Quantitative Analysis of Community Evolution in Developer Social Networks Around Open Source Software Projects

Liang Wang, Ying Li, Jierui Zhang, Xianping Tao
State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing, China
wl@nju.edu.cn, mg21070006@smail.nju.edu.cn, jierui Zhang@smail.nju.edu.cn, txp@nju.edu.cn

ABSTRACT
Understanding the evolution of communities in developer social networks (DSNs) around open source software (OSS) projects can provide valuable insights about the socio-technical process of OSS development. Existing studies show the evolutionary behaviors of social communities can effectively be described using patterns including split, shrink, merge, expand, emerge, and extinct. However, existing pattern-based approaches are limited in supporting quantitative analysis, and are potentially problematic for using the patterns in a mutually exclusive manner when describing community evolution. In this work, we propose that different patterns can occur simultaneously between every pair of communities during the evolution, just in different degrees. Four entropy-based indices are devised to measure the degree of community split, shrink, merge, and expand, respectively, which can provide a comprehensive and quantitative measure of community evolution in DSNs. The indices have properties desirable to quantify community evolution including monotonicity, and bounded maximum and minimum values that correspond to meaningful cases. They can also be combined to describe more patterns such as community emerge and extinct. We conduct experiments with real-world OSS projects to evaluate the validity of the proposed indices. The results suggest the proposed indices can effectively capture community evolution, and are consistent with existing approaches in detecting evolution patterns in DSNs with an accuracy of 94.1%. The results also show that the indices are useful in predicting OSS team productivity with an accuracy of 0.718. In summary, the proposed approach is among the first to quantify the degree of community evolution with respect to different patterns, which is promising in supporting future research and applications about DSNs and OSS development.

KEYWORDS
Developer Social Networks, Community Evolution Patterns, Entropy-Bases Indices, Open Source Software, Mining Software Repositories

1 INTRODUCTION
Building and maintaining healthy communities around open source software (OSS) projects is crucial because of the socio-technical nature of OSS development [11, 19, 25, 40, 48, 60, 66]. Cross-disciplinary research on developer social networks (DSNs) that emerge from the developers’ communication and cooperation activities in OSS repositories has been conducted by scholars from various fields ever since the early days of the OSS movement [37, 45]. Supported by rich empirical evidences, it is now widely accepted that the social aspects of OSS communities have a strong correlation with, and are important indicators of OSS projects’ productivity [65], success [62], and survivability [44, 49, 64].

While a number of research is about the statistics and patterns of static DSNs snapshots [7, 16, 29, 59, 68], there is a series of work that pay special attentions to the evolution of DSNs around OSS projects over time [9, 25, 66]. By understanding the evolution of DSNs, we can not only retrospectively reveal the status of a DSN in a given snapshot, but also predict the future trends of a DSN based on a comprehensive view of its evolution history [9, 25, 48]. Given the importance of social aspects in OSS development as discussed above, and because OSS development is a “nature product of evolution” [40, 45], in this work, we study the problem of measuring the evolution of social communities around OSS projects quantitatively to support in-depth understanding of the socio-technical aspects of OSS development.

Existing approaches to study community and network evolution can be categorized into two classes which we refer to as network-level statistics and community-level patterns. The first class of work uses sequences of statistical metrics over series of temporally-aligned snapshots of social networks to quantify their evolution. Frequently adopted metrics include network diameter, shortest path between nodes, node betweenness, modularity, hierarchy, centrality, and etc [3, 25, 28, 30, 49, 54]. While most of the above metrics measure...
properties of social networks on a gross level, a natural phenomenon in social networks is that interacting individuals form complex structures organized in communities\(^1\) [8, 46]. Understanding the communities inside DSNs can provide valuable insights about software development. For instance, the Conway’s law suggests that a system’s architecture reflects the communication structure of the organization designs it [6, 15]. Although some metrics such as clustering coefficient [25], group degree centrality [44], group betweenness centrality [44], and etc, are related to measuring sub-structures or communities [12] in social networks, these network-level statistics are less expressive in revealing the structure and evolution of communities in DSNs.

Different from the above work, the second class of approaches take the community perspective [3] to uncover the evolution patterns of social networks. Six patterns, including growth (expand), contraction (shrink), merge, split, birth (emerge), and death (extinct) are used to describe the evolutionary behaviors of communities inside social networks [3, 4, 10, 23, 27, 34, 46]. These patterns are promising in bringing insights about OSS development for corresponding to the evolution of OSS projects and members’ activities [27]. However, the existing pattern-based approaches often assign a single pattern to describe the evolution of a community, which are limited for several reasons. First, the evolution of a community can hardly be described by a single pattern. For example, a community can expand and merge simultaneously with new members and members from other communities joining it. Choosing either pattern can lead to information loss in the above case. In addition, matching communities detected in different time steps to identify them as a single, evolving community before pattern assignment is not a trivial task [46]. Second, the nominal patterns are promising in describing community evolution qualitatively, but are limited in supporting quantitative analysis. Third, these patterns are restricted to describe the evolution of a single community. It is difficult to extend the concepts to the network level, which is often desirable to gain a comprehensive understanding of the evolution of DSNs.

Our idea to tackle the above limitations can be described as: the community evolution patterns can happen simultaneously between all pairs of communities in consecutive time steps, only in different degrees. Following this idea, we propose in this paper to quantify community evolutionary behaviors comprehensively by calculating how much the communities expand, shrink, merge, and split simultaneously. As introduced in Sec. 4, the calculation is done by first building a community member migration matrix which captures the dynamics of communities detected in Sec. 3. Based on the matrix, we devise indices based on information entropy to measure the degree of community evolution with respect to the four patterns above. Community extinct and emerge are represented by special combinations of the indices. With this approach, the evolution of a community is described by four indices in each step, which addresses the challenge of finding a unique, most appropriate pattern for community evolution. It is worth noting that our approach is compatible with existing work [27] for being capable of finding the most appropriate evolution pattern with a simple set of rules as introduced later. We present the properties of the indices, including the monotonicity, and the upper- and lower-bounds, to show that the indices can correctly reveal complex, real-world community evolutionary behaviors quantitatively. We also aggregate the community-level indices to describe the overall evolutionary behaviors of communities at the network level, which can support quantitative analysis of community evolution at various scales.

We conduct empirical studies using real-world data collected from thirty-two projects hosted on GitHub to evaluate the validity and usefulness of the proposed community evolution indices. More specifically, we validate the construct validity of the proposed indices in terms of concurrent and discriminant validity criteria [31]. The results for concurrent validity suggest that the community evolution patterns detected with the proposed indices is consistent with the ones detected following an exiting approach [27], with an accuracy of 94%. By analyzing the properties of the indices, and by calculating the Spearman’s Correlation Coefficients of the indices, we find that the indices are capable in measuring different aspects of community evolution. We further demonstrate the usefulness of the proposed indices by study the correlation between the measured community evolution with OSS project team productivity measured by the count of commits [11, 63]. The results obtained from regression analysis with linear mixed-effects models suggest that the indices are good predictors of team productivity with an accuracy of 0.718 after log transformation.

The main contributions of this paper are as follows. First, we bridge the gap between nominal, community evolution patterns and quantitative measures with a set of entropy-based indices to support quantitative analysis for community evolution. Second, we demonstrate versatile ways of using the indices, e.g., aggregating community-level indices to generate network-level quantities, and finding the most appropriate pattern from the indices, which can support various tasks. Finally, we conduct empirical studies with data from real-world OSS projects to validate the proposed indices, and demonstrate the usefulness of the indices in predicting OSS team productivity.

