Searching for "Stability" in Fluidity: A Routine-based View of Open Source Software Development Process

Research-in-Progress

Xinyu Li
Case Western Reserve University
Cleveland, USA
xinyu@case.edu

Youngjin Yoo
Case Western Reserve University
Cleveland, USA
youngjin@case.edu

Zhewei Zhang
Brunel University London
Uxbridge, UK
Zhewei.zhang@brunel.ac.uk

Abstract

Open source software communities are fluid form of organizing where the constituent members and the interactions among the members are continuously and rapidly changing over time. However, certain stability such as identity and norms maintains in spite of the dynamic nature of the communities. This paper seeks to resolve the tension between the fluidity and stability that co-present in open source community through a lens of organizational routine, and proposes a two-step approach to capture routines in the context of open source software development process. Using 200 open source projects on GitHub as a preliminary analysis, this research-in-progress demonstrates the capability of the method while expecting to unleash its whole potential in a future study.

Keywords: Open source software development, open source software community, organizational routine, sequence analysis, clustering analysis

Introduction

Open Source Software (OSS) development projects are online communities that involve voluntary and heterogeneous software developers who collaborate and exchange knowledge as they develop and enhance software products (Hippel and Krogh 2003). OSS communities represent a fluid form of organizing, where the constituent members, the interactions among the members, and their outputs are constantly changing over time (Faraj et al. 2011). In an OSS community, for example, membership of the community is always in flux as new developers continuously join the community while existing members may become inactive at a time or leave the community. The leadership may also shift over time as members come in and out of the community and the focus of the project may evolve over time. New members may bring new problems that have not been previously recognized by the existing members; they might also bring new ideas or expertise that offers new way of leveraging existing codebase to build different applications. As new members join the community, the relationship among developers can change over time. Finally, as members constantly update codebase, the output from the community is also always in flux. In a way, the fluidity of the open source communities provides the social basis of the generativity that we often observe from such communities, that produces surprising and unpredictable innovations (Boland et al. 2007).

While these new forms of fluid organizing represent the dynamic nature of OSS communities, they do maintain certain stability. It retains distinguishable identity, membership, and norms. In the organizational science literature, organizational routine has been extensively studied in stable organizational settings as it provides a way to understand organizational design, management and change, and to predict the outcomes. However, limited scholarly endeavors in this regard have been extended to more fluid organizing settings (e.g. OSS communities). This paucity is probably due to the difficulty to capture the spirit of stability of routines while accounting the very nature of fluidity of those settings with the concepts and methods designed for relatively stationary organizational routines.

To this end, in this research in progress paper we propose a way to conceptualize organizational routine in OSS development communities, which will facilitate the comprehension of the relationship among collaboration, governance and performance in OSS communities, and other organizations in fluid forms in a broad sense. Specifically, we seek to answer the following research questions:

1) How can we measure organizational routines in an OSS development project where heterogeneous contributors with uncoordinated participation create a fluid community?

2) How does the fluidity of OSS communities change the stability of routines?

To address these questions, we propose a routine-based approach as a way to approach such a paradox. In particular, we draw on sequence analysis techniques together with clustering analysis techniques to identify a set of routines that are enacted in open source communities. An important theoretical and methodological novelty that we present in this work is a way to identify variations of routines and their clusters as a way of resolving the tension between the stability and fluidity that co-present in these communities.

The paper is organized as follows: we first discuss prior studies on the fluid organizing of OSS community and stable organizational routines. Facing the tension between fluidity and stability, we propose a two-step approach which involves a sequence analysis using latent motif mining in the first step to identify implicit patterns in developers' activities, and a hierarchical clustering analysis in the second step to group those patterns into related routine clusters. We then apply the proposed method to conduct a preliminary analysis on 200 comparable OSS projects on GitHub. While the preliminary study does not unfold the entire potential of this work, we make some meaningful discussion on findings produced by this method. Possible extension, expected contribution and challenging issues of this research are discussed at the end of the paper.

