimate indicates that core developers have a reduced probability to act on severe software bugs.

Analyses

A total of 4,348 actions were undertaken by a total of 194 active contributors on a total of 1,208 active software bugs over the duration of the two release cycles. Figure 1 shows the time series of problem-solving actions undertaken per day during the release cycle. On average, approximately 12 actions per day were recorded ($M = 11.94$, $SD = 15.01$) over the 364 days in which at least one action occurred. A maximum of 111 actions were recorded on a single day (August 17, 2006). Out of the total 4,348 recorded actions, we classified 164 as high-effort actions.

On average, approximately 76 contributors ($SD = 11.45$) and 333 software bugs ($SD = 29.12$) were active daily. Figure 2 shows the evolution of the number of software bugs and contributors over time. Over the duration of the project, 137 new contributors became active (or 70% of the total number of contributors) and 103 existing contributors (or 53% of the total number of contributors) became inactive. A total of 81 contributors (or 41.7% of the total number of contributors) joined and then left the project within the period of observation. Hence the change in composition of contributors corresponds to approximately 82% of the total number of contributors. Similarly, 899 software bugs (or 74.4% of the total number of software bugs) were opened during the release cycle and 780 (or 64.6% of the total number of software bugs) were solved during the release cycle, with 612 software bugs (or 50% of the total number of software bugs) being both opened and closed during the

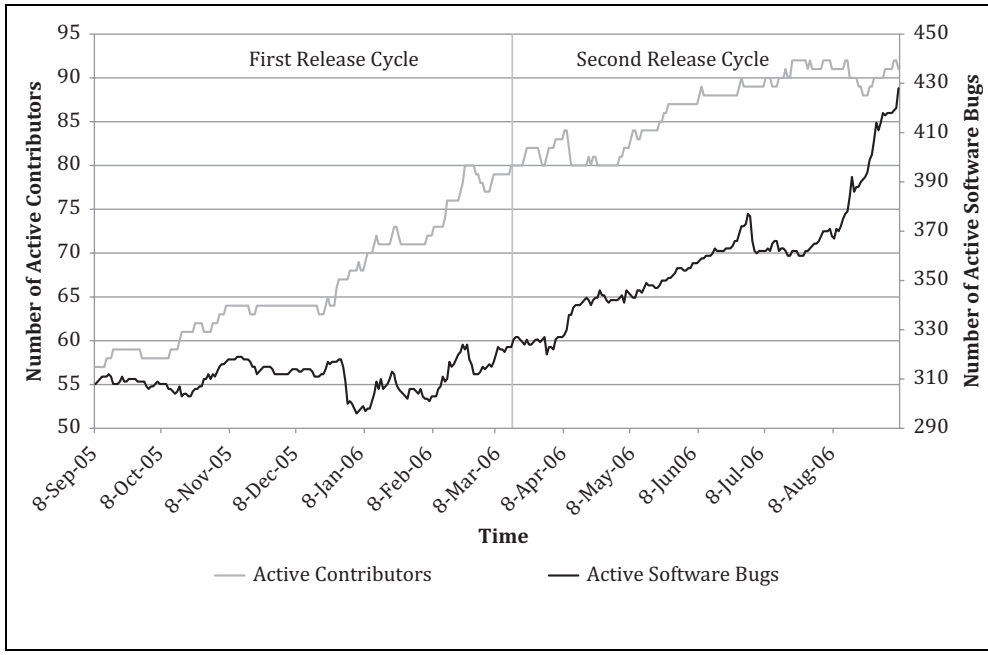


Figure 2. Active contributors and software bugs daily time series.

period of observation. Hence, approximately 88% of the total number of software bugs are either opened or closed (or both) during the period of observation. Actor membership in this project is clearly fluid. Aggregation of these data in one cross section would assume that all software bugs and contributors are continuously present throughout the observation period, which does not appear to be reasonable.

Core developers were responsible for 63% of recorded actions, showing that the distribution of contributions to the project is skewed. Core developers also contributed about 73% of all high-effort actions. There are 320 software bugs with an above average level of severity, accounting for 26% of the software bug population. Interestingly, more severe software bugs tend to be closed at a higher rate (76% are closed during the observation period) than normal software bugs (61% are closed during the observation period). Furthermore, more severe software bugs represent only 9% of the average number of active bugs on any given day (31 severe software bugs out of 333 software bugs), while they represent 26.5% of the total aggregate (320 out of 1,208). This indicates that there is a higher rotation of severe software bugs than normal software bugs during the release cycle, hence reinforcing the need to take carefully into account compositional changes in the set of active bugs. Core developers focus only 21.6% of their actions on more severe software bugs.

In the sections that follow, we begin by describing the parameter estimates for each of the models, noting the distinction between statistics calculated on high-effort or low-effort actions. Then, we provide qualitative interpretations of these results, to illustrate how relational event models can be used to address substantive questions. In this case, we are interested in understanding sustained engagement of developers in F/OSS projects.