2 OVERVIEW OF THE PROPOSED APPROACH

This section presents an overview of the proposed approach.

2.1 Terminology

We first introduce the concepts and terms used in this paper.

An OSS project or repository refers to a single repository hosted on OSS platforms such as Github, e.g., tensorflow/tensorflow\(^2\). In comparison, large projects, e.g., the Mozilla or GNOME projects that include multiple sub-projects developed together [27, 39], are considered as ecosystems or super-repositories following [36, 39]. In this work, we focus on community evolution on the project level.

A developer social network (DSN) around an OSS project include the participants of the project and their relationships, which is typically modeled using a graph model [3, 27] extracted from various sources such as mailing lists [7, 39, 48, 66], version repositories [39], issue and bug-tracking systems [3, 48], and etc.

\(1\)For the rest of the paper, a community inside a social network refers to a group of interconnected members of the network if not specified otherwise. Refer to Sec. 2.1 for details about the terms used in this paper.

\(2\)https://github.com/tensorflow/tensorflow
A community inside a DSN means a group of members in the DSN with strongly interconnected links [46], or share similar features and actions [40, 52], which is also termed as a cluster [25, 52], a clique [46, 54], an emerging team [48], or a social group [46] in the literature. We do not use the term to refer to OSS communities in the broad sense [56], e.g., the Apache community studied in [66].

The evolutionary behaviors of communities studied in this paper correspond to the six community evolution patterns studied in [3, 27, 34, 46], which is different from the general concept of ‘patterns of community evolution’ used in [40, 66].

2.2 Overview

As shown in Fig. 1, the processing pipeline proposed in this work involves three steps. First, as shown in Fig. 1(a) and as introduced in Sec. 3 and Sec. 5, given an OSS repository, we retrieve data about the conversations that take place in issues and pull-requests from the GHTorrent data set [22] and the GitHub GraphQL API. Participants of the conversations are recorded to extract the joint discussion relationships between them. The records are used to build DSNs and to perform community evolution analysis in the subsequent steps. Next, as shown in Fig. 1(b) and presented in Sec. 3, we segment the records with a sliding-window, and build snapshots of DSNs from the segments using weighted, undirected graphs. Community detection is then performed by applying an extension of the Clauset-Newman-Moore (CNM) algorithm for weighted graphs to the snapshots. Finally, correspond to Fig. 1(c), we measure the evolutionary behaviors, including the split, shrink, merge, and expand, of the communities detected in consecutive steps with the information entropy-based indices proposed in Sec. 4. The index values are aggregated to quantify community evolution at the network level before we performing analysis in Sec. 5.

3 DSN AND COMMUNITY DETECTION

In this work, we use a weighted, undirected graph \( G = (V, E, U, W) \) to model the structure of a DSN, where a vertex \( u_i \in V \) represents a user, and an edge \( e_{ij} \in E \) represents social interactions between users \( u_i \) and \( u_j \). A positive, real-valued weight \( w_{ij} \in U \) is assigned to each vertex \( u_i \) following Eq. (2) to quantify the user’s importance. And a weight \( w_{ij} \in W \) is assigned to each edge \( e_{ij} \) following Eq. (1) to quantify the strength of social interactions between the users connected by the edge. The type of social interaction studied in this paper is the joint discussion relationship between users on OSS platforms such as GitHub. Platform users can post issues and pull-requests to an OSS repository to ask questions and making contributions [8, 59]. They can also provide responses by leaving comments under posted issues and pull-requests. We use the term a conversation to refer to the collection of the initial body and all the comments to a single issue or pull-request. There is an edge \( e_{ij} \) between two users, \( u_i \) and \( u_j \), if they join the same conversation.

3.1 Construct DSN Snapshots

We construct a series of DSN snapshots from the conversations of an OSS repository.

3.1.1 Data Segmentation. Given a series of temporally ordered conversations obtained from a project’s repository, \( D = (d_1, d_2, \ldots, d_N) \), where \( d_i = (t_i, v_i) \), \( i = 1 \cdots N \), records the creation time \( t_i \) and the set of participants \( \forall V_i \) of the \( i \)-th record, we apply a non-overlapping sliding window of length \( \omega \) to divide \( D \) into a series of segments \( \hat{S} = (s_1, s_2, \ldots, s_T) \). The \( t \)-th segment in \( \hat{S} \) include conversations created in the \( t \)-th window, i.e., \( s_t = \{ d_i | t_1 + (t - 1) \omega \leq t_i < t_1 + t \omega \} \), \( t = 1 \cdots T \), where \( t_1 \) is the creation time of \( d_1 \).

3.1.2 Build Local DSNs from Data Segments. For each segment \( s_t \in \hat{S} \), we build its corresponding local DSN \( G_t = (V^L_t, E^L_t, U^L_t, W^L_t) \) by letting \( V^L_t \) to include all the participants of conversations in \( s_t \), and \( E^L_t \) to include all the joint discussion relationships between the users in \( V^L_t \). There is an edge between user \( u_i \) and \( u_j \) if there is at least one conversation \( d_k \in s_t \) that both users are participants of \( d_k \), \( U^L_t \) and \( W^L_t \) include the weights of users and edges in the local DSN, which are calculated as follows.

For each edge \( e_{ij} \in E^L_t \), there is a weight \( w_{ij} \in W^L_t \) that quantifies the strength of interactions between user \( u_i \) and \( u_j \). Let \( cnt_{ij} \) be the count of conversations both \( u_i \) and \( u_j \) participate in segment \( s_t \), we assign the weight \( w_{ij} \) as follows.

\[
\text{weight} (w_{ij}) = \ln(cnt_{ij} + 1) \quad (1)
\]

We take the natural logarithm of counts to prevent large counts to overwhelm small ones. From Eq. (1), we have \( w_{ij} = 0 \) if \( cnt_{ij} = 0 \), and \( w_{ij} > 0 \) if \( cnt_{ij} \geq 0 \). Note we always have \( cnt_{ij} = 0 \) if \( i = j \).

Further, we assign a weight \( u_i \in U^L_t \) for each user \( u_i \in V^L_t \). The participants of a conversation include the author and the users leave comments in the issue or pull-request.

---

1https://docs.github.com/en/graphql
we adopt the extension \([12, 42]\) of the Clauset-Newman-Moore (CNM) greedy modularity maximization algorithm \([41, 43]\) to detect the communities within it. Developed from the traditional CNM algorithm for unweighted graphs, the extended version of CNM performs community detection on weighted graphs using the strength of connections between nodes indicated by the edge weights.

The result of community detection for \(G_t\) is a set of non-overlapping communities \(C_t = \{c_{t,1}, c_{t,2}, \cdots, c_{t,n}\}\), where \(c_{t,i} \subseteq V_t\) contains a subset of users in the social network with strong internal joint discussion relationships. Communities involve only a single member are removed. By applying community detection to all the snapshots in \(\tilde{G}\), we obtain a series of evolving communities over time \(\tilde{C} = \{C_1, C_2, \cdots, C_t, \cdots, C_T\}\). In the next section, we propose metrics to measure community evolution in \(\tilde{C}\).

### 4 COMMUNITY EVOLUTION INDICES

Based on the temporally ordered series of communities \(\tilde{C}\) detected in the above section, this section presents the process and metrics proposed to measure community evolution.