A dynamic view of OSS community

The emergence of open source software and other types of open source innovation communities has drawn substantial attentions in the literature. Scholars have shown extensive interests in exploring the factors that motivate voluntary contributions in the communities, such as factors based on self-interest (Hippel and Krogh 2003; Lakhani and von Hippel 2003; Lerner and Tirole 2002), identity (Bagozzi and Dholakia 2006; Stewart and Gosain 2006), and social capital (Hahn et al. 2008). The literature provides insights into various structural mechanisms, such as network externality (Bonaccorsi and Rossi 2003; Grewal et al. 2006; Singh et al. 2011a) and social relationship (Bergquist and Ljungberg 2001), that bring success to an OSS community and sustain that achievement. Another research stream on OSS community discusses the coordination mechanisms that support knowledge exchange among members that are dispersed in location and time (Howison and Crowston 2014; Ovaska et al. 2003; Singh and Tan 2010). The literature also studies the principles of self-governance in OSS communities, such as authority (O'Mahony and Ferraro 2007), and roles and rules (Fleming and Waguespack 2007).

While there are substantial body of literature on OSS community, little attention has been given to its dynamic aspect (Faraj and Johnson 2011; Oh and Jeon 2007; Singh et al. 2011b), exploring OSS community as a system that continually changes over time. Faraj et al. (2011, p. 1225) conceptualizes online communities (including OSS communities) as "fluid organizational objects that are simultaneously morphing and yet retaining a recognizable shape" and highlights the importance to look at the dynamic changes in such fluid organizational objects. The ever-changing nature of OSS communities is embodied in the continuous changes in participated developers (difference in actors, or change in the attributes of the same developers) and the continuous evolutions of artifacts (software). At every moment, this flux

impacts interactions among the members, boundaries and norms of the communities, and foci and goals of the OSS projects. For example, it may be trivial from a non-dynamic point of view to see a community member upload a piece of code after being inactive after a week. However, this action implies that the developer may bring new knowledge acquired in the week to the community, and thus potentially changes many characteristics of the community as a result of impacts and turbulence from that new knowledge. This dynamic perspective is consistent with the sense of the ancient Greek philosopher Heraclitus' assertion that no one ever steps in the same river twice.

One may argue that all forms of organizations change in an evolutionary sense. Indeed, even the organizations with highly stable settings are actually changing over time (as supported by routine-based literature we will discuss next), but OSS communities are particularly featured by constant, rapid flowing and shape-shifting (Faraj et al. 2011), as one may think about stones versus water as an analogy. Therefore, a challenging problem raised is how to understand the underlying structures of OSS communities while the structures are changing all the time. In the next section, we will address this paradox by extending the organizational routine literature.

A routine-based view of OSS process

Organizational routine is used as a lens to study organizational performance, knowledge collaboration and other aspects of an organization. Following Feldman and Pentland's (2003, p. 95) definition, organizational routine can be characterized as "repetitive, recognizable patterns of interdependent actions, carried out by multiple actors". However, the repetitive or recurrent nature of routine does not imply that each repetition of a routine is identical to others, as each occurrence has a unique combination of actors, time, and context. Organizational science literature has discussed the tension between the stability nature and dynamics nature of routines (Cohen 2007; Farjoun 2010). Pentland et al. (2011) empirically investigates this issue using data on invoice processing, a highly structured, repetitive, and institutional procedure, and finds out that patterns of routines change significantly over time. As an attempt to theoretically account for this tension, organizational routine is distinguished into an ostensive layer and a performative layer (Feldman and Pentland 2003). The ostensive layer of a routine is its structure, and thus represents the stable aspect of the routine. This part of routines is stored as actors' procedural memory, which closely links to cognitive activities such as identity and norms (Cohen and Bacdayan 1994). The performative layer modifies a routine by embodying specific elements (i.e. actors, actions, time and place), and thus is the source of flexibility of the routine. We believe a routine-based view is a nature fit to account for the co-presenting of fluidity and stability in OSS development. However, given the dynamic and unstructured nature of the OSS development process, we expect to see high variants from the performative layer of routines.

One common method scholars have proposed to measure routines and their variety is to use sequence analysis technique (Gaskin et al. 2014; Gaskin et al. 2012; Pentland 2003; Pentland et al. 2010), which has also been applied to other social science research (Abbott and Hrycak 1990; Sabherwal and Robey 1993). In particular, Pentland (2003) proposes that sequences of actions can be directly compared to each other to obtain the "distance" within each pair of sequences, representing how dissimilar those two sequences are, or they can be all indirectly compared to a completely random sequence by computing how the transition matrices of real sequences are different from that of the "null" sequence. A requirement of such comparison is a finite sequence of action with a clear beginning and end that defines an occurrence of a routine, or called a routine enactment. However, in the case of OSS development, occurrences of routine are usually implicit and non-threaded due to the unstructured and generative nature of OSS development process. Typically, as the community members come and go, an OSS project involves heterogeneous developers with diverse expertise who work in their own way in different time and place. Roles of developers, as well as relations and interactions among developers vary accordingly over time. Moreover, for most of the projects, focus points change every time a critical issue raises, making OSS development an ongoing, evolving process that lasts for a long period of time (relative to invoice processing, for example). As a consequence, we are not able to apply the methodology to model routines without firstly identifying occurrences of routine.