Model 1 (see Table 3) predicts the existence of one action from a contributor on a software bug using the statistics identified in Table 2 computed on all actions that occurred prior to the action being predicted. This first model does not differentiate between high-effort and low-effort actions. In the next three models (see Table 4), we predict the existence of one action (without distinguishing

Table 3. Parameter Estimates of REMs Considering All Actions (Standard Errors in Parentheses).

	Model 1
High and low effort	
Prior edge	0.124** (0.002)
Prior activity	0.693** (0.026)
Prior popularity	0.031* (0.013)
Out-in assortativity	0.007** (0.002)
Four cycle	0.345** (0.009)
Core developer	0.381** (0.022)
Severe software bug	0.657** (0.028)
Core severe	–0.094** (0.014)
Prior activity squared	–0.294** (0.018)
Prior popularity squared	–0.036** (0.013)
Goodness of fit	
Null deviance	88,301
Residual deviance	67,870
Events (potential)	4,347 (114,968,285)

*$p < .05$. **$p < .01$.

between high effort and low effort) from a contributor to a software bug, but distinguish between high-effort and low-effort actions in the calculation of the statistics. In Models 2, 3, and 4 the high-effort statistics are computed using only past actions categorized as high effort and the low-effort statistics using only past actions characterized as low effort. Goodness of fit is reported using deviance measure of fit for each of the models (see Butts, 2008).

Results

In Model 1, the positive and significant effect of prior edge indicates that the more a contributor focused his or her actions on a software bug in the past, the higher the probability that this contributor will act again on the same software bug. The significant and positive effect of prior activity indicates that active contributors become more active over time, a self-reinforcing effect of activity. Coupled with the negative and significant effect of prior activity squared, we have evidence of a curvilinear (inverted U shape) relationship of prior activity with the propensity to contribute a further action. Hence contributors who reach a high level of activity tend to see their propensity to contribute decrease afterward. The effect of activity is above and beyond the effect of core developer, and suggests that some contributors who are not core developers account for a large share of actions taken on any software bug. Similarly, the significant and positive effect of prior popularity indicates that popular software bugs are more likely to receive another action, even though the size of the effect is much reduced compared to the effect of prior activity. Similarly to prior activity, the parameter estimate of prior popularity squared is negative and significant, indicating a curvilinear (inverted U shape) relationship. The significant and positive effect of out-in assortativity shows that active contributors have a tendency to target popular software bugs. This tendency, even if small, is above and beyond the effects of prior activity and prior popularity. The positive and significant effect of four cycle shows that contributors have an inclination to work on software bugs on which prior collaborators are also working. Results also indicate that core developers have a higher probability to work on software bugs than noncore developers (significant and positive core developer), as shown in the descriptive statistics, and that the more severe a software bug the more likely it is to attract actions (significant and positive severe software bug). Finally, the negative and

Table 4. Parameter Estimates of REMs With Predictors Segregated by Action Type (Standard Errors in Parentheses).

	Model 2	Model 3	Model 4
High effort			
Prior edge	0.036** (0.003)		0.013** (0.002)
Prior activity	0.896** (0.034)		0.822** (0.041)
Prior popularity	–0.084** (0.024)		–0.017 (0.022)
Out-in assortativity	0.021** (0.003)		0.011** (0.002)
Four cycle	0.032** (0.004)		0.034** (0.004)
Core developer	0.114** (0.014)		0.073** (0.015)
Severe software bug	0.015 (0.060)		0.005 (0.061)
Core severe	0.012 (0.014)		0.017 (0.014)
Prior activity squared	–0.453** (0.033)		–0.795** (0.039)
Prior popularity squared	–0.011 (0.010)		–0.020 (0.008)
Low effort			
Prior edge		0.121** (0.002)	0.120** (0.002)
Prior activity		0.816** (0.027)	0.655** (0.024)
Prior popularity		0.007 (0.014)	0.015 (0.014)
Out-in assortativity		0.007** (0.001)	0.006** (0.001)
Four cycle		0.331** (0.009)	0.342** (0.009)
Core developer		0.278** (0.023)	0.387** (0.021)
Severe software bug		0.705** (0.029)	0.718** (0.029)
Core severe		–0.094** (0.014)	–0.090** (0.014)
Prior activity squared		–0.346** (0.018)	–0.262** (0.019)
Prior popularity squared		–0.022** (0.009)	–0.020* (0.008)
Goodness of fit			
Null deviance		88,301	
Residual deviance	76,978	68,141	67,252
Events (potential)	4,347 (114,968,285)		

*$p < .05$. **$p < .01$.

significant effect of core severe shows that core developers have a lower probability to take an action on more severe software bugs. Effect sizes can be interpreted similarly to a logistic regression as parameter estimates can be exponentiated to yield odds ratios. For example, a parameter estimate of .381 for core developer indicates that there is a 46%, $\exp(.381) = 1.46$, increase in the likelihood of sending an action, when a core developer is involved.