#### 4.1 Community Member Migration Matrix

Given two consecutive sets of communities in \(\tilde{C}\), e.g., \(C_t\) and \(C_{t+1}\), we compute a community member migration matrix to quantify member migration between the communities as shown in Fig. 2(a). Let \(n = |C_t|\) and \(m = |C_{t+1}|\) be the number of communities in \(C_t\) and \(C_{t+1}\) (without loss of generality, we assume \(n \geq 1\) and \(m \geq 1\), respectively), we obtain a \((n + 1) \times (m + 1)\) matrix \(M_t\) where the \(ij\)-th element \(a_{i,j}\) is the number of members migrate from \(c_{t,i}\) to \(c_{t+1,j}\), calculated following Eq. (3), for \(i \leq n\) and \(j \leq m\).

\[
a_{i,j} = c_{t,i} \cap c_{t+1,j}, \; i = 1, 2, \ldots, n; \; j = 1, 2, \ldots, m
\]

where \(c_{t,i} \cap c_{t+1,j}\) is the set of common users in \(c_{t,i}\) and \(c_{t+1,j}\), i.e., the set of users in \(c_{t,i}\) who migrate to \(c_{t+1,j}\). Member migration between exiting communities in consecutive time steps correspond to the solid arrows in Fig. 2(b) and (c), respectively.

Two Null communities, \(Null_{t+1}\) and \(Null_t\), are added to \(C_{t+1}\) and \(C_t\) to include members not involved in these communities. The \(Null_{t+1}\) community involves members of a community in \(C_t\) who leave the project in time \(t + 1\), i.e., members of \(C_t\) who do not show in any community in \(C_{t+1}\). And the \(Null_t\) community include newly joined members in a community in \(C_{t+1}\) who do not
appear in any community in $C_t$. Elements related to the Null communities are in the last column and row of $M_t$ as illustrated in Fig. 2(a), which are calculated following Eq. (4).

$$b_i = c_{t,i} \setminus \bigcup_{j=1}^{m} a_{i,j}, \ 1 \leq i \leq n$$

$$d_j = c_{t+1,j} \setminus \bigcup_{i=1}^{n} a_{i,j}, \ 1 \leq j \leq m$$

(4)

where $\bigcup_{i=1}^{n} a_{i,j}$ is the set of members in $c_{t,i}$ who stay in the project (and join an arbitrary community) in time $t+1$, and $\bigcup_{i=1}^{n} a_{i,j}$ is the set of members in $c_{t+1,j}$ who migrate from an existing community in time $t$. As a result, members from $c_{t,i}$ who leave the project (into Null$t+1$), and members in $c_{t+1,j}$ who are new comers (from Null$t$) are contained in sets $b_i = c_{t,i} \setminus \bigcup_{j=1}^{m} a_{i,j}$ and $d_j = c_{t+1,j} \setminus \bigcup_{i=1}^{n} a_{i,j}$, which correspond to the dashed arrows in Fig. 2(b) and (c), respectively. We omit time indicators for matrix elements, i.e., $a_{i,j}$, $b_i$, and $d_j$, to keep it concise.

The weights on arrows in Fig. 2(b) quantify the distribution of members migrate to different communities from $c_{t,i}$ with respect to their weights at time $t$, which are obtained following Eq. (5).

$$\psi_{i,j} = \frac{f_t(a_{i,j})}{f_t(c_{t,i})}, \ j = 1, 2, ..., m,$$

$$\eta_i = \frac{f_t(b_i)}{f_t(c_{t,i})}$$

(5)

where $a_{i,j}$ and $b_i$ are sets of users as defined in Eq. (3) and Eq. (4), respectively; $c_{t,i}$ is the set of the community’s members; let $x$ be a set of users, function $f_t(x)$ is the sum of weights for users in set $x$, and the subscript $t$ indicate the weights are given by snapshot $G_t$:

$$f_t(x) = \sum_{u_i \in x} u_i, u_i \in U_t, x \subseteq V_t$$

Similarly, the weights in Fig. 2(c) are calculated following Eq. (6) to quantify the distribution of members in $c_{t+1,j}$ from different sources based on their weights in snapshot $G_{t+1}$ given by $f_{t+1}(\cdot)$.

$$\phi_{i,j} = \frac{f_{t+1}(a_{i,j})}{f_{t+1}(c_{t+1,j})}, \ i = 1, 2, ..., n,$$

$$\mu_j = \frac{f_{t+1}(d_j)}{f_{t+1}(c_{t+1,j})}$$

(6)

4.2 Entropy-Based Community Evolution Indices For A Single Community

In this section, we propose to measure the evolution of a single community between consecutive time steps, i.e., time $t$ and $t+1$, based on the information entropy of the weight distributions shown in Fig. 2(b) and (c). While existing works describe community evolution with six patterns including split, shrink, extinct, merge, expand, and emerge [27], we omit the extinct and emerge patterns because they are special cases of community evolution studied in this work, as discussed in Sec. 4.2.4. More importantly, different from existing works [27, 46] that assign a single pattern to each matched pair of communities, we propose that the evolutionary behaviors take place in every pair of communities in consecutive time steps simultaneously, just a matter of degrees. As a result, we do not require to find the match between prior and post communities.

4.2.1 The split and shrink indices of community $c_{t,i}$. We use a community detected in time $t$, i.e., $c_{t,i}$, as shown in Fig. 2(b) to explain how the split and shrink indices are calculated.

Given the $m + 1$ communities in the next time step (which include the $m$ detected communities in $C_{t+1}$, and Null$t+1$), and the weight vector $(\psi_{1,j}, \psi_{2,j}, \cdots, \psi_{m,j}, \eta_i)$, we obtain the distribution of users’ weights who stay in the project by normalizing their actual weights:

$$\hat{\psi}_{i,j} = \frac{\psi_{i,j}}{\sum_{j=1}^{m} \psi_{i,j}}$$

(7)

Clearly, we have $\sum_{j=1}^{m} \hat{\psi}_{i,j} = 1$. And we can calculate the information entropy [55] of the normalized weight vector by:

$$H_{c_{t,i}} = -\sum_{j=1}^{m} \hat{\psi}_{i,j} \log_2(\hat{\psi}_{i,j})$$

(8)

With $m$ fixed, the entropy $H_{c_{t,i}}$ is upper-bounded by the following maximum entropy obtained when $\hat{\psi}_{i,j} = \frac{1}{m}$, $j = 1, 2, \cdots, m$:

$$H_{c_{t,i}}^\ast = -\log_2\left(\frac{1}{m}\right)$$

(9)

Intuitively, $H_{c_{t,i}}$ measures the degree of $c_{t,i}$’s members spread into different communities (excluding Null$t+1$) in the next step. And $H_{c_{t,i}}^\ast - H_{c_{t,i}}$ measures the degree of $c_{t,i}$’s members gather to a single community in the next step. Both $H_{c_{t,i}}$ and $H_{c_{t,i}}^\ast - H_{c_{t,i}}$ are equal or above zero. Based on the entropy in Eq. (8), we compute community $c_{t,i}$’s split and shrink indices as follows.

The split index of $c_{t,i}$, denoted as $I_{c_{t,i}}^\psi$, measures the degree of separation of its members into concrete communities detected in the next step, which is calculated as:

$$I_{c_{t,i}}^\psi = (1 - \eta_i)H_{c_{t,i}}$$

(10)

where $\eta_i$ is defined in Eq. (5).

