Social Community Evolution Analysis and Visualization in Open Source Software Projects

Jierui Zhang, Liang Wang(✉), Zhiwen Zheng, and Xianping Tao

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
{jieruizhang,zwzheng}@smail.nju.edu.cn, {wl,txp}@nju.edu.cn

Abstract. The importance of social communities around open-source software projects has been recognized. Despite that a lot of relevant research focusing on this topic, understanding the structures and dynamics of communities around open-source software projects remains a tedious and challenging task. As a result, an easily accessible and useful application that enables project developers to gain awareness of the status and development of the project communities is desirable. In this paper, we present MyCommunity, a web-based online application system to automatically extract communication-based community structures from social coding platforms such as GitHub. Based on the detected community structures, the system analyzes and visualizes the community evolution history of a project with a set of semantic-rich events, and quantify the strength of community evolution with respect to different events with a series of indexes. Built-in support to quantitative analysis and machine learning tasks based on the quantitative evolutionary events are provided. We demonstrate the usefulness of the system by presenting its ability in predicting project success or failure with the community evolution features. The results suggest the system achieves a prediction accuracy of 88.5% with commonly available machine learning models.

Keywords: Web-based application · Open source community analysis · Community evolution

1 Introduction

Open source software (OSS) developers form implicit collaborative social networks [2], i.e., developer social networks (DSNs) [10], when working together on social coding platforms like GitHub. In [10], the authors discover that community evolution events originally proposed for general social networks (GSNs) [13,15], including community *split*, *shrink*, *merge*, *expand*, *extinct*, and *emerge*, are also feasible in understanding community evolution in DSNs. They also discover that events of community evolution in DSNs correspond to the developing stages and important events in OSS projects. As a result, keeping aware of the structure and dynamics of DSNs around OSS projects is important for OSS maintainers due

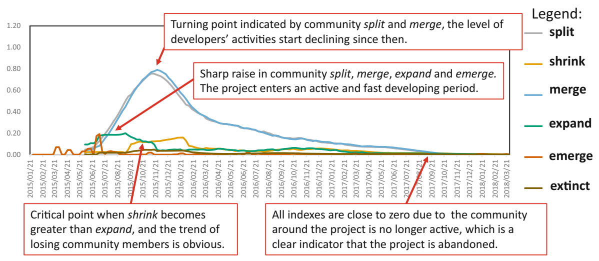


Fig. 1. Quantified community evolution events over time for GitHub project *google/material-design-lite*, with a moving average over 20 data points. It can be seen that the proposed approach can provide useful information about the dynamics of communities around the project, as well as the project's status. Better viewed in color.

to the strong correlation between the health of social communities and project viability [1,3,8–10].

In summary, this paper makes the following contributions. And our application can easily extend to commits and PRs.

1. We present a web-based online application called MyCommunity for community evolution analysis for OSS projects that implements a complete workflow from data collection, community detection, evolution analysis, and visualization, with high usability and integrity.
2. We extend existing approaches for community evolution event detection and provide quantitative analysis to the strength of community evolution with respect to the events.
3. We demonstrate the usefulness of the analysis results, and the functions of the proposed application in supporting intelligent analysis for OSS projects by predicting project success and failure with the quantified community evolution events and machine learning techniques.

2 Related Work

Existing studies show that community evolution in developer social networks and general social networks can be described by a set of semantic-rich events [10,13,15]. Community detection algorithms such as the Clauset-Newman-Moore (CNM) algorithm [14], clique percolation method (CPM) [15], etc, are proposed to discover communities inside social networks. Despite that community evolution can potentially provide valuable insights about the current and future status of OSS projects [10,12], performing community evolution analysis requires a lot of tedious work.

With respect to web-based systems, there are many applications and services developed to understand OSS projects. INFOX [17] is a web-based application to automatically identify non-merged features in forks and generate an overview of fork status of the project. OSR [11] is a visualization application to identify OSS developers' practices and generate biographies for them. However, applications that focus on understanding community evolution in OSS projects are rare. In this paper, we present the MyCommunity application to fill the niche and bridge the gap between community evolution research [10,13,15] and web-based services for OSS projects.

3 System Architecture

This section shows MyCommunity's architecture (see Fig. 2) which consists of three components. Each part is described as follows.

