Analysis of Software Repositories Using Process Mining

Roohi Arora and Anchal Garg

Abstract Software repositories such as Issue Tracking Systems (ITS) log bug history that contains important information about the activities followed during bug resolution. Mining of ITS can provide useful insights of such processes. This study uses process mining to extract knowledge from the event logs recorded in bug history report of ITS Monorail to determine the as-is process model. This study will help in identifying the kind and source of inefficiencies and inconsistencies and eventually help improving the software bug resolution process.

1 Introduction

Software repositories such as Issue Tracking Systems (ITS), peer code review systems, and version control subsystems store trails of different software development activities. The bug history of the ITS database includes unique identification number of the bug, its publisher, priority, status, state, different dates, summary and labels, programmers working on a bug and operating system in which it occurs. ITS follows a defined sequence of activities through its lifecycle from the time it is first reported to when it is closed, viz. unconfirmed, untriaged, assigned, started, fixed, reopened, and verified. Mining ITS will help us identify the sequence of activities actually followed by programmers while resolving a bug and thus help us in identifying the lacunas and improve the process of resolving the issue. Such study can be a step to improve the overall process quality for best software engineering practices. Process Mining is one of the techniques that can provide useful insights into the quality of the process and aid as an assessment tool of ITS process.

R. Arora (✉)
Accenture Solutions Private Limited, Pune, Maharashtra, India
e-mail: roohi9arora@gmail.com

A. Garg
Department of Computer Science & Engineering, Amity School of Engineering & Technology, Amity University Uttar Pradesh, Noida, Uttar Pradesh, India
e-mail: agarg@amity.edu

S.C. Satapathy et al. (eds.), *Smart Computing and Informatics*, Smart Innovation, Systems and Technologies 78, https://doi.org/10.1007/978-981-10-5547-8_65

The interest in the area of Mining Software Repositories (MSR) has gained significant importance by researchers over past few years. Poncin [1] first used process mining to discover process map by combining events from different repositories. Sunindyo [2] analyzed bug reporting system of Red Hat Enterprise limited and performed conformance checking for process improvement. Gupta and Surekha [3] mined the bug report history of open source project such as Firefox and Mozilla to study process inefficiencies. Gupta et al. [4] analyzed multiple repositories such as ITS, peer code review subsystem and version control subsystem to improve software defect resolution process. Similarly, a study was conducted by Zimmermann et al. [5] on Windows operating system to categorize probable causes of reopened bugs.

Lately, a lot of work had been done by researchers to process mine software configuration management activities [6, 7] but no work has been performed on the ITS of Google Chromium. Hence, it is worthwhile to study the process from this repository and derive useful insights. One of the widely accepted web browser viz Google Chrome, derives its source code from Chromium. This work conducts in depth analysis of the event logs of ITS repository of Google Chromium.

The motivation behind this study is to mine the bug history of ITS to identify the bug resolution process. This study will help us provide useful insights into the process which can aid in identifying whether the sequence of activities is performed as desired or are there any inefficiencies and inconsistencies. In case of lacunas, the study will help us determine the kind and source of inefficiencies, thus providing us a roadmap on resolving such concerns in future. The kind of analysis will might help in improving the efficiency and productivity of the process.

This paper is structured as follows: Sect. 2 gives the details of the dataset adapted for conducting the study, Sect. 3 discusses the details of the work done, Sect. 4 presents and discusses the results of the experiment.

2 Experimental Dataset Adapted

This paper uses Google Chromium's Monorail Issue Tracker for the study. The details of experimental dataset used for the study are given in Table 1.

Table 1 Details of the dataset used for experiment

Attribute	Value
First issue creation date	8 April 2015
Last issue creation date	18 February 2016
Date of extraction	28 February 2016
Total issues extracted	23,826
Range of issues	475,000–525,000
Total number of events	77,752

3 Simulation Model

This research framework has four different stages Data Extraction, Data Cleaning, Process Mining and Process Analysis.

Data Extraction: Python script was used to extract trails from the ITS of Google Chromium. Details of issues extracted are given in Table 1. A trail contains all the information associated to a particular issue. Open and closed issues are considered for analysis in this study.