The following three models provide more nuanced information about the impact of past high- or low-effort actions on the probability that a future action will occur. In Model 2, statistics are computed using only actions that are classified as high effort, but nevertheless predict any action type. The results are similar to those presented in Model 1, with some notable differences. Prior popularity becomes negative and significant (and the squared term becomes nonsignificant), which means that software bugs that attracted more high-effort actions in the past are less likely to receive a further action. Severe software bug and core severe parameters estimates become nonsignificant.[9] In Model 3, statistics are computed using only actions that have been classified as low effort. The results are very similar to those obtained when considering all actions together. The only notable difference is the loss of significance of the prior popularity parameter estimate, even though the effect of prior popularity squared is still significant and negative indicating the same curvilinear relationship.

In Model 4, we model the probability of the existence of a new action based on two sets of statistics; one set is computed on previous high-effort actions and the other on previous low-effort actions. The effect of prior edge remains positive and significant for both high-effort and

low-effort actions. However, the size of the parameter estimate for high-effort actions is reduced compared to Model 2, which signals that contributors who act repetitively on a given software bug with low-effort actions also tend to act on the same software bug with high-effort actions. We find a similar reduction of effect size for out-in assortativity and for core developer in high-effort actions. Furthermore, the prior popularity parameter estimate loses its significance in high-effort actions. The effects of the other parameters in the model are identical in sign and significance to those presented in Models 2 and 3.

In comparing high-effort and low-effort actions, we want to understand whether past collaboration occurring in low-effort actions is a better predictor of a future action than past collaboration in high-effort actions, hence whether different propensities for self-organization exist in high-effort versus low-effort actions. We calculate a z test to assess whether the difference between high-effort and low-effort four cycle parameter estimates is significant.[10] The z score is above 2.33, hence the difference between the two is statistically significant. Because the parameter estimate of the four cycle in low-effort actions is larger than the parameter estimate of the four cycle in high-effort actions, we can conclude that past collaboration in low-effort actions is a stronger predictor of a future action than past collaboration in high-effort actions.

Qualitative Interpretation

The results presented above carry two key insights. The first is that the characteristics of software bugs that could potentially enhance the visibility of contributors have mixed impact on attracting actions from contributors. The popularity of software bugs leads to more actions (Model 1), but the effect is weak compared to the effect of the activity of contributors, and it becomes negative (Model 2) or nonsignificant (Models 3 and 4) when we distinguish between high- and low-effort actions.[11] Furthermore, while severe software bugs attract consistently more low-effort actions, they do not have a marked tendency to attract high-effort actions. By contrast, characteristics of the contributors are clear predictors of future actions, as seen through the strong and significant prior activity and core developer effects.

The second insight is that the micro-level patterns of problem solving actions by contributors suggest the emergence of self-organizing tendencies. An element of centralization is revealed by the existence of a central group of contributors in high- and low-effort actions (positive and significant core developer and activity parameter estimates). Having a formal role in the project (core developer) increases the propensity that a contributor will take action, yet noncore developers still perform substantial volume of actions. Furthermore, we witness elements of collaboration (significant and positive four cycle) in high-effort and especially in low-effort actions.

As a conclusion, these results point toward a potential contribution to the current debate regarding the sustained engagement of volunteer contributors to the production of software products. While the prevalent explanation focuses on visibility and recognition as motivating elements driving the continuous engagement of contributors in a project, our analysis provides an opportunity to develop a complementary explanation of the same phenomenon by looking at the micro-dynamics of realized engagement. The examination of micro-level functioning of the two release cycles we have observed provides at least some evidence that matches between problems and problem solvers are associated to a variety of endogenous processes, which we interpret as the elementary components of the organizational structure of the project. Our analysis suggests that the actions of the contributors are embedded within a form of social organization. We find elements of centralization and collaboration over time. Collaboration is especially visible in low-effort actions, for which the visibility seeking motivation is less likely to hold. This indicates that the continuous engagement of contributors over time, especially in mundane actions, can be conceived as a form of social interaction through which contributors signal a sustained affiliation to a project and a community.

Discussion and Conclusions

In this article, we presented a modeling technique, based on Butts's (2008) REM, which enables the statistical analysis of full sequences of relational events. We proposed a model, akin to a conditional logistic regression, which considers each event as a data point in a sequence of events and assumes that observed events are conditionally independent on prior events in the sequence. The REM framework improves on existing statistical frameworks for social network analysis such as SAOM and ERGM by removing the need to aggregate sequences of relational events over time. Furthermore, we extended Butts's REM, originally derived for the analysis of events linking individuals, to the analysis of events linking organizational participants to organizational tasks or issues and proposed a set of predictive statistics that may assist with the modeling of sequences of relational events linking different sets of actors.