The shrink index of $c_{t,i}$, denoted as $I_{c_{t,i}}^\eta$, measures the degree of its members move to the Null$t+1$ community in the next step:

$$I_{c_{t,i}}^\eta = \eta_i(H_{c_{t,i}}^\ast - H_{c_{t,i}}) + \sigma_{\eta_i}$$

(11)

where $\sigma_{\eta_i}$ is defined as follows to guarantee that the shrink index is always positive when $m \geq 1$ and $\eta_i > 0$.

$$\sigma_{\eta_i} = \begin{cases} 0.5\eta_i, & m = 1, \\ 0, & otherwise. \end{cases}$$

(12)

4.2.2 Properties of the split and shrink indices. Fig. 3 illustrates the curves of the indices under varying conditions. The proposed split index, $I_{c_{t,i}}^\psi$, and shrink index, $I_{c_{t,i}}^\eta$, have the following properties that are desirable when measuring community evolution.

$I_{c_{t,i}}^\psi$ and $I_{c_{t,i}}^\eta$ are strictly monotonic increasing functions of $m$, given that $\eta_i$ is fixed, and $\hat{\psi}_{i,j} = 1/m, j = 1, 2, \cdots, m$. Intuitively, the property states that when a fixed portion of members in community $c_{t,i}$ leave the project (i.e., $\eta_i$ fixed), and the remaining members of $c_{t,i}$ who stay in the project migrate evenly to the communities detected in the next step (i.e., $\hat{\psi}_{i,j} = 1/m, j = 1, 2, \cdots, m$), the split and shrink indices increase with the increasing number of communities detected in the next step (i.e., $m$).

$I_{c_{t,i}}^\psi / I_{c_{t,i}}^\eta$ is a strictly monotonic increasing / decreasing function of $\eta_i$, respectively, given that $m > 1$ and the distribution of $\hat{\psi}_{i,j}, j = 1, 2, \cdots, m$ fixed. Intuitively, this property states that the
We omit the proof in this paper due to page limits.

Given \( m \) and \( \eta_i \), the maximum split index
\[
I_{c_{\ell},i}^\eta = \left(1 - \eta_i\right) H_i^{s+1}\n
\] is obtained when the members of \( c_{\ell,i} \) migrate to the communities detected in the next step with a even distribution, i.e., when we have \( \hat{\psi}_{j,i} = 1/m, j = 1, \ldots, m \). And the minimum split index
\[
I_{c_{\ell},i}^\phi = 0
\]
\[
= 0
\]
is obtained when all the members of \( c_{\ell,i} \) stay in the project to migrate to a single community in the next step, i.e., there exists a \( j' \)-th community in time \( t+1 \) that \( \hat{\psi}_{j',i} = 1 \) and \( \hat{\psi}_{j,i} = 0, \forall j \neq j' \), resulting in \( H_{c_{\ell},i} = 0 \) and \( I_{c_{\ell},i}^\phi = 0 \).

Given \( m > 1 \) and \( \eta_i \), the maximum shrink index
\[
I_{c_{\ell},i}^\eta = \eta_i H_i^{s+1}\n
\]
is obtained when the corresponding split index is minimized, i.e., all stayed members of community \( c_{\ell,i} \) migrate to a single community in the next step. And the minimum shrink index
\[
I_{c_{\ell},i}^\phi = \eta_i H_i^{s+1}\n
\]
is obtained when the members of \( c_{\ell,i} \) migrate to a single community in time \( t+1 \). For the special case of \( m = 1 \), the shrink index is only determined by \( \eta_i \) and \( \eta_i \).

It is straightforward to proof the above properties given the indices’ definitions and the properties of information entropy [55]. We omit the proof in this paper due to page limits.

4.2.3 The merge and expand indices of \( c_{t+1},i \). Next, to measure community evolution on community \( c_{t+1},i \)'s side as shown in Fig. 2(c), we develop the merge and expand indices of \( c_{t+1},i \) following a similar framework as described above.

The merge index of \( c_{t+1},i \), denoted as \( \tilde{T}_{c_{t+1},i}^\phi \), measures the degree of its members that come from different source communities in the previous time step, which is calculated as follows:
\[
\tilde{T}_{c_{t+1},i}^\phi = (1 - \mu_j) H_{c_{t+1},i}^\phi
\]

where \( \sigma_{j} \) is as defined similar to \( \sigma_j \), as follows.
\[
\sigma_{j} = \begin{cases} 0.5 \mu_j, & n = 1, \\ 0, & \text{otherwise.} \end{cases}
\]

The entropy values in the above equations are computed by:
\[
H_{c_{t+1}^s} = -\log_2\left(\frac{1}{n}\right), \quad H_{c_{t+1}^s} = -\sum_{i=1}^{n} \hat{\phi}_{j,i} \log_2(\hat{\phi}_{j,i})
\]

where \( \hat{\phi}_{j,i} = \frac{\phi_{j,\ell}}{\phi_{\sum_{j'} \phi_{j',\ell}}} \) is the normalized weights.

The merge and expand indices have similar properties as split and shrink, respectively. We omit the details to keep the paper concise.

4.2.4 Community Extinction and Emergence. With the above indices, community extinction and emergence are two special combinations of index values as follows:

The extinction of a community \( c_{\ell, i} \) happens if all of its members leave the project and migrate to Nulll, which leads to the case that \( \eta_i = 1 \) and \( H_{c_{\ell},i} = 0 \). In this case, the shrink index takes the maximum value of \( I_{c_{\ell},i}^\phi = H_i^{s+1} + \sigma_j \). And the split index takes the minimum value of \( I_{c_{\ell},i}^\phi = 0 \).

The emergence of a new community \( c_{t+1},i \) happens if all its members are from \( \text{Nulll} \), in which case we have \( \mu_j = 1 \) and \( H_{c_{t+1},i} = 0 \). In this case, the expand index takes the maximum value of \( I_{c_{t+1},i}^\phi = H_i^{s+1} + \sigma_j \). And the merge index takes the minimum value of \( I_{c_{t+1},i}^\phi = 0 \).

There are more combinations of the index values that form meaningful cases, which we leave for our future studies.

4.3 Aggregate Indices for All Communities

Following the above process, for each community we can obtain indices that quantify its degree of split and shrink with respect to the next step, and its degree of merge and expand with respect to the previous step. We calculate the aggregated indices for the four patterns across all communities in the snapshot by taking the weighted average of the communities’ indices, where the weight for a community is the sum of node weights in the community over the total weight of nodes in all communities. We can then obtain four aggregated indices that measure the overall, snapshot-level community evolution for each time step.

5 EMPIRICAL STUDIES

In this section, we conduct empirical studies to evaluate the validity and usefulness of the proposed community evolution measures.