There are various techniques and tools for pattern recognition, including Hidden Markov models (Elliott et al. 1995), process mining (Aalst et al. 2002), heuristic mining (Neamsirorat and Premchaiswadi 2015), and sequential pattern mining (Mabroukeh and Ezeife 2010; van Dongen et al. 2005). In this work, we

use a sub-sequence mining technique (Burgin and Ritschard 2014; Ritschard et al. 2013) to uncover the latent patterns (occurrences of routine) in a series of sequences of OSS development actions, as we will discuss in detail in the next section.

Methodology

To answer our research questions, we collected the data from Github.com. As one of the largest online hosting repositories for open source software, GitHub stores all user activities on the website as digital trace. Table 1 shows some examples[1] of the digital trace data on GitHub. Each entry of the digital trace describes a project, an actor, a pre-defined action type determined by GitHub, and the time that action is performed. There are a number of action types[2], among which twelves are pertinent to software development process while others are related to team management, social relationship establishment, and other functionalities of the website. We took the twelves development-related action types to form a basic lexicon (Pentland 1995; Pentland 2003). In addition, we included role information of actors by distinguishing between core developer and peripheral developers (Crowston et al. 2006; Setia et al. 2012) based on the Bradford's Law of Scatter[3] (Bradford 1950), doubling the size of the lexicon to twenty-four "actor type-action type" combinations. Based on the lexicon, we constructed a sequence of actions for each project, and thus each sequence contains all the development activities of one single project, ordered by time, in a period of time.

Project	Actor	Action	Time
jquery.videoBG	Difrakt	Issue Open	7/25/2012 17:09:21
jquery.videoBG	sydlawrence	Issue Close	7/25/2012 22:58:01
jquery.videoBG	sydlawrence	Issue Comment	7/25/2012 22:58:01
jquery.videoBG	Difrakt	Issue Comment	7/26/2012 8:19:26
jquery.videoBG	eschleeper	Issue Open	7/30/2012 19:26:31
jquery.videoBG	cerisier	Push	10/25/2012 19:51:14

Table 1. Examples of Digital Trace Data on GitHub

We adopted a two-step approach to identify routines in OSS development process. In the first step, we used a sub-sequence mining technique to detect frequently repeated sub-sequences from a bunch of sequences of actions created from the time-stamped digital trace data. This technique allows us to recognize all of the identical pieces of sequence that are co-possessed by a given percentage of the bunch of sequences of actions. In a way, this approach is essentially following the notion that routines are observed as repetitive or recurrent patterns (Pentland et al. 2011). Therefore, each of the sub-sequences detected is conceptualized as a routine, and each instance of that sub-sequence in a sequence of actions is an occurrence of that routine. This step was performed using TraMineR package in R (Burgin and Ritschard 2014; Ritschard et al. 2013).

[1] Each activity log contains more than 110 data fields. We only show 4 relevant fields here for demonstration.

[2] https://developer.github.com/v3/activity/events/types/

[3] Bradford's Law of Scatter is a bibliometric law describing how the literature on a particular subject is distributed among journal literature. It states the fact that a few journals would have many articles on one subject, while only a few articles on that subject could be found in many other journals. The ubiquity of this distribution makes it almost a universal law for human behaviors in which prior success leads to further success. In the context of OSS development, success in prior contribution to a community encourages the contributors to make more commitment, making them the core developers of that community. Based on the mathematical property of this distribution, a core developer group is defined as those who are top contributors and who jointly contribute at least 1/3 of the total contribution (Crowston et al. 2006), and peripheral developer group otherwise.

In the second step, we took advantage of a clustering analysis to group those sub-sequences detected into several routine clusters. The technique we applied is a hierarchical agglomerative clustering method in the sense that it compares the "distance" within each pair of the sub-sequences, and combines the two sub-sequences with the least distance into a branch, which are themselves fused in a future step. This principle is repeated in every algorithmic step of the analysis until the last two branches are combined. In this sense, this analysis is following the spirit of Pentland (2003) to measure sequential variety in process. In the end, we are able to identify routine clusters and trace different variations of routine in each cluster through a dendrogram produced by the clustering analysis.