We also illustrated the value of using an REM in organizational settings. To do so we provided an empirical example based on problem-solving actions taken by contributors on software bugs during two release cycles of an F/OSS project. We obtained information about the exact temporal order in which these actions took place as well as characteristics of the contributors and of the software bugs. We integrated in the model information about composition changes of both contributors and software bugs and distinguished between rarer high-effort actions and more mundane low-effort actions. By doing so, we showed that our model was able to disentangle competing structural tendencies that outlast the influence of the set of contributors participating in bug fixing at any point in time and that these can be specified for different types of actions performed by contributors on software bugs. We provided an interpretation of our results within the current debate on the organizational factors that may sustain the attention and engagement of voluntary contribution to the production of F/OSS software products. Notably, we provided some insights into the emergence of a complex form of social organization of problem solving that cannot be fully reduced to individual self-interest, especially for low-effort actions (maintenance tasks).

The model presented here can be generally adapted to any sequence of relational events or actions between two sets of actors (or senders and recipients) known at the time of each event. The minimal information required for the model to be estimated is a temporally ordered list of events with unique identifiers for the sender and the recipient of each event, together with information about the window of activity of each sender and recipient during the sequence. The optimal data set size is a function of the number of events given a certain number of actors. Adding events to the data set increases statistical power. The repetition of events among the same actors generates the relational structures that the REM framework allows to investigate. If many senders and recipients exist in the data set, but few events connect them, finding repetition of events among the same actors over time is generally unlikely (i.e., very sparse network structure). As a rule of thumb, our experience with the model suggests that it is usually more tractable to have a limited number of senders and recipients (in the order of between 20 and 200), and a larger number of events (from several hundreds to thousands).

Beyond this minimal information, the model can handle characteristics of the senders (such as core developer in our empirical example) and characteristics of the recipients (such as bug severity in our empirical example). The variables can be binary, categorical, and continuous with no specific implications for model estimation or convergence. The type of statistics that can be created using actor covariates parallel those existing in the ERGM and SAOM frameworks. The model can also accommodate time-varying covariates (i.e., attributes of the actors that change over time such as, for example, tenure in a project) and dyadic covariates (e.g., attributes that express similarities or differences between actors and that may be used to capture, for example, aspects of homophily). As our empirical example illustrates, information on the type of event can be included in the model in a flexible way (i.e., the distinction between high-effort and low-effort actions). Finally, continuous

time information can also be included in the model and used to characterize temporally the effect of certain statistics. For example, in our empirical context different time frames could be used to investigate whether contributors have a tendency to act on software bugs that other contributors have acted on recently, a form of short-term popularity, which may explain the reduced influence of popularity of software bugs in driving actions from contributors. More generally, including temporal information would allow discrimination of the effects of stable patterns of interactions that have developed over long time frames, and those that occur over short time frames and are driven by contingency factors.

The model presented here can be applied to a variety of empirical contexts. At a fundamental level, relational events underlie the creation, maintenance, and dissolution of network ties, and hence the aggregate dynamics of organizational networks. As such, investigating micro-level processes in sequences of relational events provide information about the evolution of network ties and of social structure. More specifically, organizational communications captured via phone calls, email communications, and instant messages provide rich contexts for studying the evolutionary dynamics of social networks (Aral & Van Alstyne, 2011) and social influence (Aral, Muchnik, & Sundararajan, 2009). In this context, the use of relational event models would provide much finer grained understanding of the role of social interactions, and their interplay with social relations, in the transmission of novel information that underlies influence processes. Furthermore, the interactional dynamics of individuals in specific structural positions, such as brokers or stars, could be investigated with the aim to understand not only the origins of the position but also the mechanisms that enable these individuals to reproduce their positions.

Similarly, research on project teams and group work is an area where additional understanding of dynamics has been called for (Stewart, 2010) and where the model we presented here could be adopted to address a number of substantive questions. Because the changes in the composition of a team can be taken into account directly in the model, understanding the dynamics of project teams as they evolve over time would be much facilitated. Specifically, the relational event model would be very well suited to understand how project teams that experience important compositional changes over time are able to stay cohesive (e.g., Quintane et al., 2013). Is this cohesiveness more important at different phases of the development of the project? How does a core group of team members manage to integrate new members quickly and efficiently in a constantly evolving project team? In addition, the two-mode specification of model would be specifically appropriate for the empirical analysis team processes such as dynamics of project team affiliation (Carley, 1991) with a potential aim to explain knowledge transfer across teams, or the evolution of entrepreneurial teams (Hillmann & Aven, 2011).

At an interorganizational level, applications of the model can be conceived when investigating patterns of resource exchange between firms (Podolny, 1994; Uzzi, 1997), joint ventures or strategic alliances (Gulati, 1995, 1998; Stuart, 1998), or interorganizational coalitions (Rosenkopf & Schleicher, 2008). From a two-mode perspective, the study of interlocking boards of directors (Davis, 1991), venture capital investment (Sorenson & Stuart, 2008) or more generally interactions with financial institutions (Mizruchi, 1992) could benefit from analyzing directly sequences of relational events. Here, the relational event model could help distinguish between patterns that are evocative of relationships between organizations from those that are based on market transactions and explore the mechanisms that drive each of these types of ties.