5.1 Preliminaries

5.1.1 Data Set. As summarized in Table 1, we collect a data set involving thirty-two OSS projects on GitHub to conduct the studies. The projects are randomly selected from a candidate set of popular (with #stars ≥ 1000) projects with sufficient data (with #Convs ≥ 800) that cover different topics including programming.
Table 1: Overview of the data set, where # Conv. is the count of conversations, # Weeks is the time duration of the project's data, and Status indicates whether the project is active by the end of May, 2019.

| No. | Project                  | #Conv. | #Weeks | Status  |
|-----|-------------------------|--------|--------|---------|
| 1   | angular-ui/bootstrap    | 34783  | 342    | inactive |
| 2   | apache/couchdb          | 9211   | 522    | active  |
| 3   | beautify-web/js/beautify| 6151   | 475    | active  |
| 4   | boldhtml/bolt           | 2830   | 221    | inactive |
| 5   | carrierwaveuploader/cARRIERwave| 10531  | 528    | active  |
| 6   | celery/celery           | 26594  | 521    | active  |
| 7   | Compass/compass         | 2692   | 484    | inactive |
| 8   | davezuko/react-redux-starter-kit| 6138   | 156    | inactive |
| 9   | div/ngxjs/ngxfs          | 27416  | 526    | active  |
| 10  | django-extensions/django-extensions| 5716   | 528    | active  |
| 11  | ecreamy/spectacle        | 4198   | 456    | inactive |
| 12  | eventmachine/eventmachine| 4565   | 526    | active  |
| 13  | facebookincubator/create-react-app| 28207  | 79     | inactive |
| 14  | i3/i3                    | 17286  | 532    | active  |
| 15  | JakeWharton/ActionBarSherlock| 4860   | 341    | inactive |
| 16  | jcjjohnson/neutral-style| 3328   | 191    | inactive |
| 17  | jekyll/jekyll            | 46534  | 530    | active  |
| 18  | jsquisites/javascripts    | 10390  | 305    | inactive |
| 19  | jruby/jruby              | 28347  | 527    | active  |
| 20  | junit-team/junit4        | 2840   | 501    | active  |
| 21  | kevinzhou/PNCart         | 938    | 277    | inactive |
| 22  | mdeigh/acts-as-taggable-on| 3786   | 526    | active  |
| 23  | onecvat/VDocumenter-Xcode| 854    | 231    | inactive |
| 24  | prawnpdf/prawn           | 5161   | 526    | active  |
| 25  | Prinzhorn/skrolr         | 3729   | 299    | inactive |
| 26  | rails/rails              | 196487 | 2578   | active  |
| 27  | redis/redis               | 4000   | 523    | active  |
| 28  | sferik/twitter            | 3934   | 521    | active  |
| 29  | Shopify/liquid           | 6125   | 526    | active  |
| 30  | sinatra/sinatra          | 6598   | 531    | active  |
| 31  | sparklemotion/nokogiri   | 10516  | 530    | active  |
| 32  | thoughtbot/paperclip     | 13025  | 524    | active  |

Table 2: Rules to determine the evolution patterns of a community, e.g., the ith community at time t, \( c_{it} \).

| Pattern | Existing Approach [27] | Our Approach |
|---------|------------------------|--------------|
| **extract** | Community \( c_{it} \) has no matching post community. | \( P^0_{it} \leq 0.05(H^0_{it-1} + \sigma_{it}) \) and \( P^0_{it} \geq 0.95(H^0_{it-1} + \sigma_{it}) \), i.e., split index close to zero and shrink index close to the maximum value. |
| **merge**   | Community \( c_{it} \) has no matching prior community. | \( P^0_{it} \leq 0.05(H^0_{it-1} + \sigma_{it}) \) and \( P^0_{it} \geq 0.95(H^0_{it-1} + \sigma_{it}) \), i.e., merge index close to zero and expand index close to the maximum value. |
| **split**   | Community \( c_{it} \) has at least two communities in its post community set. | \( P^0_{it} > P^0_{it-1} \), i.e., split index greater than shrink index. |
| **shrink**  | Community \( c_{it} \) has only one community in its post community set. | \( P^0_{it} < P^0_{it-1} \), i.e., split index less than shrink index. |
| **merge**   | Community \( c_{it} \) has at least two communities in its prior community set. | \( P^0_{it} > P^0_{it-1} \), i.e., merge index greater than expand index. |
| **expand**  | Community \( c_{it} \) has only one community in its prior community set. | \( P^0_{it} < P^0_{it-1} \), i.e., merge index less than expand index. |

\( H^0_{it} \) is the maximum value of the community index for the ith community at time t. \( \sigma_{it} \) is the standard deviation of the community index for the ith community at time t. The size of a community is the sum of its members' node weights. We also slightly modify the rule for shrink in [27] to compare the size of \( c_{it} \) with its post community to make the detection results comparable to ours.

5.1.2 Experiment Organization. We evaluate the construct validity of the proposed indices with the concurrent and discriminant validity criteria following exiting work and guides [31, 38, 50]. As defined in [31], construct validity is “the extent to which your measure or instrument actually measures what it is theoretically supposed to measure”. In this work, we aim at showing the proposed indices are effective in capturing and quantifying community evolutionary behaviors of DSNs around OSS projects.

First, we adopt the concurrent validity criterion to validate the proposed indices. Concurrent validity assesses the ability of a measure to distinguish between groups it should theoretically be able to distinguish between [31, 50]. In this work, we compare the categorical evolution patterns detected following an existing approach [27] with the patterns detected using the proposed indices.

Next, we evaluate the discriminant validity of the indices which requires the dimensions to be independent or orthogonal to each other. Based on the properties of the indices discussed in Sec. 4.2.2, and by further evaluating the discriminant validity of the measures, we aim at showing that the proposed indices are capable of measuring different aspects of community evolution.

Unfortunately, we are unable to assess the convergent validity of the proposed indices because to the best of our knowledge, this is the first work that quantifies the degree of community evolution that include different evolution patterns simultaneously.

Besides construct validity, we also show the usefulness of the proposed indices by studying their correlations with projects’ team productivity measured by the count of commits [11, 63]. Details of the experiments and their results are as follows.

5.2 Concurrent Validity

We evaluate the concurrent validity of the proposed indices by comparing the community evolution pattern detection results of our approach and existing approaches. We adopt the process proposed in [27] to detect community evolution patterns in DSNs which serve as the ‘ground-truth’ for evaluation. More specifically, for consecutive time steps \( t \) and \( t + 1 \), we first calculate the similarity
between each pair of communities detected in the two DSN snapshots. The generation of DSN snapshots and detection of communities are the same as introduced in Sec. 3 to conduct a fair comparison. Slightly different from the original algorithm, the size of a community is defined as the sum of its members’ weights instead of the count of its members. The similarity of two communities is the sum of weights of their common nodes over the smaller value of the two communities’ total weights. We take the threshold of 0.3 proposed in the original work to perform community matching. Based on the matching results, we adopt rules in Table 2 to determine the evolution patterns for each community. We modify the rule for determining community shrink by comparing a community’s size with its matching community in the next step to make the results comparable with ours. With these rules, for each community, we can assign two evolution patterns to it: 1) one to describe how it evolves towards the next step, which can be extinct, split, or shrink; 2) the other to describe how it evolves from the previous step, which includes merge, emerge, and expand. An undefined pattern is added to label cases that are not included in the existing set of rules.

For community evolution pattern detection using the indices proposed in this paper, we simply compare the index values to determine the patterns of community split, shrink, merge, and expand as shown in Table 2. For a community, say community $c_{t,i}$, to extinct, the ideal case is that its split index is zero ($I_{c_{t,i}}^s = 0$), and its shrink index takes the maximum value ($I_{c_{t,i}}^b = H_{c_{t,i}}^{t+1} + \sigma_n$) as discussed in Sec. 4.2.4. However, this condition is too stringent to effectively detecting community extinction in real-world data. As a result, we relax the condition as shown in Table 2 that label a community to extinct when its split index is close to zero, and its shrink index is close to the maximum value. The same relaxation also applies to the rule to detect community emerge.