As a preliminary analysis, we applied this two-step approach to 200 small size JavaScript projects on GitHub. We started with small size projects because small size projects generally involve shorter sequences of actions and fewer participants than medium and large size projects, making this initiation feasible and simple. Because JavaScript is the most popular programming language on GitHub, focusing on JavaScript projects maximizes our ability to compare the development process of projects while retaining the generalizability of our findings.

Altogether, we extracted 89,884 rows of the digital trace data with relevant action types in the period between Feb 2012 and Dec 2014 for the 200 projects. These raw data were converted into 200 sequences of actions with average length of 450 actions (ranged from 4 to 10,827). The 200 projects involves 12,293 unique developers, including 757 developers participating in multiple projects. On average, each developer is responsible for a total of 7.3 actions on one project or across multiple projects. However, the distribution of actions is skewed as only a small number of developers are extremely active while most of the others are inactive (Nonnecke and Preece 2000). Descriptive statistics of the elements in the lexicon for this data set is shown in Table 2.

Actor Type / Action Type	Core Developer		Peripheral Developer		Total	
	Count	Frequency	Count	Frequency	Count	Frequency
Push (PU)	15,484	17.23%	12,749	14.18%	28,233	31.41%
Issue Comment (IC)	14,720	16.38%	18,510	20.59%	33,230	36.97%
Commit Comment (CC)	577	0.64%	694	0.77%	1,271	1.41%
Issue Open (ISO)	957	1.06%	9,311	10.36%	10,268	11.42%
Issue Close (ISC)	5,498	6.12%	1,967	2.19%	7,465	8.31%
Issue Reopen (ISR)	185	0.21%	133	0.15%	318	0.35%
Pull-request Comment (PC)	563	0.63%	634	0.71%	1,197	1.33%
Pull-request Open (PRO)	506	0.56%	3,590	3.99%	4,096	4.56%
Pull-request Close (PRC)	975	1.08%	522	0.58%	1,497	1.67%
Pull-request Merge (PRM)	1,545	1.72%	438	0.49%	1,983	2.21%
Pull-request Reopen (PRR)	26	0.03%	22	0.02%	48	0.05%
Release (RE)	197	0,22%	81	0.09%	278	0.31%
Total	41,233	45.87%	48,651	54.13%	89,884	100%

Table 2. Descriptive Statistics

Preliminary Analysis and Results

In our preliminary analysis, we found 305 sub-sequences from the first step of our approach. The sub-sequences detected are those which present in at least 10%[4] of the 200 sequences of actions, and thus can

[4] We also tried other thresholds but only present the 10% results due to page limit.

be considered frequently repeated sub-sequences or motifs. Each sub-sequence contains different numbers of sequential actions, ranged from one to seven (which means the longest motif has a length of seven). However, an extremely short sub-sequence is not able to produce enough significant social meanings. An extreme example in our case is the ubiquity of the sub-sequence of one single "Push" action across all of the 200 sequences. More importantly, by the construction of sub-sequence, short sub-sequences are usually embraced by their longer extensions. Therefore, we reduced the number of motifs to 80 by removing all the sub-sequences with a length of less than four[5]. Examples of popular motifs are shown in Table 3, in which the number 1 or 2 indicates whether this action is carried out by a core developer or a peripheral developer.

Motif	Count	Cluster	Motif	Count	Cluster
(1PU)-(1PU)-(1PU)-(1PU)	396	1	(2PU)-(2PU)-(2PU)-(2PU)	202	3
(1PU)-(1PU)-(1PU)-(1PU)-(1PU)	181		(2PU)-(2PU)-(2PU)-(2PU)-(2PU)	93	
(1PU)-(1PU)-(1ISC)-(1IC)	159		(2PU)-(2PU)-(2PU)-(2PRO)	62	
(1ISC)-(1IC)-(1ISC)-(1IC)	376	2	(2IC)-(2IC)-(2IC)-(2IC)	108	4
(1IC)-(1ISC)-(1IC)-(1ISC)	232		(1IC)-(2IC)-(1IC)-(2IC)	87	
(1IC)-(1ISC)-(1IC)-(1IC)	158		(2IC)-(1IC)-(2IC)-(1IC)	54	

Table 3. Examples of Motif and Their Routine Clusters

Next, we conducted a hierarchical clustering analysis based on the 80 sub-sequences to discover the similarity and dissimilarity of these routines. However, the analysis itself does not specify an appropriate number of clusters, but provides a dendrogram that illustrates how similar sub-sequences are to one another. To determine the appropriate cluster number, we repeated the clustering based on several randomly reduced set of data, and find out that four clusters is a reasonable and stable solution (Figure 1). The four routine clusters are discussed separately below, and a summary of the discussions is provided at the end of the section (Table 4).