We conclude by highlighting a potential direction for future research on relational event models. In our empirical example, we have expressly followed Butts (2008) by reporting deviance measures of fit for each model. A novel approach to fit would be similar to goodness-of-fit procedures used in ERGM of social networks (see Hunter, Goodreau, & Handcock, 2008) where the model is simulated from parameter estimates and the observed network is compared with graphs from the simulated models on a number of nonparameterized features. Standard graph statistics and properties can be

utilized for this comparison. For event sequences, however, careful thought needs to be given to the nonparameterized features that are best for such a goodness-of-fit comparison, as our understanding of relational event dynamics is at an early stage and the development of useful descriptive statistics is best seen as an open problem.

Appendix

Road Map to Model Relational Events

Input Files. A key aspect of building a relational event model is formatting the input data files correctly. There should be at least two distinct (text) files. The first file (event file) contains information about the event sequence (order of events, actors involved, time, and any other information that pertains to each event). More specifically in our empirical context the event file contains:

Event file

Id Event: continuous integer

Id Sender: continuous integer

Id Recipient: continuous integer

Time: continuous integer

Developer/Bug first and last action: Four columns, binary variables, 1 if first/last action, 0 otherwise
Other event specific characteristics (e.g., effort)

Beyond the event file, there should be at least one other file containing information about actor attributes. In the case of events linking different sets of actors (two-mode networks), one attribute file should be created for senders and a separate one for recipients. These files list all the senders and all the recipients that are involved at any point in time in at least one event in the event file. Each file contains an id number for the sender/recipient and any additional characteristic of interest as an additional column. The id corresponds to the one used in the events file. More specifically in our empirical context:

Attribute files. Id Sender (or Recipient): continuous integer
Active Sender (or Recipient): binary variable (1 active in prior release cycles, 0 not active in prior release cycles)
Other characteristics of sender/recipient (e.g., core developer)

Java Program. The goal of the Java program is to combine the event and attribute files and calculate the statistics that will be used for model estimation in SPSS. The output of the Java program is a text file with all events and potential events in rows and characteristics of the event, including id of the event, sender, recipient, time, and all statistics in columns. More specifically, the operation of the Java program follows these broad logical lines:

1. Declare variables
   a. Arrays to store events, developers, bugs
   b. Arrays to compute actions and statistics
   c. Arrays to track active developers and bugs
   d. Variable to store if event is observed or not (obs)
2. Import event file
   a. Store in array (event)
3. Import attributes file
   a. Store in array (developers/bugs)

4. Initialize arrays
   a. Actions and statistics are initialized to 0
   b. Active developers and bugs initialized to values of attribute files
5. Loop for all events
   a. Assign events values for sender, recipient, time
   b. Update active bugs/developers with values of current event (developer/bug first action)
   c. Start with second event (there is no history to predict the first event)
      i. Loop through all active developers and active bugs
      ii. If current bug and developer are those of the event, then obs = 1 else obs = 0
      iii. Calculate statistics (see Table 2)
         1. For example: Core developer.
            a. Statistics array receives one if developer is core, 0 otherwise
         2. For example: Prior edge
            a. Prior edge statistic receives value of action counter between current developer and current bug
         3. For example: Developer activity
            a. Loop through all bugs
            b. Developer activity statistic receives value of action counter for all events that included the current developer
      iv. Scale Statistics (see Table 2)
         1. For example: Prior edge scaled using the value of activity developer
         2. For example: Activity and popularity scaled using the number of events
      v. Write statistics to file
         1. Event number, senderid, recipientid, obs
         2. All statistics
         3. Time, timev variable (= 2 if event is not observed, = 1 if event is observed)
   d. Update action array with values of current observed event (i.e., increment the array by 1 for current sender and recipient).
   e. Update active bugs/developers with values of current event (developer/bug last action).

SPSS Syntax (Estimation). After the Java program has created the output file, SPSS will be used to import the file, standardize all variables, and estimate the model.

1. Import file (output from the Java program)
2. Standardize variables (DESCRIPTIVES command)
3. Estimate Model (COXREG command)

   a. Time = timev
   b. Status = obs
   c. Strata = Id event

Additional Elements

Time-varying statistics

1. Declare time-varying statistics (one array per threshold, e.g., one week)
2. Initialize time-varying arrays to 0
3. Loop through all past events
   a. if the event is within temporal threshold then update time-varying array

For example, if past event occurred within one week of current event, then week statistics for current sender and current recipient is updated.

Then calculate statistics, including time varying, and write to file.

Dyadic covariates. Include each dyadic covariate as a separate text file, in matrix or edge list format.

In the Java program, import dyadic covariate, and match for each event sender recipient pairs to the value of the dyadic covariate.

Time-varying covariates. For covariates that change at fixed points in time (e.g., core developers), it is enough to add columns to the attribute file with the value for each time frame. Then, in the Java program, match the time of the event to the time of the column to use the correct value for the covariate.

For covariates that change continuously over the period of observation, it is easier to format directly the event file to track the changes in the value of the covariate for each event.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship and/or publication of this article: The research leading to this article was made possible, in part, by the generous financial support provided by the Schweizerscher Nationalfonds (Swiss National Science Foundation, Projects 133273 and 142338).

