Mining and Predicting Micro-Process Patterns of Issue Resolution for Open Source Software Projects

Yiran Wang[‡], Jian Cao*[‡] and David Lo[§]
[‡]Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, 200240, China
[§]School of Information Systems, Singapore Management University, 178902, Singapore
Email: [‡]{wangyiran33,cao-jian}@sjtu.edu.cn, [§]davidlo@smu.edu.sg

***Abstract*—Addressing issue reports is an integral part of open source software (OSS) projects. Although several studies have attempted to discover the factors that affect issue resolution, few pay attention to the underlying micro-process patterns of resolution processes. Discovering these micro-patterns will help us understand the dynamics of issue resolution processes so that we can manage and improve them in better ways. Of the various types of issues, those relating to corrective maintenance account for nearly half hence resolving these issues efficiently is critical for the success of OSS projects. Therefore, we apply process mining techniques to discover the micro-patterns of resolution processes for issues relating to corrective maintenance. Four and five typical patterns are found for the identification stage and solving stage of the resolution processes respectively. Furthermore, it is shown that the consequent patterns can be predicted with a certain degree of accuracy by selecting the appropriate features and models. Furthermore, we make use of the pattern information predicted to forecast the issue lifetime and the results show that this information can also improve the accuracy in the earlier observation points. At the same time, pattern predictions provide good interpretability to the forecast of issue lifetime.**

***Index Terms*—issue resolution, micro-pattern, process mining, issue pattern prediction, issue lifetime prediction**

I. Introduction

Issue registering, tracking and resolution are very important in open source software (OSS) projects [1]. Previous studies show that problems of issue overstocking may arise over time [2] and a considerable portion of issues in GitHub are pending for months or even over a year [3]. Therefore, effective strategies should be implemented to improve the issue resolution process such as prioritizing issues to be resolved and allocating resources for each issue clearly in practice and more importantly, we should understand the influential factors behind successful issue resolution processes.

As a collaboration process, issue resolution consists of multiple actions that occur in order and involves several persons. It is beneficial for us to understand issue resolution from the process view. For example, bottlenecks, the critical paths and the most frequent paths can be identified and studied when issue resolution is modeled as a process. Predicting the resolution time is not sufficient to understand and manage the dynamic issue resolution process that is full of uncertainties. Therefore, we aim to provide a richer prediction on the ongoing process.

DOI reference number: 10.18293/SEKE2020-031
*Corresponding Author

Issue resolution is essentially a problem-solving process and follows a typical problem-solving procedure that includes steps consisting of defining the problem, brainstorming the ideas, deciding on a solution, implementing a solution and reviewing the results. However, these steps can only roughly describe the issue resolution process and in practice, there are many details for each step. The detailed typical process models for different steps or stages are called ***micro-process patterns*** or ***micro-patterns*** in this paper. On open source hosting platforms such as GitHub, different types of events, including development events and interaction events, are recorded in event logs. We can mine micro-patterns for issue resolution processes from these event logs by applying process mining techniques [4].

The value of process mining in OSS projects, as been described in [5], is that it not only reveals the variety of processes followed by open source communities, it also helps standardize or improve core activities. Unfortunately, no research has been conducted on the topic of process pattern mining and prediction on issue resolution processes yet. Knowing typical micro-process patterns help developers, managers and stakeholders gain a deeper understanding on the processes of issue resolution. It would be extremely helpful to risk assessment, work prioritization and resource allocation. For example, some patterns cost more time than others so that OSS project members can take actions to guide an issue resolution process to follow more efficient patterns to intentionally speed up the resolution process. At the same time, micro-process pattern information can also be a useful feature for the resolution time prediction model and provides better interpretability to the prediction results.

There are different types of issues and they may have very different micro-process patterns. In this paper, we focus on the issues relating to the maintenance activities. Moreover, the ISO/IEC 14764 standard [6] defines four types of maintenance activities spanning the different motivations that software engineers have while undertaking changes to an existing software system. Issues can be mapped to these maintenance activities through a classification model [7]. In this paper, we only focus on issues relating to corrective maintenance, which corresponds to bugs and accounts for nearly half of all issues.

Therefore, in this paper, we aim at answering the following research questions:

- *RQ1: What are the frequent micro-process patterns of resolution processes of issues relating to corrective main-*

tenance in OSS projects?