The confusion matrix that compares the ground-truth and our detection results is shown in Fig. 4(a). The overall accuracy of our approach involving all six patterns and the undefined pattern is 89.2%. Because the undefined pattern is not clearly defined in existing work [27, 46], we also evaluate the accuracy by removing the undefined label (i.e., by removing the last column and row in the confusion matrix). Without the undefined label, the overall accuracy is 84.7%. From Fig. 4(a), we can observe that 52% of the shrink patterns are classified as split with our approach. And 45.8% of the expand cases are detected as merge. This is because there is a threshold of 0.3 to filter out matching but less similar communities in the existing approach. As a result, the ground-truth is biased to shrink and expand compared to our approach which does not perform filtering. Since there is no evidence showing that the threshold of 0.3 is the optimal one, we repeat the test by setting a threshold of 0.1 to increase the chance of community matching when obtaining the ground-truth. As shown in Fig. 4(b), we can observe that the detection accuracy with and without the undefined pattern increases to 92.7% and 94.1%, respectively.

The above results suggest that, with a simple set of rules, the proposed indices can accurately distinguish between different patterns of community evolution which are consistent with the results obtained following the existing approach [27]. We conclude that the proposed indices for community evolution are valid with respect to the concurrent validity criterion.

### 5.3 Discriminant Validity

We calculate the Spearman’s Correlation Coefficients of each pair of the indices given a project. Spearman’s Correlation Coefficients are selected over the Pearson’s Correlation Coefficients because we find that the indices do not always follow normal distributions. We adopt the rule of thumb in [2, 18] to interpret the correlation coefficients, i.e., the $r$ values, as summarized in Table 3.

| Levels | Correlation Coefficients ($r$) | Symbols |
|--------|--------------------------------|---------|
| Perfect | $r = 1$ or $r = -1$ | P |
| Strong  | $0.7 \leq r < 1$ or $-1 < r \leq -0.7$ | S |
| Moderate | $0.4 \leq r < 0.7$ or $-0.7 < r \leq -0.4$ | M |
| Weak    | $0.1 \leq r < 0.4$ or $-0.4 < r \leq -0.1$ | W |
| No      | $0 \leq r < 0.1$ or $-0.1 < r \leq 0$ | N |

It is worth noting that a strong correlation alone does not invalidate the design of the proposed measure because the correlations between community evolution indices are largely determined by the members’ activities in real-world OSS repositories. The phenomenon we would like to observe in this study is that whether there are occasions that the indices are weakly correlated in the real-world.
Table 4: Pairwise correlation of the aggregated indices (Spearman’s Correlation Coefficient). Project No. follow Table 1. Notations follow Table 3. The significance levels are shown without the p values to make the table concise.

| Project No | split-shrink | split-expand | shrink-expand | split-merge | shrink-merge | merge-expand |
|------------|--------------|--------------|---------------|-------------|--------------|--------------|
| 1          | 0.051 N      | 0.41 M       | -0.13 W      | 0.51 N      | 0.34 M       | 0.22 W       |
| 2          | 0.021 W      | 0.35 S       | -0.11 M      | 0.25 W      | 0.20 M       | 0.11 M       |
| 3          | 0.071 W      | 0.32 W       | 0.14 W       | 0.11 W      | 0.61 N      | 0.41 M       |
| 4          | 0.101 W      | 0.39 W       | 0.16 W       | 0.10 N      | 0.21 W      | 0.11 W       |
| 5          | 0.051 N      | 0.49 M       | 0.24 W       | 0.06 N      | 0.29 W      | 0.31 W       |
| 6          | 0.17 W       | 0.59 M       | 0.15 W       | 0.28 W      | 0.31 W      | 0.26 W       |
| 7          | 0.52 W       | 0.44 M       | 0.15 M       | 0.35 W      | 0.39 W      | 0.56 M       |
| 8          | 0.061 N      | 0.41 M       | 0.19 W       | 0.16 W      | 0.12 W      | 0.29 W       |
| 9          | 0.071 N      | 0.32 M       | 0.061 M      | -0.01 N    | 0.01 N      | 0.17 N       |
| 10         | 0.021 N      | 0.32 W       | 0.09 W       | 0.09 N      | 0.15 W      | 0.41 M       |
| 11         | 0.091 N      | 0.32 W       | 0.41 M       | -0.071 N    | 0.15 W      | 0.41 M       |
| 12         | 0.021 N      | 0.36 M       | -0.18 M      | 0.07 N      | 0.18 W      | 0.38 W       |
| 13         | 0.20 W       | 0.24 W       | -0.22 W      | 0.01 N      | 0.41 N      | -0.49 W      |
| 14         | 0.65 M       | 0.76 (S)     | 0.42 M       | 0.63 M      | 0.64 M      | 0.64 M       |
| 15         | 0.48 M       | 0.79 (S)     | 0.48 M       | 0.42 M      | 0.39 W      | 0.43 M       |
| 16         | 0.11 W       | 0.64 M       | 0.18 M       | 0.15 W      | 0.24 W      | 0.41 M       |
| 17         | 0.031 N      | 0.40 M       | 0.12 W       | 0.07 N      | 0.12 W      | 0.23 W       |
| 18         | 0.32 W       | 0.85 M       | 0.12 M       | 0.40 M      | 0.52 M      | 0.51 M       |
| 19         | 0.11 W       | 0.67 M       | 0.14 W       | 0.17 W      | 0.21 W      | 0.21 W       |
| 20         | 0.35 W       | 0.79 (S)     | 0.43 M       | 0.54 W      | 0.51 M      | 0.51 M       |
| 21         | 0.071 N      | 0.50 M       | 0.34 W       | 0.061 N    | 0.39 N      | 0.37 W       |
| 22         | 0.091 N      | 0.43 M       | 0.24 W       | -0.11 W    | 0.23 W      | 0.31 W       |
| 23         | 0.27 W       | 0.74 (S)     | 0.45 M       | 0.25 W      | 0.12 W      | 0.37 W       |
| 24         | 0.021 N      | 0.48 M       | 0.18 W       | 0.06 N      | 0.07 N      | 0.07 N       |
| 25         | 0.22 W       | 0.25 W       | 0.16 W       | 0.27 W      | 0.33 M      | 0.34 M       |
| 26         | 0.16 W       | 0.84 M       | 0.21 W       | 0.25 W      | 0.24 W      | 0.24 W       |
| 27         | 0.00 (N)     | 0.56 M       | 0.41 M       | -0.01 N    | 0.10 N      | 0.38 W       |
| 28         | 0.03 (N)     | 0.44 M       | 0.32 W       | 0.03 N      | 0.14 W      | 0.34 W       |
| 29         | 0.28 W       | 0.77 (S)     | 0.49 M       | 0.26 W      | 0.19 W      | 0.25 M       |
| 30         | 0.06 (N)     | 0.50 M       | 0.31 W       | 0.12 (N)   | 0.01 N      | 0.37 W       |
| 31         | 0.071 N      | 0.37 W       | 0.13 W       | 0.12 W      | 0.21 W      | 0.29 W       |
| 32         | 0.011 N      | 0.35 M       | 0.15 W       | 0.03 M      | 0.15 M      | 0.22 W       |
| Overall    | 0.32 M       | 0.64 M       | 0.17 W       | 0.22 W      | 0.27 W      | 0.44 M       |

Combing the results in this study and the properties of the indices as discussed in Sec. 4, we aim at showing that the proposed indices are measuring different aspects of community evolution.