Notes

1. We define *contributors* as encompassing three types of participants in free/open source software projects as classified by Crowston and Howison (2005): "developers," "co-developers," and "active users." Developers are active both in the production of new code and in bug-fixing activities, whereas co-developers and active users are mainly active in bug fixing.
2. All software bugs are reported using a dedicated online bug repository, and all software bug reports are accessible by the entire community of Epiphany contributors. When opened, a software bug receives a unique identification number. All actions taken on a given bug are reported in the online bug repository against the identification number of the software bug, with a time stamp and the user name of the contributor who took the action. In addition, contributors receive automatic notifications of newly reported software bugs and may add themselves or others to receive further notifications concerning actions taken on the specific software bug. A software bug can be closed or solved, even though it might be reopened at a later stage if the solution was not viable.
3. Specifically, we classify as low-effort actions (a) triaging-based actions (e.g., changing the severity level, modifying the description of a software bug, setting a new milestone), (b) people-based actions (e.g., coordinating other developers by subscribing them to relevant software bug reports), and (c) resolution-based actions (e.g., signaling that a software bug has been fixed or needs further action).
4. The raw data were collected by parsing the web pages of all relevant bug reports with the software Bicho (Robles, González-Barahona, Izquierdo-Cortazar, & Herraiz, 2009).
5. While we collected exact timing information for each action, we use time information only to reconstruct the sequence of events. In our empirical context, the information about sequence is more instructive than the exact timing between actions, because actions have exogenous influences to their rates of occurrence and because we are more interested in the likelihood that certain patterns of past actions will lead to a future action rather than when the future action will occur.

6. The code of the Java program is available from the first author at www.quintane.net.
7. We are indebted to an anonymous reviewer for highlighting the nonlinear nature of activity and popularity.
8. See http://mail.gnome.org/archives/epiphany-list/2008-April/msg00000.html.
9. Note that the lack of statistical significance of some parameter estimates in high-effort actions reflects empirically grounded processes rather than being an artifact of differences in statistical power between high-effort and low-effort actions.
10. The parameter estimate for four cycle in low-effort actions is .342 with a standard error of .009. The parameter estimate for four cycle in high-effort actions is .034 with a standard error of .004. The correlation between the parameter estimates is –.03 (the value can be obtained using the CORR option in the COXREG command in SPSS). The z test between high-effort and low-effort four cycle is equal to the difference between the parameter estimates divided by the estimated standard error for the difference between the parameter estimates, which is the square root of the variance of the difference between the parameter estimates. The variance of the difference between the parameter estimates is equal to the variance of the first parameter estimate plus the variance of the second parameter estimate minus twice the covariance between the two parameter estimates. Variance is the square of the standard error. A z test above 2.33 indicates a difference that is significant at $\alpha = .01$.
11. We note here that the effect of popularity of software bugs found to be weakly significant in Model 1, negative and significant in Model 2, and nonsignificant in Models 3 and 4 is different from the effect bearing the same name reported by Conaldi, Lomi, and Tonellato (2012) in their analysis and found there as significant and positive. We interpret our result as indicative that the past popularity among contributors of a software bug is not a good predictor of its future popularity. One way to understand the difference in results, assuming everything else equal, is that Conaldi et al. measure popularity as the number of distinct contributors who have acted (at least once) on a software bug, while we measure popularity as the total number of actions received by a software bug.

References

Ahuja, G., Soda, G., & Zaheer, A. (2011). The genesis and dynamics of organizational networks. *Organization Science*, *23*(2), 434-448.

Ahuja, M. K., & Carley, K. M. (1999). Network structure in virtual organizations. *Organization Science*, *10*(6), 741-757.

Agresti, A. (2002). *Categorical data analysis* (2nd ed.). New York, NY: John Wiley.

Anderson, B. S., Butts, C., & Carley, K. (1999). The interaction of size and density with graph-level indices. *Social Networks*, *21*(3), 239-267.

Aral, S., Muchnik, L., & Sundararajan, A. (2009). Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks. *PNAS*, *16*(51), 21544-21549.

Aral, S., & Van Alstyne, M. (2011). The diversity-bandwidth tradeoff. *American Journal of Sociology*, *117*(1), 90-171.

Borgatti, S. P., Everett, M. G., & Freeman, L. C. (2002). *Ucinet for Windows: Software for social network analysis*. Cambridge, MA: Analytic Technologies.

Borgatti, S. P., & Halgin, D. S. (2011). On network theory. *Organization Science*, *22*(5), 1168-1181.

Brandes, U., Lerner, J., & Snijders, T. A. B. (2009, July). *Networks evolving step by step: Statistical analysis of dyadic event data*. Paper presented at the International Conference on Advances in Social Network Analysis and Mining, Athens, Greece (pp. 200-205). doi:10.1109/ASONAM.2009.28

Breiger, R. (2000). A tool kit for practice theory. *Poetics*, *27*, 91-115.