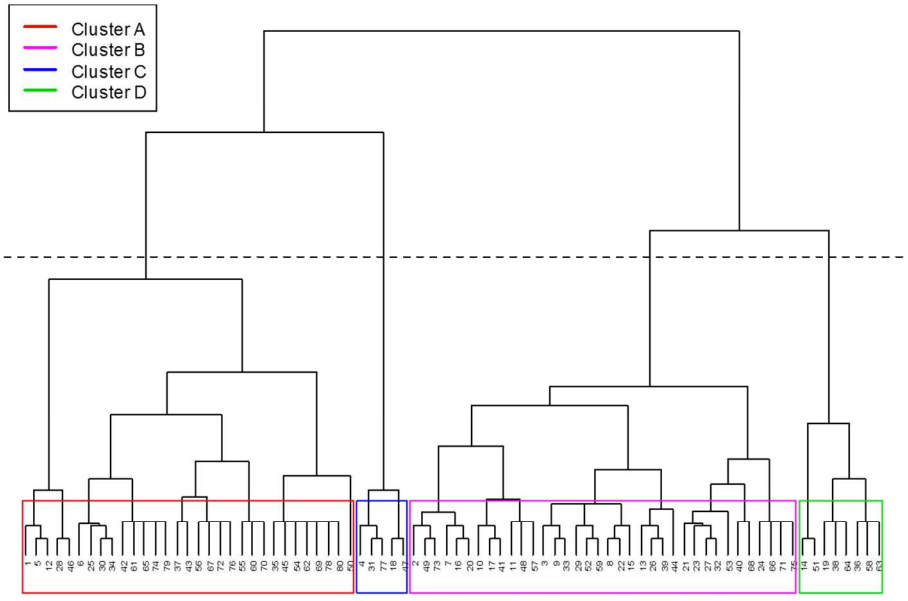

Figure 1. Dendrogram of Routines

[5] We only present this specification due to page limit. Other specifications produce similar results.

The routines in Cluster A can be described as a series of straight Push actions by core developers, blending with other types of action occasionally. In GitHub, "Push" is a privilege of the owners of a repository, and this privilege can be granted to other users as a way to manage the community. A "Push" allows the users to modify codes in a repository directly without approval from others. Thus, the occurrence of this type of routines implies the fact that the development was primarily managed by a single or a small number of developers, and few collaborations are made among developers for solving problems in the development process because there are very few commenting-related actions. This type of routine usually emerges when revisions are made only by a few core developers, making the development highly centralized. It also suggests that the projects are probably relatively uncomplicated since development efforts can be completed by individuals without assistance from others. There are 31 routines in this cluster.

Cluster B features a series of issue-related actions initiated primarily by core developers including Issue Comment, Issue Closing, etc. Some routines in this group also involve "Push" by core developers, as those Push actions may generate new issue topics for developers to discuss and work on. The fact that all 36 routines in this cluster do not contain any actions performed by peripheral developers indicates that these routines also form in an environment of high centrality as those did in Cluster A. However, substantial collaborations through issue comments suggest the difficulty of solving problems individually, which usually happens in complicated projects.

In Cluster C, we see that each variant is dominated by "Push" and accompanied by a few "Pull-request Open" actions. While there are only five routines in the group, this cluster clearly shows a pattern of decentralized working styles as all of the routines include actions only performed by non-core developers. Meanwhile, the pervasion of the privilege of the use of Push action confirms the decentralization of the communities. Finally, in GitHub, a "Pull-request" is triggered when an unauthorized user intends to merge his or her code to a repository, and can be approved or rejected only by owners or collaborators of the project. The presence of "Pull-request" actions indicates that a large number of developers are indeed participating in the development process. However, a lack of collaboration suggests the relatively low complexity of the projects.