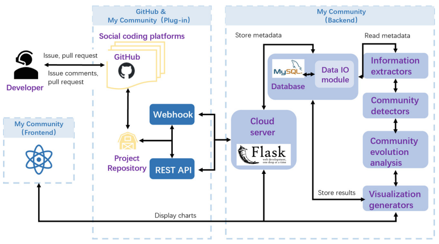


Fig. 2. Overview of system architecture.

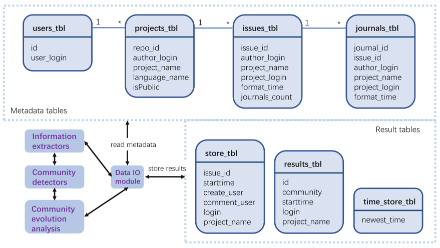


Fig. 3. Overview of database.

3.1 MyCommunity GitHub Plug-in

The third-party application has two mechanisms to obtain repository data on GitHub. The MyCommunity GitHub Plug-in, which establishes the connection between GitHub and the MyCommunity backend, is responsible for providing data and creating webhooks from GitHub. In addition, the MyCommunity GitHub plug-in gets all the data since the repository was built, and will provide a crawler function to help the application get the communication of this repository before setting up the webhook.

Users can connect our applications to different social coding platforms and databases by selecting different MyCommunity Plug-ins (see Fig. 4).

3.2 MyCommunity Backend

The backend of MyCommunity has four components: read/write data from a database, information extraction, community detection, and visualization generation (see Fig. 2). In particular, we made an image of our application based on image *tiangolo/uwsgi-nginx-flask:python3.9*[1], which can run Flask web applications in a single container with uWSGI and Nginx.

Data IO module uses a relational database to store data (see Fig. 3). The connection in the metadata table box represents the foreign key relationship.

[1] https://github.com/tiangolo/uwsgi-nginx-flask-docker.

Table 1. Rules to determine the evolution events of a community c detected in one time step, e.g., community i in time t (adopted from [10,16]).

Event	Community evolution event detection
extinct	Community c has no matching community in the next step. This event marks a massive exodus of developers from the community
emerge	Community c has no matching community in the prior step. This mode marks the rise of community discussion and may also represent new issues to be discussed
split	Community c has at least two communities matched to it in the next step. Members of the community have dispersed interests and are no longer working towards a common goal
shrink	Community c has only one community matched in the next step, and the size of the matching community is less than the size of $c^{\dagger}$. This event represents a shrinking community size and users are attracted to other communities
merge	Community c has at least two matching communities in the previous time step. This mode represents the integration of multiple communities, perhaps users have a common interest or solve a shared bug
expand	Community c only matches with one community in the previous step, and the size of the prior community is less than the size of c. This event represents an increase in the size of the community and new members are drawn into the community

†We modify the rule in [10] to match c with communities detected in the next step when determining community *shrink*. With this modification, there are even numbers of evolution events potentially associated with a community when matching communities in the previous and next time steps, respectively—community *extinct*, *split*, and *shrink* associated with communities in the next step, and community *emerge*, *merge*, and *expand* by matching communities in the previous step.

We use the user's login name and project name to uniquely identify a project, and use *issue_id* to uniquely identify an issue of the same project. Information extraction module, prepare data for subsequent community detection, can extract information from metadata and write results to the database using the Data IO module. In visualization generation module, the results of the calculation will be formatted in the form required for the presentation of the chart.

The algorithm in the application is described as follows.

Community Detection: Community detection uses the extension of the Clauset-Newman-Moore (CNM) algorithm [4,14], which can be used on weighted graphs using the strength of connections between the nodes indicated by edge weights. Input is all communication content in a time slice of a project for detection.

We apply an overlapping sliding window with a length of a month that slides temporally forward one week at a time. Then we obtain a series of segments $\boldsymbol{S} = \langle s_1, s_2, \ldots, s_T \rangle$. We then use a weighted, undirected graph $G_t = \langle V_t, E_t, U_t, W_t \rangle$ to model the structure of a developer social network (DSN) in the segment $s_t \in \boldsymbol{S}$, where V_t includes all the participants of conversations in s_t, and E_t is the set of edges representing the relationship between users in V_t. For each edge $e_{ij} \in E_t$, there is a edge weight $w_ij \in W_t$ that quantifies the strength of interactions between user v_i and v_j. And we assign a weight $u_i \in U_t$ to quantify the importance of user v_i.