Breiger, R., & Mohr, J. W. (2004). Institutional logics from the aggregation of organizational networks: Operational procedures for the analysis of counted data. *Computational and Mathematical Organization Theory*, *10*, 17-43.

Butts, C. T. (2008). A relational event framework for social action. *Sociological Methodology*, *38*(1), 155-200.

Butts, C. T. (2009). Revisiting the foundations of network analysis. *Science*, *325*, 414-416.

Carley, K. (1991). A theory of group stability. *American Sociological Review*, *56*, 331-354.

Cattani, G., & Ferriani, S. (2008). A core/periphery perspective on individual creative performance: Social networks and cinematic achievements in the Hollywood film industry. *Organization Science*, *19*, 824-844.

Cohen, M. D., March, J. G., & Olsen, J. P. (1972). A garbage can model of organizational choice. *Administrative Science Quarterly*, *17*(1), 1-25.

Conaldi, G., Lomi, A., & Tonellato, M. (2012). Dynamic models of affiliation and the network structure of problem solving in an open source software project. *Organizational Research Methods*, *15*(3), 385-412.

Crowston, K., & Howison, J. (2005). The social structure of free and open source software development. *First Monday*, *10*(2). Retrieved from http://journals.uic.edu/ojs/index.php/fm/article/view/1207

Crowston, K., & Scozzi, B. (2008). Bug fixing practices within free/libre open source software development teams. *Journal of Database Management*, *19*(2), 1-30.

David, P. A., & Rullani, F. (2008). Dynamics of innovation in an "open source" collaboration environment: Lurking, laboring, and launching FLOSS projects on SourceForge. *Industrial and Corporate Change*, *17*(4), 647-710.

Davis, G. F. (1991). Agents without principles? The spread of the poison pill through the intercorporate network. *Administrative Science Quarterly*, *36*(4), 583-613.

de Nooy, W. (2011). Networks of action and events over time: A multilevel discrete-time event history model for longitudinal network data. *Social Networks*, *33*(1), 31-40.

Dempsey, B. J., Weiss, D., Jones, P., & Greenberg, J. (1999). *A quantitative profile of a community of open source Linux developers*. Unpublished manuscript. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.9791

Ducheneaut, N. (2005). Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work*, *14*, 323-368.

Fang, Y., & Neufeld, D. (2009). Understanding sustained participation in open source software projects. *Journal of Management Information Systems*, *25*(4), 9-50.

Grannis, R. (2010). Six degrees of "who cares?" *American Journal of Sociology*, *115*(4), 991-1017.

Grund, T. U. (2012). Network structure and team performance: The case of English Premier League soccer teams. *Social Networks*, *34*(4), 682-690.

Gulati, R. (1995). Social structure and alliance formation patterns: A longitudinal analysis. *Administrative Science Quarterly*, *40*(4), 619-652.

Gulati, R. (1998). Alliances and networks. *Strategic Management Journal*, *19*(4), 293-317.

Hess, A. M., & Rothaermel, F. T. (2011). When are assets complementary? Star scientists, strategic alliances and innovation in the pharmaceutical industry. *Strategic Management Journal*, *32*, 895-909.

Hillmann, H., & Aven, B. L. (2011). Fragmented networks and entrepreneurship in late imperial Russia. *American Journal of Sociology*, *117*, 484-538.

Hunter, D., Goodreau, S., & Handcock, M. (2008). Goodness of fit of social network models. *Journal of the American Statistical Association*, *103*, 248-258.

Koskinen, J., & Edling, C. (2012). Modelling the evolution of a bipartite network—Peer referral in interlocking directorates. *Social Networks*, *34*(3), 309-322.

Lakhani, K. R., & von Hippel, E. (2003). How open source software works: "Free" user-to-user assistance. *Research Policy*, *32*(6), 923-943.

Lazega, E., Mounier, L., Snijders, T., & Tubaro, P. (2012). Norms, status and the dynamics of advice networks: A case study. *Social Networks*, *34*(3), 323-332.

Lazer, D., Pentland, A., Adamic, L., Aral, S., Barabasi, A.-L., Brewer, D., . . . Van Alstyne, M. (2009). Social science: Computational social science. *Science*, *323*(5915), 721-723.

Lee, G. K., & Cole, R. E. (2003). From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organization Science*, *14*(6), 633-649.

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics*, *50*(2), 197-234.

Leydesdorff, L. (1995). *The challenge of scientometrics: The development, measurement, and self-organization of scientific communication*. Leiden, Netherlands: DSWO Press.

Lusher, D., Koskinen, J., & Robins, G. (Eds.). (2012). *Exponential random graph models for social networks: Theories, methods and applications*. New York, NY: Cambridge University Press.

Markus, M. L. (2007). The governance of free/open source software projects: Monolithic, multidimensional, or configurational? *Journal of Management & Governance*, *11*(2), 151-163.

Mizruchi, M. (1992). *The structure of corporate political action: Interfirm relations and their consequences*. Cambridge, MA: Harvard University Press.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, *11*(3), 309-346.