Cluster D contains eight routines, each of which is either straight Issue Comment actions by peripheral developers, or a mixture of the action of both types of developers, describing a community-wide collaboration among members through discussion on issue topics. This type of routine usually appears in those OSS communities where developers are facing challenging problems which require intensive collaborations. However, the development process in those communities may be driven by democratic control, and that is why we observe a balanced level of participation between core developers and peripheral developers.

	Cluster A	Cluster B	Cluster C	Cluster D
Routine	31	36	5	8
Centrality	High	High	Low	Low
Complexity	Low	High	Low	High

Table 4. Summary of Four Routine Clusters

Discussion

It is worth noting that routines from the four clusters jointly constitute the development process of OSS projects. In fact, routines from all four clusters can be found in almost the entire preliminary observation group. The only four exceptions are those with extremely short activity sequences in which the diversity of routines is limited. A routine cluster should not be treated as a trait to label an OSS project regardless of how many routines in that cluster are presenting in the project. Instead, the level of each cluster should be examined simultaneously to describe a project. As one of the two illustrative examples (Table 5), Project 1 has a development process that is dominated by routines from Cluster A and C but has a scarcity of routines from Cluster B and D. The presence of this combination suggests that the project is likely to be a relatively complicated project. In Project 2, Cluster C and D are overwhelming, which indicates that the

development of this project may be driven by democratic control, where each revision is based on broad discussion among community members.

	Project 1		Project 2	
	Occurrence	Percentage	Occurrence	Percentage
Cluster A	80	53	53	9.33%
Cluster B	16	123	123	21.65%
Cluster C	54	210	210	36.97%
Cluster D	8	182	182	32.04%
Total	158	568	568	100%

Table 5. Comparing Projects

One may argue, however, that high levels on both Cluster C and D in Project 2 do not indicate a consensus on complexity of the project. Recall that OSS projects are fluid in nature. The features of a project may change over time, which can be reflected by changes in the dominant routine cluster to some extent. We cut the sequence of Project 2 in half based on its time of existence, ran the analysis separately for the sequence from each half of the time, and we observed that Project 2A is dominated by Cluster C routines while Project 2B is dominated by Cluster D. Specifically, the level of Cluster C routines drops dramatically whereas that of Cluster B and D increase significantly from Project 2A to Project 2B. This typically means the complexity of Project 2 rises over time (Table 6).

	Project 2A		Project 2B	
	Occurrence	Percentage	Occurrence	Percentage
Cluster A	19	9.64%	34	9.16%
Cluster B	20	10.15%	103	27.76%
Cluster C	112	56.85%	98	26.42%
Cluster D	36	18.27%	146	39.35%
Total	197	100%	371	100%

Table 6. Comparing Different Periods of a Project

Future Plan

We believe our approach is an appropriate way to understand the stability of routines in the fluid OSS community. Our preliminary analysis also shows the promising outcomes of our approach for understanding the stability in fluidity. Currently, our sample data set is limited to the 200 JavaScript projects. We plan to expand our sample data set by including 2000 small to medium size OSS projects written in JavaScript programming language and that last for at least one year. Once we examine the full data set, we plan to investigate whether we are able to predict project performance (e.g. success/failure rate) using the routine cluster framework discussed above. Additionally, we plan to include social activities of community members to the sequence analysis. However, placing members' social activities in a project-level sequence makes little sense because they are not a part of the development process. Therefore, one direction of our future research is to construct developer-level sequences so that we are able to blend together development activities and social activities.

Expected Contribution

Based on our study, we expected to contribute the extant literature by filling up the theoretical gap on understanding the "stable" routines in the "fluid" online community. In particular, we aim to identify common routines across different OSS projects through the sequence motif detection techniques. By analyzing the variations of routines in different projects, we will discuss how the stability of organizational routines are changed by the fluid nature of online community. Our study will also provide important practical implication. The results can reveal how the development of OSS is actually carried out by different developers. This is especially important for firms hoping to leverage open innovation through sponsorship. By showing the potential factors affecting the design practice, firms will get a deeper understanding on how to "organize" OSS development.