- *RQ2: Is it possible to predict which micro-process pattern will appear during the issue's lifetime?*
- *RQ3: Is pattern information useful for issue lifetime prediction?*

In order to answer these questions, firstly, we mine frequent process patterns in different stages of issue resolution using process mining techniques. The distributions and characteristics of these patterns in various projects are analyzed. Then, we construct models for pattern prediction during issue lifetime using dynamic and static features. Finally, we try to utilize the pattern probability predicted as an additional feature for issue lifetime predictions.

II. Related Work

Influencing factors for issue resolution time [2], [7]–[9], bug-fixing time prediction and issue lifetime prediction [10]–[19] have received significant attention in recent years. Weiss et al. [11] predict the time spent on fixing an issue based on the average time of its similar issues. Al-Zubaidi et al. [19] use multi-objective search-based approach to estimate issue resolution time which makes estimation models accurate and simple simultaneously. Giger et al. [12] and Rees-Jones et al. [14] present decision-tree-based models to predict bug fix time while Panjer et al. [13] utilize logistic regression models, Zhang [15] utilize kNN-based model. Kikas [10] predict issue resolution time based on random forest models using dynamic and contextual features. In this work, we also construct issue lifetime prediction models but our emphasis is to show an improvement in the predictors' performance when calculating the probability of micro-process patterns predicted into features.

Process mining is now considered to be in a mature phase allowing its application to extract knowledge from event logs to a variety of sectors. Applying process mining in open-source software communities has seldom been studied [20]–[23]. [21]–[23] uses the characteristics of process mining to perform conformance checks to study the differences between the actual bug life cycle and the standard process on the guide. However, these studies are confined to extracting the overall process model and none of them mine internal micro-process patterns further. In contrast, we try to discover the micro-process patterns in the issue resolution process.

III. Mining Frequent Micro-Patterns of the Issue Resolution Process

In this section, we answer RQ1. In order to mine the micro-process patterns of issue resolution, we need to observe the micro-processes of issues. The events extracted through GitHub APIs record what has happened in an issue resolution process. We perform pre-processing on the raw data and filter some helpful and common events which are shown in I to make an event log made up of 38978 records and apply process mining algorithms on it.

TABLE I
Events of issues used in this study

Event name	Description
Created	The issue was created by the actor.
Assigned	The issue was assigned to the actor.
Labeled	A label was added to the issue.
Mentioned	The actor was @mentioned in an issue body.
Referenced	The issue was referenced from a commit message.
Renamed	The issue title was changed.
Reopened	The issue was reopened by the actor.
Closed	The issue was closed by the actor.

A. Dataset

In this study, we collect issue data using its public APIs[1] from GitHub and only closed issues updated at or after January 1, 2017 are included. The ten popular projects used in this study vary in domain, scale and programming language and they all provide dynamic platforms for bug reporting, discussing and fixing.

Since we focus on issues relating to corrective maintenance, issue reports must be classified first. We rely on labels applied to each issue report in GitHub to identify their maintenance type. Labels used by developers in GitHub are reliable since they are applied by the persons who actually perform the maintenance activity [7]. Finally, 4863 issues relating to corrective maintenance are collected and included in the dataset. We perform a preliminary analysis on the duration distribution for the dataset and find that the majority of issues are closed in a short period while few issues are pending for a long time, which appears to be a typical long-tail distribution.

B. Approach

The main approach we use to discover micro-processes is process mining. This technique has been successfully applied to distill a structured process description from a set of real executions in practice [24]. Its main objective is to discover processes, do conformance checking and process improvement. Many algorithms can be applied to generate process models like α–algorithm, heuristic miner, genetic algorithm. In accordance with these methods, process mining automatically discovers fact-based process models out of the raw data. Therefore, we use process mining to discover micro-process patterns in issue resolution processes. Celonis[2] as a mature process mining tool is used in this work.