Data Cleaning: Only relevant attributes such as issue ID, status, updated time, closed date, and actors (resources) were selected for analysis. The comments that depicted the progress of the work done on the issues were extracted and appended. Missing values (NULL) and inconsistencies if any were removed. An initial state *REPORTED* was created for all the issues based on attributes such as Published Time and Owner. Finally, the final event log was obtained with attributes such as timestamp, issue id, activity and resource associated with that issue.

Process Mining: Process Mining was performed for analyzing the event logs of ITS using DISCO tool. Generated event log were mined from multiple perspectives to derive insightful process maps of the ITS process Discovered process maps were further analyzed to scrutinize the performance of the developers, productivity of the process and list the shortcoming so that its overall performance and efficiency could be enhanced. Process Mining ITS of Google Chromium will assists developers to become familiar with the development process, its progress, obstacles and challenges. It will also reveal what developers are actually doing in contrast to what they claim they have been doing.

Process Discovery: The discovered process map is presented in Fig. 1. Edges between the activities represent transition between the activities in a process. Frequency of transition is indicated on the edge. Darker edge corresponds to more frequent transition in contrast to lighter edge signifying less frequent transition.

4 Results and Analysis

The process map generated for the ITS repository was analyzed, initially to determine the frequency of each activity. Table 2 present the results of activity frequency analysis.

Activity Frequency Analysis: For all 23,826 issues *Reported* is the initial activity. A significant number of bugs have been fixed as reported by *Fixed* activity. However, the frequency of issues being *Verified* is apparently low indicating that issues are being fixed but are not being *Verified* either by the tester or by the person who initially *Reported* that issue. This probably indicates a loophole in the system. Thus, once the issue is *Fixed*, it should prompt its owner or the tester to verify the resolved issue and assign *Verified* status, if it works well or *ReAssigned* if it is unresolved. The frequency of *Wontfix* is comparatively high which is undesirable.

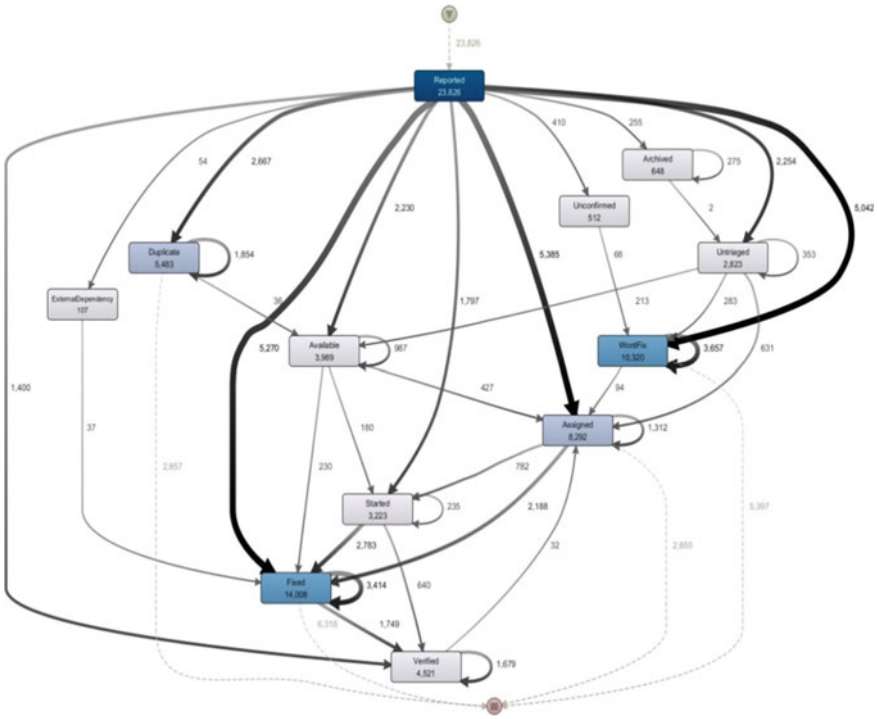

Fig. 1 Process map of ITS of Google Chromium with labeled absolute frequency

Table 2 Activities and their relative frequency

Activity	Relative frequency (%)	Frequency
Reported	30.64	23,826
Fixed	18.01	14,008
Wontfix	13.27	10,320
Assigned	10.67	8,294
Duplicate	7.05	5,483
Verified	5.81	4,521
Available	5.13	3,991
Started	4.14	3,223
Untriaged	3.63	2,824
Archived	0.83	648
Unconfirmed	0.66	512
Ext. dependency	0.14	107