Finally, we execute the community detection algorithm and get a set of non-overlapping communities $C_t = \{c_{t,1}, c_{t,2}, \cdots, c_{t,n}\}$ $(c_{t,i} \subseteq V_t)$ in graph G_t of the t-th snapshot.

Community Evolution Analysis: U_i^k represents the weight of the node k if it belongs to community i. And in Eq. 2, $S_{i,j}$ calculates the similarity between community i and community j.

$$U_i^k = \begin{cases} 0, k \notin c_i \\ u_i^k, k \in c_i \end{cases} \tag{1}$$

$$S_{i,j} = \frac{\sum_{k=0}^{p} max(U_i^k, U_j^k)}{min(\sum_{l=0}^{q} max(U_i^l, U_j^l), \sum_{m=0}^{r} max(U_i^m, U_j^m))} \tag{2}$$

where p is the set of the common nodes of community i and j, q and r are the node sets of the two communities before and after.

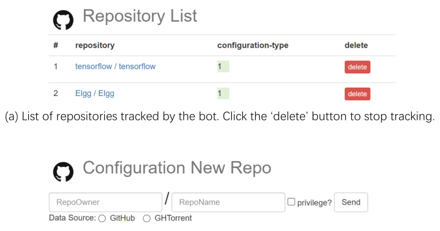
We filter out irrelevant communities by comparing $S_{i,j}$ with threshold ε=0.3 [10,16]. We obtained the predecessor community set $C_{prior_i}^t = \left\{c_{prior_i}^{t,1}, c_{prior_i}^{t,2} \ldots\ldots c_{prior_i}^{t,n}\right\}$ and the successor community set $C_{next_i}^t = \left\{c_{next_i}^{t,1}, c_{next_i}^{t,2} \ldots\ldots c_{next_i}^{t,n}\right\}$ of each community through the calculation of community tracer.

We adopt the six events used in [10,13,16] to determine the evolution of each community c by matching the communities detected in time steps before and after c is detected following the rules defined in Table 1. An evolutionary event vector is taken at time t-th denoted as V_t, is obtained by:

$$V_t = 0.001 * \left[\sum_{i=0}^{n} \left(P_i^t * W_{next_i}^t\right) + \sum_{j=0}^{m} \left(P_j^{t+1} * W_{prior_j}^{t+1}\right)\right] \tag{3}$$

where P_i^t is the event label of i-th community in t-th snapshot, $W_{prior_i}^t$ and $W_{next_i}^t$ represent its weight relative to its prior and post communities. Finally, the values and scales for each event are displayed inline and in pie charts, as shown in Fig. 5. We standardize all values to keep them in range $[0, 1]$.

3.3 MyCommunity Frontend



Repository List

#	repository	configuration-type	delete
1	tensorflow / tensorflow	1	delete
2	Elgg / Elgg	1	delete

(a) List of repositories tracked by the bot. Click the 'delete' button to stop tracking.

Configuration New Repo

RepoOwner / RepoName ☐ privilege? Send

Data Source: ○ GitHub ○ GHTorrent

(b) Add a new repository to be tracked and analyzed, and specify the data source.

Fig. 4. The configuration page where users can: (a) list existing repositories; (b) add new repositories from a specified data source.

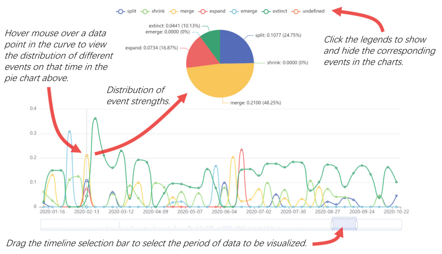

split
shrink
merge
expand
emerge
extinct
undefined
Hover mouse over a data point in the curve to view the distribution of different events on that time in the pie chart above.
extinct: 0.0441 (10.13%)
emerge: 0.0000 (0%)
expand: 0.0734 (16.87%)
split: 0.1077 (24.75%)
shrink: 0.0000 (0%)
merge: 0.2100 (48.25%)
Click the legends to show and hide the corresponding events in the charts.
Distribution of event strengths.
0.4
0.3
0.2
0.1
0
2020-01-16
2020-02-13
2020-03-12
2020-04-09
2020-05-07
2020-06-04
2020-07-02
2020-07-30
2020-08-27
2020-09-24
2020-10-22
Drag the timeline selection bar to select the period of data to be visualized.