C. Findings

The original extracted model is very complex and difficult to read and understand, so we remove a few activities and connections with low frequency to improve the readability of the model. Figure 1 shows the process model with 97.3% activities and 87.2% connections covered. Dashed arrows indicate connections from the Process Start or to the Process End. It can be easily found that the most common process path is :

[1]https://developer.github.com/v3/

[2]https://academiccloud.celonis.com

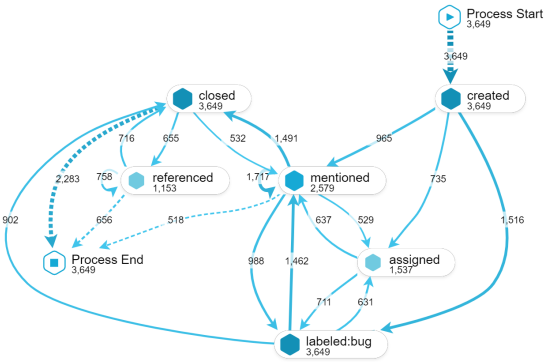


Fig. 1. The process model with 97.3% activities and 87.2% connections covered

created − > labeled:bug − > closed

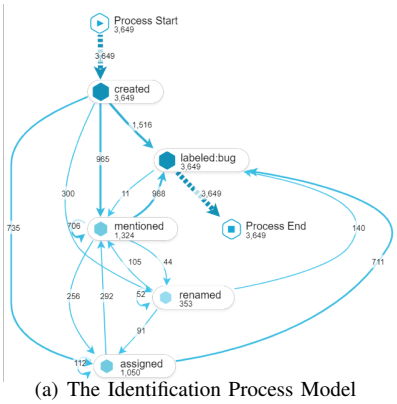
This indicates that most issue resolution processes start with 'created' and end with 'closed', via 'labeled:bug' except for those which still have activities after the issue is closed. For this reason, the whole process model can be divided into two stages by the 'labeled:bug' activity:

1) *The First Stage (Identification Stage): 'created' to 'labeled:bug':* In this stage, project contributors inspect the issue in order to know the environment and details with or without further conversations with the issue authors. If they judge the issue is a bug rather than a misuse, they will assign the bug label to the issue for further fixing.
2) *The Second Stage (Solving Stage): 'labeled:bug' to 'closed':* In this stage, when an issue is confirmed as a bug, the actors try to fix it. It may finish directly without any activities, or a contributor or a team may be @mentioned or assigned to fix the found bug, and commits may be made to fix the bug.

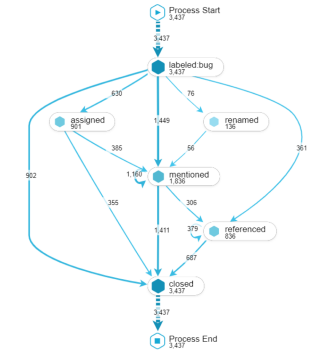
Compared with the general problem solving process model, the first stage roughly corresponds to the step of defining the problem while the second stage corresponds to brainstorming the ideas, deciding on a solution, implementing a solution and reviewing the results. For issue resolution processes in OSS projects, it is very difficult if not impossible to divide the second stage into different steps since these steps are interwoven. Figure 2 shows the process models for the two stages, respectively.

Table II presents the most frequent micro-process model variances which exceed 5% of all for each stage. In general, Stage 1 costs much less time than Stage 2, which indicates that it is easier to confirm a bug than to fix it. The bottleneck period usually occurs in Stage 2.

In each stage, the durations differ clearly between patterns. It should be noted that activities with the circle arrow(↻) mean the resolution process goes through the activity twice or more times. The reason why we don't merge it into the process model that goes through the activity only once is that their durations of them differ greatly (See Table II). For example, the third pattern of the 1st stage spends 34.9 more days(68.7%) than the fourth pattern on average, and the only difference between these two patterns is that the former goes through 'mentioned' activity twice or more while the latter goes through it only once. Going through the 'mentioned' activity twice or more may mean a several rounds of discussion or @mentioning a team and @mentioning a team often costs more time than @mentioning a person when judging a bug. It must be noted that spending more time doesn't necessarily imply inefficiency. The issue lifetime depends on many factors, one being the complexity of an issue. For a complex issue, the project manager may @mentioning a team so that more time is needed to label this issue and this is often the right way.