Murray, F., & O'Mahony, S. (2007). Exploring the foundations of cumulative innovation: Implications for organization science. *Organization Science*, *18*(6), 1006-1021.

O'Mahony, S. (2003). Guarding the commons: How community managed software projects protect their work. *Research Policy*, *32*(7), 1179-1198.

Opsahl, T., Agneessens, F., & Skvoretz, J. (2010). Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, *32*, 245-251.

Podolny, J. M. (1994). Market uncertainty and the social character of economic exchange. *Administrative Science Quarterly*, *39*(3), 458-483.

Quintane, E., & Kleinbaum, A. (2011). Matter over mind? E-mail data and the measurement of social networks. *Connections*, *31*(1), 22-46.

Quintane, E., Pattison, P. E., Robbins, G. L., & Mol, J. M. (2013). Short- and long-term stability in organizational networks: Temporal structures of project teams. *Social Networks*, *35*, 528-540.

Qureshi, I., & Fang, Y. (2011). Socialization in open source software projects: A growth mixture modeling approach. *Organizational Research Methods*, *14*(1), 208-238.

Raymond, E. S. (1999). *The cathedral & the bazaar*. Sebastopol, CA: O'Reilly.

Rivera, M. T., Soderstrom, S. B., & Uzzi, B. (2010). Dynamics of dyads in social networks: Assortative, relational, and proximity mechanisms. *Annual Review of Sociology*, *36*(1), 91-115.

Robins, G., & Alexander, M. (2004). Small worlds among interlocking directors: Network structure and distance in bipartite graphs. *Computational & Mathematical Organization Theory*, *10*, 69-94.

Robles, G., González-Barahona, J. M., Izquierdo-Cortazar, D., & Herraiz, I. (2009). Tools for the study of the usual data sources found in libre software projects. *International Journal of Open Source Software & Processes*, *1*(1), 24-45.

Rosenkopf, L., & Schleicher, T. (2008). Below the tip of the iceberg: The co-evolution of formal and informal interorganizational networks in the telecommunications industry. *Managerial and Decision Economics*, *29*, 425-441.

Snijders, T. A. B., van de Bunt, G. G., & Steglich, C. E. G. (2010). Introduction to stochastic actor-based models for network dynamics. *Social Networks*, *32*(1), 44-60.

Sorenson, O., & Stuart, T. E. (2008). Bringing the context back in: Settings and the search for syndicate partners in venture capital investment networks. *Administrative Science Quarterly*, *53*, 266-294.

Stadtfeld, C., & Geyer-Schulz, A. (2011). Analyzing event stream dynamics in two-mode networks: An exploratory analysis of private communication in a question and answer community. *Social Networks*, *33*, 258-272.

Stewart, D. (2005). Social status in an open-source community. *American Sociological Review*, *70*, 823-842.

Stewart, G. L. (2010). The past twenty years: Teams research is alive and well at the Journal of Management. *Journal of Management*, *36*, 801-805.

Stuart, T. E. (1998). Network positions and propensities to collaborate: An investigation of strategic alliance formation in a high-technology industry. *Administrative Science Quarterly*, *43*(3), 668-698.

Uzzi, B. (1997). Social structure and competition in interfirm networks: The paradox of embeddedness. *Administrative Science Quarterly*, *42*(1), 35-67.

van de Bunt, G. G., & Groenewegen, P. (2007). An actor-oriented dynamic network approach: The case of interorganizational network evolution. *Organizational Research Methods*, *10*, 463-482.

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

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

Wang, P., Pattison, P. E., & Robins, G. (2013). Exponential random graph model specifications for bipartite networks—A dependence hierarchy. *Social Networks*, *35*, 96-115.

Xu, J., Gao, Y., Christley, S., & Madey, G. (2005). A topological analysis of the open source software development community. In *Proceedings of the 38th annual Hawaii International Conference on System Sciences* (p. 198a). New York, NY: IEEE.

Zaheer, A., & Soda, G. (2009). Network evolution: The origins of structural holes. *Administrative Science Quarterly*, *54*, 1-31.

Author Biographies

Eric Quintane is an assistant professor of organization theory in the School of Management at the University of Los Andes, Colombia. His research currently focuses on the dynamics of social networks.

Guido Conaldi is a senior lecturer in economic sociology at the University of Greenwich, UK. He holds an MSc in sociology from the London School of Economics and a PhD in social sciences from the Sant'Anna School of Advanced Studies in Pisa, Italy. His research interests lie in the area of interpersonal and organizational social networks; his current research investigates the social mechanisms contributing to the endogenous formation of structure and hierarchy in self-managing groups.

Marco Tonellato is a PhD candidate at the Institute of Management (IMA) of the University of Italian Switzerland, Lugano and a visiting researcher at Tepper School of Business, Carnegie Mellon University, USA. His research interests include the analysis of social networks within and between organizations and the organizational aspects of open productions. His current research focuses on coordination and learning processes of self-managed groups in open source software projects.

Alessandro Lomi is a professor of organization theory and behavior in the Faculty of Economics at the University of Italian Switzerland, Lugano.