Fig. 5. Visualization of community evolution events with interactive charts provided by the frontend.

The frontend of our application consists of a series of HTML files returned by the backend.

There are three parts that users can operate MyCommunity directly: configuring settings, selecting a repository that you can view, and viewing the results of the analysis for a specific project.

Users can configure the owner and project name of a repository to get a convenient and unique identification (see Fig. 4). Then the application will crawl all communication information by GitHub REST API as well as waiting for a callback to be triggered. After the configuration process is completed, users can view their repository in the list of repositories and click the table item they want to check.

There are four charts on the exhibit page (see Fig. 5). We use *echarts*[2] combined with JavaScript files to generate charts. Event line charts are a higher level of abstraction of the community evolution map, quantitatively reflecting the evolutionary history of the community. We can look at the lines from two dimensions: the event proportion and the overall trend. The overall trend chart can be observed by the line chart below Fig. 5, while the event scale can be seen in the pie chart above Fig. 5. We explain an example of a project which is once active and has stopped developing, *google/material-design-lite*, in Fig. 1, and show the overall trend of the project and the continuous decline of certain indices (e.g. Continued decline in a split and merge means less active project members and reflects the continued increase in the project's risk of failure) may indicate the future success or failure of the project.

4 Predict Project Success or Failure

To demonstrate the usefulness of the proposed system, and to help developers to gain information about the future trend of their projects, we provide in our system a function to predict the success or failure of a project based on the time series of quantified community evolution events as shown in Fig. 6. As illustrated in Fig. 1, the curve trend of the project community model has some practical significance, representing certain specific events, and the occurrence of these specific times ultimately affects the success or failure of the project. We built and evaluated the model based on a dataset of 339 successful projects and 192 failed projects. We used the number of stars to measure popularity and selected some of the most popular projects from GitHub. Next we label these projects as *success* and *failure*. The failed projects are selected following the steps proposed in [7], which are projects without receiving commits for a year and declared as no longer maintained in the documents (or manually checked by authors).

First, we extract features from the sequential data for each of the community evolution events using the *tsfresh* package [6]. To filter features, we use Fisher's exact test to determine whether the feature is strongly or weakly correlated with the resulting label (based on Scalable Hypothesis tests) [5]. Additionally, we store the filtered feature dict for later use on the test set. We use the number n to denote the final count of features. For a project j, we generate a $n+1$-dimensional vector using *tsfress*: $M_j = \left\langle f_1^j, f_2^j, \ldots, f_i^j, \ldots, f_n^j, L_j \right\rangle$, where f_i^j

[2] https://echarts.apache.org/zh/index.html.

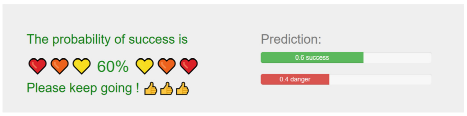

The probability of success is ❤️🧡💛 60% 💛🧡❤️
Please keep going ! 👍👍👍

Fig. 6. An example of predicting results of the future success or failure of a project.

Table 2. Accuracy of different classification models

Model	Accuracy	Precision	Recall	F1
Decision Tree Classifier	0.840 ± 0.056	0.783	0.771	0.777
Random Forest Classifier	0.885 ± 0.041	0.849	0.823	0.836
SVM	0.828 ± 0.078	0.755	0.802	0.778
K-Neighbors Classifier	0.871 ± 0.035	0.815	0.849	0.832

represents the i-th feature and L_j represents the label of the j^{th} project. The label marks the success or failure of a given project.

Next, we use the 10-fold-cross-validation to compare and select a suitable model. Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. We compared the efficiency of several models (see Table 2) and finally chose Random Forest Classifier.