(a) The Identification Process Model



(b) The Solving Stage Process Model

Fig. 2. The two-stage process models

TABLE II
FREQUENT MICRO PROCESS PATTERNS IN TWO STAGES

Pattern No.	Micro Process Patterns	Freq.	Duration Median	Standard Deviation
The 1st Stage: Identification Stage				
1	created − > labeled:bug	41%	14.2d	69.8d
2	created − > assigned − > labeled:bug	12%	11.4d	52.0d
3	created − > mentioned↻ − >labeled:bug	9%	50.8d	134.0d
4	created − > mentioned − > labeled:bug	8%	15.9d	56.4d
others		30%	57.5d	127.5d
The 2nd Stage: Solving Stage				
1	labeled:bug − > closed	26%	114.7d	226.0d
2	labeled:bug − > mentioned − > closed	10%	134.8d	237.3d
3	labeled:bug − > mentioned ↻ − > closed	17%	189.5d	279.3d
4	labeled:bug − > assigned ↻ − > closed	7%	29.8d	72.0d
5	labeled:bug − > referenced ↻ − > closed	8%	61.5d	142.4d
others		32%	79.5d	232.6d

Another finding is that processes going through the 'assigned' activity in stage 1 or 2 and processes going through the 'referenced' activity in stage 2 will reduce the durations greatly. This conforms to our common sense that explicitly assigning the task to people can increase efficiency and it is extremely likely that committing to a pull request marks the end of issue resolution. It may also mean that when an issue is easy, the manager clearly knows who should be responsible for it and can directly assign it to him or a developer can directly commit a pull request to fix it.

IV. Pattern Prediction

Predicting which pattern occurs next in advance during issue lifetime can provide with a richer and more interpretable forecast result and in this section, we construct models to predict patterns to answer RQ2. The steps include feature selection, model training and evaluation.

A. Feature Selection

The performance of prediction models relies on features that are properly selected. Obviously, the patterns to be followed are affected by many factors. Our feature engineering is based on the work of [10] and [14]. Furthermore, we also add the following new features to the features we chose based on previous work: *CodeIncluded* for whether the body of an issue includes code or not, *CleanedTitleLength* for the number of words in the issue body with markdown parsed and tags removed and *CreatorAuthority* for whether the creator has authoritative identity in the project. It should be noted that since we would like to predict emerging patterns with time, the dynamic features proposed in [10] are used.

The selected features can be divided into three classes, i.e., **Issue features**, **Issue creator features** and **Project features**. Issue features describe the contents of an issue and its related events. Issue creator features reflect the characteristics of the author of an issue, which relates to issue contents and quality. The resolution processes of issues are also be affected by their projects and project features reflects their issue resolution statuses and activity levels. Features are not listed for lack of space.

B. Model Training

The target of pattern prediction is to select the most possible pattern type from a limited number of pattern types. This can be regarded as a classification problem. Since we will predict patterns with time, we trained different classification models at different observation points. For example, the observation point of 1 day means that we make a pattern prediction for an issue that has been opened for 1 day.

The observation points are chosen to match calendric periods, which leads to six observation points (1, 7, 14, 30, 90, and 180 days) . For each observation point, we train two classifiers to predict the pattern for the 1st stage and the 2nd stage respectively. Finally, we train 12 models in total.

TABLE III
The macro-F1(macF1) and micro-F1(micF1) scores at different stages

metrics		Observation Point					
		1d	7d	14d	30d	90d	180d
Stage 1	macF1	0.617	0.679	0.653	0.591	0.604	0.664
	micF1	0.706	0.741	0.746	0.719	0.756	0.869
Stage 2	macF1	0.469	0.646	0.667	0.674	0.708	0.590
	micF1	0.508	0.626	0.661	0.682	0.713	0.702

C. Pattern Prediction Performance

We trained multiple classifiers including *MultiLayer Perceptron*, *Linear Discriminant Analysis*, *Gaussian Naive Bayes*, *Multinomial Naive Bayes*, *Bernoulli Naive Bayes*, *Logistic Regression*, *Decision Tree* and *Random Forest*. Random forest classifiers [25] perform best for most of the time and we utilize them in the following section. Table III shows the macro-averaged F1-score and micro-averaged F1-score [26] for Random forest classifiers at different observation point.

We calculate yhe Top-10 ranking of feature importance for models at different observation points. The importance distributions of features of different stages at the same observation point are quite similar. However, we can find a huge difference between importance distributions of features in different observation points of the same stage. In early periods, static features seem to play a greater role while at late observation points dynamic features play a major role. It is also shown that 'nMentionedByT' which denotes 'Number of times actors was mentioned in the issue body before T' is always of great importance.

To summarize the findings with respect to RQ2, it can be concluded that we can predict which pattern has the highest probability of appearing during issue lifetime with a certain degree of accuracy by selecting appropriate features and models.

V. Issue lifetime Prediction with Micro-Process Pattern Information

In order to answer RQ3, we construct models with micro-process pattern information in contrast to models without pattern information to show that predicting patterns with time is beneficial to lifetime prediction.