The Spearman’s Correlation Coefficients together with their significance levels for all the projects are listed in Table 4. We also append the results of mixing all projects’ data in the last row of the table. It can be observed that among the thirty two projects involved in our study, there are four projects, i.e., project 4, 10, 13, 31, show a weak or no correlation between all pairs of the indices, which cover both active and inactive projects (refer to Table 1). And over half of the projects only show a moderate correlation between a single pair of indices. As summarized in the last row of Table 4, most of the pairs show a weak, but significant correlation with r values below 0.4. The merge and expand indices show a moderate correlation with a r value of 0.4, which is at the borderline between a moderate and weak correlation.

An interesting discovery made from the results is that the split and merge indices show a moderate correlation for most of projects, and even a strong correlation for five projects. The overall correlation coefficient between split and merge is 0.68, suggesting a positive, moderate correlation which is statistically significant. Recall

The way we calculate community split and merge, and the results of concurrent validity presented above, it is clear that the two indices are capable of measuring different patterns of community evolution. The correlation between split and merge is a natural result emerges from the issue and pull-request discussion activities of the community members. A possible explanation to this positive correlation is that real-world communities often regroup themselves in OSS projects studied in this paper. It is our future work to inspect more patterns of community evolution based on the proposed indices.

Table 5: Correlation between log transformed community evolution indices and team productivity. The later is measured by log(count of commits + 0.5). Regression analysis with linear mixed-effects models.

| Coefs (Errors) | Sum Sq. |
|----------------|---------|
| (Intercept)    | 2.9307  | 0.2287*** |
| log(split+0.5)  | 1.1428  | 0.04734*** |
| log(shrink+0.5) | 0.2593  | 0.07427*** |
| log(merge+0.5) | 1.0495  | 0.04620*** |
| log(expand+0.5) | -0.9238 | 0.08525*** |

| R²_m | 0.139 | R²_L | 0.455 |
|------|-------|------|-------|
| Note | 13018 | groups | project name = 32 |

...
and expand. A possible explanation to the effects of split and merge is that members of communities with high degrees of split and merge are actively regrouping themselves to join issue and pull_request discussions which results in productive teams. And a potential reason for the negative effects of expand is that team performance are known to be influenced by many factors beyond team size, e.g., team diversity [51, 63], developer synchronization [69], and etc. Growth in team size potentially increases the communication and coordination costs which reduce the working efficiency. The above results, together with the results in Sec. 5.3, inspire us to explore different combinations of the indices as meaningful patterns, and to conduct more studies to understand the effects of community evolution in team productivity and performance in our future work.

In summary, the regression model achieves an accuracy (i.e., the correlation coefficient between the actual and the predicted value of the outcome [35]) of 0.718, suggesting that the aggregated indices of community evolution are useful predictors of team productivity. A potential threat to the robustness of the results is that non-normality and heteroscedasticity of the residuals are detected. We consult the literature [21, 32, 53], and conclude that the above results are trustworthy after carefully inspecting the Q-Q diagram.

6 THREATS TO VALIDITY

We present a discussion the threats to validity as follows.

Internal Validity. We conduct a data-driven approach to reveal the correlation between community evolution and OSS development such as team productivity. However, there are many other factors that are known to affect or characterize the operation of OSS teams [11, 54]. As a result, the conclusions made in this paper do not fully explain how OSS teams behave or imply a causal relationship. Instead, the proposed indices should be used as useful supplements to existing measures of DSN and community evolution.

External Validity. We select thirty-two projects that cover various topics and status in our study. However, the collection of projects is small compared to the number of projects hosted on GitHub and other platforms. The conclusions made in this paper may not generalize well to other projects. It is our future work to include more projects in our studies.

Construct Validity. The indices are designed on top of the well-established theory of information entropy to measure community evolution with clear properties about how their values respond to parameter changes. We evaluate the construct validity of the proposed indices in terms of concurrent and discriminant validity. We plan to seek more criteria to validate our approach in the future.

7 RELATED WORK

From a general perspective, this work belongs to the series of research about the social aspects of OSS development started since the beginning of the OSS movement [24, 37, 60]. In this section, we introduce work closely related to this paper.

OSS projects are developed and maintained by the collaborative efforts of OSS contributors over channels such as mailing lists [7], and on "social coding platforms" [17, 61, 67] such as GitHub and GitLab. The interactions of OSS contributors naturally generate social networks around OSS projects, which is refer to as the developer social networks (DSNs) in existing literature [27]. Different types of DSNs—including communication [3, 5, 7, 16, 20, 25, 27, 44, 48, 66], collaboration [24, 30, 56, 57, 65], and hybrid [47, 59] networks—have been extracted by mining developer mailing lists and OSS repositories to study the properties and evolution of DSNs. The extracted DSNs are often modeled as graphs [3, 27]. In this work, weighted, undirected graphs are used to model DSNs that reveal the communication structures of OSS contributors, based on which a series of indices are proposed to quantify the evolution of communities inside DSNs. It should be noted that the proposed approach can also be applied to DSNs of other types.

Technologies for social network analysis (SNA) are applied to DSNs to understand the properties, the structures, and the evolution of DSNs. Readers can refer to [12, 26, 54] for comprehensive surveys for social network and community analysis in DSNs. Different from work that performs analysis to static DSNs [7, 16, 29, 59, 68], this work focuses on understanding the dynamic evolution of DSNs over time. Many studies about DSN evolution are conducted by analyzing the changes and trend in series of DSN snapshots take over time [3, 25, 27, 28, 30, 39, 44, 65]. Network metrics, such as network diameter, shortest path between nodes, node betweenness, modularity, hierarchy, centrality, clustering coefficient, and etc, are adopted in existing work [3, 12, 25, 28, 30, 49] to assess the properties of DSNs. There are also metrics developed exclusively to measure software communities [65]. Based on these metrics, statistical and data mining technologies are applied to understand the evolution of DSNs [28, 58]. This work is related to the above work in that we propose indices to quantify the evolution of DSN communities which can support further analysis including but not limited to regression analysis, and pattern detection as shown in our experiments. Different from the above work that takes the network perspective, this work takes the community perspective [3] and analyzes the evolutionary behaviors of communities in DSNs.

The patterns adopted in this paper that describe the evolutionary behaviors of DSN communities are originally proposed in [46], which involve six patterns as introduced in the rest of the paper. These patterns are shown to be promising in describing the evolution of communities in various types of social networks such as co-authorship, phone call, and blog networks [34, 46]. In [27], the authors adopt these patterns to characterize community evolution in OSS projects including Firefox, and Bugzilla. Similar patterns are also adopted in [3] to describe the evolution of DSNs at the community level. This work is closely related to the above work in that we also adopt the patterns as the basis to describe community evolution. Different from these approaches that assign a unique, nominal pattern to a matched pair of communities in consecutive time steps, we quantify the degree of different patterns which can occur simultaneously during the evolution of the communities. Other differences of this work from the above work include: 1) community emerge and extinct are defined by combinations of the indices. As a result, we only include indices for four patterns as introduced in the paper; and 2) we do not require to match communities detected
in different time steps to recognize them as a single evolving community, which is a challenging task [46] that can lead to biases if not treated with caution.

Finally, we note that the community evolution patterns studied in this work are about the patterns introduced above, which are different from the general patterns of OSS evolution in existing work, which include studies that characterize the process of preferential attachment in OSS developers when forming DSNs [66], or the stages of OSS project evolution [40], and etc.