Finally, we evaluate the model and give the project in the application a success or failure prediction result. It can be seen that the results obtained by the random forest classifier are most consistent with the true values. The overall mean accuracy of our model is 88.5%. Precision, which is the fraction of relevant instances among the retrieved instances, is 84.9%. While recall, which is the fraction of relevant instances that were retrieved, is 82.3%. And F1 score, which is the harmonic mean of precision and recall, is 83.6%.

5 Conclusion

We introduce the MyCommunity application which is a community metrics application seamlessly integrated with GitHub to assist developers in monitoring project communication status and notify them of the probability of project success. MyCommunity's main contribution is the ability to support the quantitative and automatic analysis of how the developers evolved in the project.

Our future work involves support for a more accurate predictive model and more systematic evaluation. During the evaluation, we identified the need for further improvements in our predictions such as obtaining a larger dataset.

Acknowledgement. This work is supported by the National Key R&D Program of China under Grant No. 2018AAA0102302, the NSFC under Grant No. 62172203, and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

1. Antwerp, M.V., Madey, G.: The importance of social network structure in the open source software developer community. In: 2010 43rd Hawaii International Conference on System Sciences (HICSS) (2010)
2. Begel, A., Khoo, Y.P., Zimmermann, T.: Codebook: discovering and exploiting relationships in software repositories. In: ACM/IEEE International Conference on Software Engineering (2010)
3. Catolino, G., Palomba, F., Tamburri, D.A., Serebrenik, A., Ferrucci, F.: 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 Society (ICSE-SEIS) (2019)
4. Chakraborty, T., Dalmia, A., Mukherjee, A., Ganguly, N.: Metrics for community analysis: a survey. ACM Comput. Surv. (CSUR) **50**(4), 1–37 (2017)
5. Christ, M., Kempa-Liehr, A.W., Feindt, M.: Distributed and parallel time series feature extraction for industrial big data applications (2016)
6. Christ, M., Braun, N., Neuffer, J., Kempa-Liehr, A.W.: Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh - a Python package). Neurocomputing **307**, 72–77 (2018)
7. Coelho, J., Valente, M.T.: Why modern open source projects fail. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 186–196 (2017)
8. Ducheneaut, N.: Socialization in an open source software community: a socio-technical analysis. Comput. Support. Coop. Work **14**(4), 323–368 (2005)
9. Hannemann, A., Klamma, R.: Community dynamics in open source software projects: aging and social reshaping. In: Petrinja, E., Succi, G., El Ioini, N., Sillitti, A. (eds.) OSS 2013. IAICT, vol. 404, pp. 80–96. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38928-3_6
10. Hong, Q., Kim, S., Cheung, S.C., Bird, C.: Understanding a developer social network and its evolution. In: 2011 27th IEEE International Conference on Software Maintenance (ICSM), pp. 323–332. IEEE (2011)
11. Jaruchotrattanasakul, T., Yang, X., Makihara, E., Fujiwara, K., Iida, H.: Open source resume (OSR): a visualization tool for presenting OSS biographies of developers. In: 2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP), pp. 57–62 (2016). https://doi.org/10.1109/IWESEP.2016.17
12. Le, Q., Panchal, J.H.: Analysis of the interdependent co-evolution of product structures and community structures using dependency modelling techniques. J. Eng. Des. **23**(10–11), 807–828 (2012)
13. Lin, Y.R., Sundaram, H., Chi, Y., Tatemura, J., Tseng, B.L.: Blog community discovery and evolution based on mutual awareness expansion. In: IEEE/WIC/ACM International Conference on Web Intelligence (WI 2007), pp. 48–56. IEEE (2007)
14. Newman, M.E.: Analysis of weighted networks. Phys. Rev. E **70**(5), 056131 (2004)
15. Palla, G., Barabási, A.L., Vicsek, T.: Quantifying social group evolution. Nature **446**(7136), 664–667 (2007)
16. Wang, L., Li, Y., Zhang, J., Tao, X.: Quantitative analysis of community evolution in developer social networks around open source software projects. arXiv preprint arXiv:2205.09935 (2022)
17. Zhou, S., Stanciulescu, S., Leßenich, O., Xiong, Y., Wasowski, A., Kästner, C.: Identifying features in forks. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 105–116. IEEE (2018)