A. Feature Selection

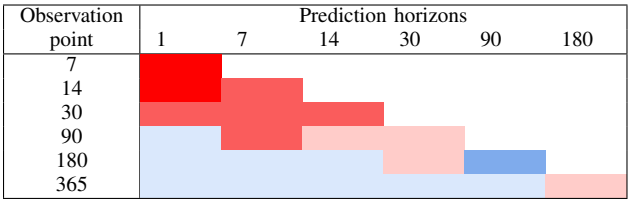
The feature selection is similar to last section. In addition, predicted pattern information is used as the extra inputs to the models. Rather than providing the concrete predicted pattern information, here we provide the appearance probabilities of patterns as inputs, for example, [0.12, 0.21, 0.03, 0.64] for the 1st stage. The reason is the consequent pattern is essentially nondeterministic and can be changed with time due to several factors. This is also the reason why pattern prediction accuracy is not so high. Therefore, providing the name of a most possible pattern is too risky. If the predicted result is wrong, it will completely mislead the lifetime prediction model.

TABLE IV
PREDICTION PERFORMANCES OF MODELS FOR DIFFERENT OBSERVATION POINTS AND PREDICTION HORIZONS

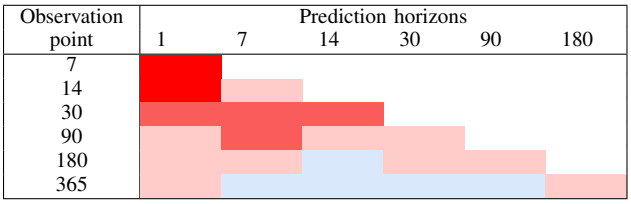
Pred. horizon	macro-F1			micro-F1			AUC		
	init	*prob*	*fore*	*init*	*prob*	*fore*	*init*	*prob*	*fore*
Observation at 1 day after issue is opened									
7d	0.524	**0.576**	0.554	0.740	**0.785**	0.759	0.531	**0.571**	0.552
14d	0.575	**0.623**	0.611	0.664	**0.704**	0.686	0.574	**0.618**	0.607
30d	0.661	**0.680**	0.671	0.662	**0.682**	0.680	0.658	**0.678**	0.672
90d	**0.696**	0.695	0.696	0.720	**0.726**	0.722	0.691	0.690	**0.692**
180d	**0.741**	0.728	0.737	0.803	**0.804**	0.803	**0.724**	0.708	0.720
365d	0.774	0.772	**0.775**	0.892	**0.894**	0.892	0.740	0.734	**0.741**
Observation at 7 days after issue is opened									
14d	0.508	**0.535**	0.512	0.860	**0.865**	0.854	0.517	**0.538**	0.517
30d	0.647	**0.669**	0.673	0.738	**0.761**	0.758	0.640	**0.667**	0.661
90d	0.673	**0.693**	0.680	0.676	**0.698**	0.680	0.678	**0.690**	0.680
180d	0.739	0.736	**0.746**	0.762	**0.765**	0.770	0.732	0.729	**0.740**
365d	0.778	0.756	**0.783**	0.864	0.852	**0.866**	0.750	0.727	**0.754**
Observation at 14 days after issue is opened									
30d	0.609	**0.637**	0.633	0.796	**0.820**	0.811	0.596	**0.618**	0.614
90d	0.660	**0.667**	0.661	0.667	**0.677**	0.670	0.657	**0.661**	0.661
180d	**0.744**	0.737	0.740	0.750	0.748	**0.751**	0.737	0.734	**0.737**
365d	0.751	0.750	**0.765**	0.839	0.836	**0.845**	0.730	0.725	**0.739**
Observation at 30 days after issue is opened									
90d	0.624	**0.626**	0.613	0.741	**0.743**	0.733	0.617	**0.620**	0.609
180d	0.712	**0.730**	0.716	0.716	**0.732**	0.718	0.713	**0.733**	0.717
365d	**0.749**	0.735	0.743	**0.790**	0.784	0.787	**0.738**	0.722	0.732
Observation at 90 days after issue is opened									
180d	**0.590**	0.584	0.554	0.724	**0.729**	0.721	0.578	**0.580**	0.566
365d	**0.703**	0.702	0.694	**0.716**	0.713	0.710	**0.704**	0.702	0.695
Observation at 180 days after issue is opened									
365d	0.678	0.683	**0.691**	0.692	0.708	**0.725**	0.679	0.689	**0.691**

TABLE V
HEAT MAP FOR VARIOUS MODELS AT DIFFERENT OBSERVATION POINTS AND PREDICTION HORIZONS.