8 CONCLUSION

In this paper, we present four entropy-based indices to measure the degree of community evolutionary behaviors with respect to patterns including split, shrink, merge, and expand to understand the evolution of DSNs around OSS projects. This work bridges the gap between pattern-based approaches which can provide meaningful insights about the evolution of DSNs from the community perspective, and metric-based social network analysis which can support comprehensive and quantitative analysis. The construct validity of the proposed indices are evaluated on a real-world data set about the issue and pull-request discussions in thirty-two OSS projects. The results suggest the indices can accurately capture different patterns of community evolution by achieving a detection accuracy of 94.1% compared to existing work [27], and show weak and moderate correlations. The results also show that the indices can accurately predict OSS team productivity after log transformation with linear mixed-effects models.

The proposed indices can be used and extended in many ways to support future studies. For example, we can use the indices to assess and predict the behavior and status of OSS teams and projects, e.g., to assess the health of OSS teams, and to predict OSS projects’ sustainability [50]. More patterns of community evolution can be developed based on combinations of the indices, e.g., community regroup as discussed in Sec. 5.3, following the way we define community emerge and extinct. We can also apply the proposed framework to other types of DSNs, e.g., collaboration networks [30], to understand their evolution.

REFERENCES

[1] Abdelnourm A Alfit, Jenny B Kotlerman, Susan L Ettner, and Marie Cowan. 2007. Methods for improving regression analysis for skewed continuous or censored responses. Anna. Rev Public Health 28 (2007), 95–111.
[2] Haldun Akoglu. 2018. User’s guide to correlation coefficients. Turkish journal of emergency medicine 18, 3 (2018), 91–93.
[3] Mohamed Abdelrahman Aljemabi and Zhongjie Wang. 2018. Empirical study on the evolution of developer social networks. IEEE Access 6 (2018), 51049–51060.
[4] Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. 2009. An event-based framework for characterizing the evolutionary behavior of interaction graphs. ACM Transactions on Knowledge Discovery from Data (TKDD) 3, 4 (2009), 1–36.
[5] Nicolas Bettenburg. 2011. Mining development repositories to study the impact of collaboration on software systems. In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. 376–379.
[6] Stefanie Betz, Samuel Fricker, Andrew Moss, Wasif Afzal, Mikel Svalbargren, Claes Wohlin, Jürgen Börierl, Tony Gorschek, et al. 2013. An evolutionary perspective on socio-technical congruence: The rubber band effect. In 2013 3rd International Workshop on Replication in Empirical Software Engineering Research. IEEE, 15–24.
[7] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social networks. In Proceedings of the 2006 international workshop on Mining software repositories. 137–143.
[8] Christian Bird, David Pattison, Raissa D’Souza, Vladimir Filkov, and Premkumar Devanbu. 2008. Latent social structure in open source projects. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering 24–35.
[9] Thomas Bock, Angelika Schmid, and Sven Apel. 2021. Measuring and Modeling Group Dynamics in Open-Source Software Development: A Tensor Decomposition Approach. ACM Transactions on Software Engineering and Methodology (TOSM) 31, 2 (2021), 1–50.
[10] Prateek Brodka, Stanisław Saganowski, and Przemysław Kazienko. 2013. GED: the method for group evolution discovery in social networks. Social Network Analysis and Mining 3, 1 (2013), 1–14.
[11] Gemma Catolino, Fabio Palomba, Damian A Tamburri, Alexandre Serebrenik, and Filomena Ferrucci. 2019. Gender diversity and women in software teams: How do they affect community smells? In 2019 IEEE/ACM 41st International Conference on Software Engineering; Software Engineering in Industry (ICSE-SEI). IEEE, 11–20.
[12] Tannmoy Chakraborty, Ayushi Dalmai, Animesh Mukherjee, and Niloy Ganguly. 2017. Metrics for community analysis: A survey. ACM Computing Surveys (CSUR) 50, 4 (2017), 1–37.
[13] Jallot Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 186–196.
[14] Patricia Cohen, Stephen G West, and Leona S Aiken. 2014. Applied multiple regression/correlation analysis for the behavioral sciences. Psychology press.
[15] Mylvin E Conway. 1968. How do committees invent. Datamation 14, 4 (1968), 28–31.
[16] Kevin Crowston and James Howison. 2005. The social structure of free and open source software development. First Monday (2005).
[17] Laura Dabbish, Colleen Stuart, Jason Tasy, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In Proceedings of the ACM 2012 conference on computer supported cooperative work. 1277–1286.
[18] Christine P Dancey and John Resdy. 2007. Statistics without maths for psychology: Pearson education.
[19] Nicolas Ducheneaut. 2005. Socialization in an open source software community: A socio-technical analysis. Computer Supported Cooperative Work (CSCW) 14, 4 (2005), 323–368.
[20] Kate Ehrlich and Marcelo Catálado. 2012. All-for-one and one-for-all? A multi-level analysis of communication patterns and individual performance in geographically distributed software development. In Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work. 945–954.
[21] Andrew Gelman and Jennifer Hill. 2006. Data analysis using regression and midelev/ hierarchical models. Cambridge university press.
[22] Georgios Goutsos and Diomidis Spinellis. 2012. GHTorrent: GitHub’s data from a firehose. In 2012 9th IEEE Working Conference on Mining Software Repositories (MSR). IEEE, 12–21.
[23] Derek Greene, Donal Doyle, and Padraig Cunningham. 2010. Tracking the evolution of communities in dynamic social networks. In 2010 international conference on advances in social networks analysis and mining. IEEE, 176–183.
[24] Junggil Hahn, Jae Yun Moon, and Chen Zhang. 2008. Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. Information Systems Research 19, 3 (2008), 369–391.
[25] Anna Hannemann and Ralf Klamma. 2013. Community dynamics in open source software projects: Aging and social reshaping. In IFIP International Conference on Open Source Systems. Springer, 80–96.
[26] Steffen Herbold, Aynur Asumaihah, Fabian Trautsch, and Jens Grabowski. 2021. A systematic mapping study of developer social network research. Journal of Systems and Software 171 (2021), 118082.
[27] Qiaona Hong, Sung hun Kim, Shing Chi Cheung, and Christian Bird. 2011. Understanding a developer social network and its evolution. In 2011 27th IEEE international conference on software maintenance (ICSM). IEEE, 323–332.
[28] Hao-Yun Huang, Qize Le, and Jitesh H Panchal. 2011. Analysis of the structure and evolution of an open-source community. Journal of Computing and Information Science in Engineering 11, 3 (2011).
[29] Carlos Jensen, Scott King, and Victor Kuechler. 2011. Joining free/open source software communities: An analysis of newcomers’ first interactions on project mailing lists. In 2011 44th Hawaii international conference on system sciences. IEEE, 1–10.
[30] Mitchell Joblin. 2017. Structural and Evolutionary Analysis of Developer Networks. Ph. D. Dissertation. Universität Passau.
[31] Arora Kanika et al. 2015. Research methods: The essential knowledge base. Cengage learning.
[32] Ulrich Kneuf and Wolfgang Forstmeier. 2021. Violating the normality assumption may be the lesser of two evils. Behavior Research Methods 53, 6 (2021), 2576–2590.
[33] Zhiqiang Li, Yue Yu, Minghui Zhou, Tao Wang, Gang Yin, Long Lan, and Huaimin Wang. 2020. Redundancy, context, and preference: An empirical study of duplicate pull requests in OSS projects. IEEE Transactions on Software Engineering (2020).