Limitations

A classical challenge in the usage of optimal matching to determine the distance between two sequences derives from an ideal assumption that the cost of switching from any state to any other state is a constant. This is also the case when we adopt this technique in our second step. Although a customized cost matrix may address this issue, a bigger problem raised is how to derive differential yet reasonable costs for each transition. In addition, the sequences of actions we construct are generally dominated by Push action and Issue Comment action, therefore overwhelming potential impacts of other types of action. This problem can be mitigated by subdividing Push and Issue Comment actions into detailed actions, and utilizing those sub-actions to construct sequences of actions. A possible solution is to categorize Push actions by the number of lines of code that they added, deleted or modified. For Issue Comment actions, subdividing could be realized by using sentiment analysis or text mining techniques to discriminate issue comments (e.g. positive feedback versus negative feedback). Finally, this research could be limited due to the weak logical connections among activities in a sequence. The lack of logical relationships comes with the open and casual nature of OSS projects. To strengthen the inner logic of a sequence, we have proposed in the section above that we intend to construct an activity sequence for individual developers.

Conclusion

In this paper, we establish a theoretical foundation for organizational routines in open source software communities, and explore the possibility to use sequence motif mining and clustering analysis techniques to capture routines in OSS development process. In spite of a research in progress, this paper briefly shows the capability of the method while expecting to unleash its whole potential in a future study.

Acknowledgements

This paper is based on work supported, in part, by the National Science Foundation under Grant #1447670. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Aalst, W. v. d., Hee, K. M. v., and Publishing, E. 2002. Workflow Management: Models, Methods, and Systems. Cambridge, Mass: MIT Press.

Abbott, A., and Hrycak, A. 1990. "Measuring Resemblance in Sequence Data: An Optimal Matching Analysis of Musicians' Careers," American Journal of Sociology (96:1), pp. 144-185.

Bagozzi, R. P., and Dholakia, U. M. 2006. "Open Source Software User Communities: A Study of Participation in Linux User Groups," Management Science (52:7), pp. 1099-1115.

Bergquist, M., and Ljungberg, J. 2001. "The Power of Gifts: Organizing Social Relationships in Open Source Communities," Information Systems Journal (11:4), pp. 305-320.

Boland, R. J., Jr., Lyytinen, K., and Yoo, Y. 2007. "Wakes of Innovation in Project Networks: The Case of Digital 3-D Representations in Architecture, Engineering, and Construction," Organization Science (18:4), pp. 631-647.

Bonaccorsi, A., and Rossi, C. 2003. "Why Open Source Software Can Succeed," Research Policy (32:7), pp. 1243-1258.

Bradford, S. C. 1950. Documentation.

Burgin, R., and Ritschard, G. 2014. "A Decorated Parallel Coordinate Plot for Categorical Longitudinal Data," The American Statistician (68:2), pp. 98-103.

Cohen, M. D. 2007. "Reading Dewey: Reflections on the Study of Routine," Organization Studies (28:5), pp. 773-786.

Cohen, M. D., and Bacdayan, P. 1994. "Organizational Routines Are Stored as Procedural Memory: Evidence from a Laboratory Study," Organization science (5:4), pp. 554-568.

Crowston, K., Wei, K., Li, Q., and Howison, J. 2006. "Core and Periphery in Free/Libre and Open Source Software Team Communications," IEEE, pp. 118a-118a.

Elliott, R. J., Aggoun, L., and Moore, J. B. 1995. Hidden Markov Models: Estimation and Control. New York: Springer-Verlag.

Faraj, S., Jarvenpaa, S. L., and Majchrzak, A. 2011. "Knowledge Collaboration in Online Communities," Organization Science (22:5), pp. 1224-1239.

Faraj, S., and Johnson, S. L. 2011. "Network Exchange Patterns in Online Communities," Organization Science (22:6), pp. 1464-1480.

Farjoun, M. 2010. "Beyond Dualism: Stability and Change as a Duality," The Academy of Management Review (35:2), pp. 202-225.

Feldman, M. S., and Pentland, B. T. 2003. "Reconceptualizing Organizational Routines as a Source of Flexibility and Change," Administrative Science Quarterly (48:1), pp. 94-118.

Fleming, L., and Waguespack, D. M. 2007. "Brokerage, Boundary Spanning, and Leadership in Open Innovation Communities," Organization Science (18:2), pp. 165-180.

Gaskin, J., Berente, N., Lyytinen, K., and Yoo, Y. 2014. "Toward Generalizable Sociomaterial Inquiry: A Computational Approach for Zooming in and out of Sociomaterial Routines," MIS Quarterly (38:3), p. 849.