(a) macro-F1



(b) micro-F1



Red ■ ■ ■ denotes that a model with pattern probabilities performs better than initial model and blue ■ ■ ■ denotes the contrary. A dark color ■ ■ denotes the gap is more than 6%, the medium ■ ■ denotes the gap is between 3%-6% and a light color ■ ■ denotes the gap is less than 3%.

B. Model Training

As in [10], our lifetime prediction tries to answer the question as to whether the issue can be closed before the given time or not. Therefore, it is a classification problem and we train prediction models at different observation points. At each observation point, we make predictions for whether it will be closed with different prediction horizons. Naturally, the prediction horizon should end after the observation point. For each combination of an observation point and a prediction horizon, we should train one model. For example, we make a prediction for an issue to answer the question as to whether it will be closed within 30 days after it has been opened for 14 days.

C. Prediction Performance

Although the prediction task is a binary classification problem in our study, we still utilize the macro-averaged F1-score and micro-averaged F1-score to evaluate the classifiers because correctly predicting the fact that an issue can be closed in a given prediction horizon or not is equally important so that traditional metrics for binary-classification are not sufficient. As in [27], we also use random forest as the model for this task because it has been proven that random forest is better than other conventional models on this task.

We analyze model performance for different observation points and compare initial models ($init$), models with pattern probability predicted ($prob$) and models with exact pattern predicted ($fore$). Table IV shows the obtained macro-averaged and micro-averaged F1-scores of various models.

Accordingly, Table V shows the performance comparison in the form of a heat map. We find that models with pattern probability predicted achieve better performance when doing short-term prediction at earlier observation points.

Firstly, pattern probability is similar to prior probability. At an early period when other features can barely provide information, adding prior probability is of great help for classifiers. As time progresses, other features are of more value and begin to modify the 'prior probability' and even break away from it. So, the value of pattern probability become less and less at later observation points.

Secondly, we find that after adding pattern probability features, the classifiers tend to predict that a certain issue can be closed within a given period. Without pattern probability, issue lifetime predicted may range widely. But with pattern probability, the resolution time distribution predicted may shrink due to the pattern restrictions. In this case, the capability of the model to distinguish the unconventional ultra-long period issue becomes weak, so prediction performance deteriorates after adding pattern features.

Another interesting finding is that models with pattern probability predicted even perform better than models with the foresight of exact patterns in many cases, especially at earlier observation points. We explain the phenomenon in this way: The model precision is low at an early period, adding an exact pattern may result in over-fitting to some extent. Nevertheless, the probability of patterns means various possibilities, which improves the ability of generalization.

VI. THREATS TO VALIDITY

Threats to internal validity In this study, we assume that properties of issues in each project are uniformly distributed while the heterogeneity of issues in each project cannot

be avoided. In addition, there are may several relationships between some issues in a project, i.e., they are not independent. These will inevitably affect our results to some extent. Another issue is we apply 17 features to predict possible patterns while the most important factor, the complexity of the issue itself, is not included since it is difficult to measure directly. We have tried to remedy this by putting some of the features that are closely related to issue complexity into the model. Also, issue misclassification is reported to occurs in [28], which may have impact on issue pattern prediction and issue lifetime prediction in our study.

Threats to external validity In this study, we use issues from 10 projects that are representatives of the open source domain which have different backgrounds, development practices and goals. To improve generality, we propose extending our study to more representative projects.

VII. Conclusions

In this paper we try to mine micro-process patterns of the resolution process of issues of a corrective maintenance type. Based on the issues extracted from 10 distinctive and representative projects in the open source domain, we apply process mining techniques to extract process patterns from them. We divide the whole issue solving process into two stages, i.e., the identifying stage and the solving stage. Four and five patterns are discovered for the first stage and second stage, respectively. Then we analyze their properties and get some interesting findings. After this, we construct models for pattern prediction during issue lifetime using static and dynamic features and our model shows an improved performance with time. Then we construct models for issue lifetime prediction in GitHub projects for different calendric periods with the probability of patterns predicted in order to demonstrate value within pattern information. The results show that models with predicted pattern information achieve better accuracy for issue lifetime prediction at earlier observation points.

Acknowledgment

This work is partially supported by National Key Research and Development Plan(No. 2018YFB1003800).

References

[1] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, 2010, pp. 291–300.

[2] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, and S. Liu, "Exploring the characteristics of issue-related behaviors in github using visualization techniques," *IEEE Access*, vol. 6, pp. 24 003–24 015, 2018.

[3] R. Kikas, M. Dumas, and D. Pfahl, "Issue dynamics in github projects," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2015, pp. 295–310.

[4] W. Van Der Aalst, *Process mining: discovery, conformance and enhancement of business processes*. Springer, 2011, vol. 2.

[5] E. Kouzari and I. Stamelos, "Process mining in software events of open source software projects," in *2nd International Symposium & 24th National Conference on Operational Research, HELORS*, 2013, pp. 25–27.

[6] ISO/IEC, "International standard-iso/iec 14764 ieee std 14764-2006 software engineering; software life cycle processes &; maintenance," 2006.

[7] A. Murgia, G. Concas, R. Tonelli, M. Ortu, S. Demeyer, and M. Marchesi, "On the influence of maintenance activity types on the issue resolution time," in *Proceedings of the 10th international conference on predictive models in software engineering*. ACM, 2014, pp. 12–21.

[8] G. Destefanis, M. Ortu, S. Counsell, S. Swift, M. Marchesi, and R. Tonelli, "Software development: do good manners matter?" *PeerJ Computer Science*, vol. 2, p. e73, 2016.

[9] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *2012 19th Working Conference on Reverse Engineering*. IEEE, 2012, pp. 225–234.

[10] R. Kikas, M. Dumas, and D. Pfahl, "Using dynamic and contextual features to predict issue lifetime in github projects," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 291–302.

[11] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 1–1.

[12] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. ACM, 2010, pp. 52–56.

[13] L. D. Panjer, "Predicting eclipse bug lifetimes," in *Proceedings of the Fourth International Workshop on mining software repositories*. IEEE Computer Society, 2007, p. 29.

[14] M. Rees-Jones, M. Martin, and T. Menzies, "Better predictors for issue lifetime," *arXiv preprint arXiv:1702.07735*, 2017.

[15] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: an empirical study of commercial software projects," in *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 2013, pp. 1042–1051.

[16] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?" in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 207–210.

[17] P. Anbalagan and M. Vouk, "On predicting the time taken to correct bug reports in open source projects," in *2009 IEEE International Conference on Software Maintenance*. IEEE, 2009, pp. 523–526.

[18] G. Canfora, M. Ceccarelli, L. Cerulo, and M. Di Penta, "How long does a bug survive? an empirical study," in *2011 18th Working Conference on Reverse Engineering*. IEEE, 2011, pp. 191–200.

[19] W. H. A. Al-Zubaidi, H. K. Dam, A. Ghose, and X. Li, "Multi-objective search-based approach to estimate issue resolution time," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2017, pp. 53–62.

[20] E. Kouzari, L. Sotiriadis, and I. Stamelos, "Process mining for process conformance checking in an oss project: An empirical research," in *IFIP International Conference on Open Source Systems*. Springer, 2018, pp. 79–89.

[21] E. Kouzari and I. Stamelos, "Process mining in software events of open source software projects," in *2nd International Symposium & 24th National Conference on Operational Research, HELORS*, 2013, pp. 25–27.

[22] W. Poncin, A. Serebrenik, and M. Van Den Brand, "Process mining software repositories," in *2011 15th European Conference on Software Maintenance and Reengineering*. IEEE, 2011, pp. 5–14.

[23] M. Gupta, "Nirikshan: process mining software repositories to identify inefficiencies, imperfections, and enhance existing process capabilities," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 658–661.

[24] W. M. van der Aalst, H. A. Reijers, A. J. Weijters, B. F. van Dongen, A. A. De Medeiros, M. Song, and H. Verbeek, "Business process mining: An industrial application," *Information Systems*, vol. 32, no. 5, pp. 713–732, 2007.

[25] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[26] V. Van Asch, "Macro-and micro-averaged evaluation measures [[basic draft]]," *Belgium: CLiPS*, pp. 1–27, 2013.

[27] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[28] P. S. Kochhar, T.-D. B. Le, and D. Lo, "It's not a bug, it's a feature: does misclassification affect bug localization?" in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 296–299.