Gaskin, J., Lyytinen, K. J., Yoo, Y., and Pentland, B. 2012. "The Effects of Digital Intensity on Combinations of Sequential and Configural Process Variety,").

Grewal, R., Lilien, G. L., and Mallapragada, G. 2006. "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems," Management Science (52:7), pp. 1043-1056.

Hahn, J., Moon, J. Y., and Zhang, C. 2008. "Emergence of New Project Teams from Open Source Software Developer Networks: Impact of Prior Collaboration Ties," Information Systems Research (19:3), pp. 369-391.

Hippel, E. v., and Krogh, G. v. 2003. "Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science," Organization Science (14:2), pp. 209-223.

Howison, J., and Crowston, K. 2014. "Collaboration through Open Superposition: A Theory of the Open Source Way," MIS Quarterly (38:1), p. 29.

Lakhani, K. R., and von Hippel, E. 2003. "How Open Source Software Works: “Free” User-to-User Assistance," Research Policy (32:6), pp. 923-943.

Lerner, J., and Tirole, J. 2002. "Some Simple Economics of Open Source," The Journal of Industrial Economics (50:2), pp. 197-234.
Mabroukeh, N., and Ezeife, C. I. 2010. "A Taxonomy of Sequential Pattern Mining Algorithms," ACM Computing Surveys (CSUR) (43:1), pp. 1-41.
Neamsirorat, W., and Premchaiswadi, W. 2015. "Analysis of Surgical Event Logs in a Hospital by Using Heuristics Miner Technique," ICT and Knowledge Engineering (ICT & Knowledge Engineering 2015), 2015 13th International Conference on: IEEE, pp. 105-109.
Nonnecke, B., and Preece, J. 2000. "Lurker Demographics: Counting the Silent," ACM, pp. 73-80.
O'Mahony, S., and Ferraro, F. 2007. "The Emergence of Governance in an Open Source Community," The Academy of Management Journal (50:5), pp. 1079-1106.
Oh, W., and Jeon, S. 2007. "Membership Herding and Network Stability in the Open Source Community: The Ising Perspective," Management Science (53:7), pp. 1086-1101.
Ovaska, P. i., Rossi, M., and Marttiin, P. 2003. "Architecture as a Coordination Tool in Multi-Site Software Development," Software Process: Improvement and Practice (8:4), pp. 233-247.
Pentland, B. T. 1995. "Grammatical Models of Organizational Processes," Organization Science (6:5), pp. 541-556.
Pentland, B. T. 2003. "Sequential Variety in Work Processes," Organization Science (14:5), pp. 528-540.
Pentland, B. T., Hærem, T., and Hillison, D. 2010. "Comparing Organizational Routines as Recurrent Patterns of Action," Organization Studies (31:7), pp. 917-940.
Pentland, B. T., Hærem, T., and Hillison, D. 2011. "The (N)Ever-Changing World: Stability and Change in Organizational Routines," Organization Science (22:6), pp. 1369-1383.
Ritschard, G., Bürgin, R., and Studer, M. 2013. "Exploratory Mining of Life Event Histories," Contemporary Issues in Exploratory Data Mining in the Behavioral Sciences), pp. 221-253.
Sabherwal, R., and Robey, D. 1993. "An Empirical Taxonomy of Implementation Processes Based on Sequences of Events in Information System Development," Organization Science (4:4), pp. 548-576.
Setia, P., Rajagopalan, B., Sambamurthy, V., and Calantone, R. 2012. "How Peripheral Developers Contribute to Open-Source Software Development," Information Systems Research (23:1), pp. 144-163.
Singh, P. V., and Tan, Y. 2010. "Developer Heterogeneity and Formation of Communication Networks in Open Source Software Projects," Journal of Management Information Systems (27:3), pp. 179-210.
Singh, P. V., Tan, Y., and Mookerjee, V. 2011a. "Network Effects: The Influence of Structural Capital on Open Source Project Success," MIS Quarterly (35:4), p. 813.
Singh, P. V., Tan, Y., and Youn, N. 2011b. "A Hidden Markov Model of Developer Learning Dynamics in Open Source Software Projects," Information Systems Research (22:4), pp. 790-807.
Stewart, K. J., and Gosain, S. 2006. "The Impact of Ideology on Effectiveness in Open Source Software Development Teams," MIS Quarterly (30:2), pp. 291-314.
van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. 2005. "The Prom Framework: A New Era in Process Mining Tool Support." Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 444-454.