A detailed analysis revealed that the programmers are reworking on the issues that cannot be *Fixed*. Further, many activities developers are reworking on the issues that have been resolved earlier as indicated by *Duplicate* status. Thus, during assignment additional information should be provided for issues already marked

Duplicate. This would save the efforts of developers and thus reduce organizational cost. Many issues are still *Available* after being assigned priority and being *Confirmed* which is again not an efficient practice. *Untriaged* status of the issues is a matter of concern as the developers may be fixing issues having low priority when there high priority issues lined up. Few issues have *Unconfirmed* status, but this can be mended if it is *Confirmed* as soon as an issue is *Reported*, sooner it is *Confirmed* and *Triaged* sooner it is in the hands of developer. Very few issues have *External Dependency* as their status which is good for our process.

Event Frequency Analysis: An analysis of event frequency revealed various loopholes. First, 3/23826 events have frequency 1, indicating that the issue is *Reported* but no further event/activity is performed on it and that is a matter of concern. Although the number is too low but such a practice should be avoided as the issue may be of high priority and severity which might reduce the quality of the product or may appear in later stages of development and cost huge to the development team and organization. Second, in few cases *Fixed* issues are being reassigned. Such issues are reopened several times as the root cause of the issues might not be clear. This has caused delay of 273 days. If we analyze the event frequency of closed issues which have been *Fixed* and *Verified* there do 8714 cases with 30,668 events constitute 36% of the total cases and 42% of the total events. However, many of these issues are restarted and reassigned and do not adhere to the process but they are eventually *Fixed* and *Verified*. Numbers of events range from 3 to 16.

Variant Analysis: For 23,826 cases, 757 unique variants were obtained. Most frequent variants are listed in Table 3.

Top 4 variants cover 38% of the total cases and 33% of the total events. Majority of the variants have self loop, i.e., the activity is repeated twice. It may be assumed that the developers might not be sure of their decision that is delaying the final issue resolution. Variant analyses reveal that many issues are being reassigned status (*Fixed*/*Wontfix*/*Duplicate*) after 300 or more days.

Analysis of Issues with Different Priority: Issues in chromium ITS needs to triage with different priorities depending upon their effect the software evolution process. Issues can be assigned one of the following priorities:

Priority 0: Requires Quick resolution
Priority 1: High impact issues
Priority 2: Low impact issues
Priority 3: Can be resolved anytime

Table 3 Most frequent variants in the process

S. No.	Variant	Percentage (%)
1.	*Reported→Wontfix→Wontfix*	37.78
2.	*Reported→Fixed→Fixed*	28.24
3.	*Reported→Duplicate→Duplicate*	19.19
4.	*Reported→Assigned*	14.79

Further analysis was conducted to identify how the developers handled the issues with different priorities. We looked for the answers to following questions: (i) Are the issues served on the basis of their priority? (ii) Do developers fix problems as per their importance? (iii) To what extent the process followed comply with the specified process.

An analysis indicates that the issues are resolved as per their priority levels. Priority 0 and Priority 1 issues are resolved quickly. However, it is also observed that though the issues are assigned *Fixed* status after resolution, not all of them are *Verified.* Mean case duration of such cases is 67.1 that are undesirable. On the other hand, issues at Priority 2 and Priority 3 have a delayed response from the resolvers. Although these issues are not of much importance to the current release but they can affect functionality in the upcoming releases. Mean case duration of issues with Priority 2 was 64.9 days and that with Priority 3 is 84.7 days. Analysis also reveals that 23 issues were assigned invalid priority such as 4, 21, etc. This indicates that there should be a check while setting the priority of the issue.

The analysis provides vital insights into the bug resolution process. This study has falsified our conception that the process followed during bug resolution is satisfactory. As-is process model revealed that the sequence of activities followed is not as assumed. The developers either skip or duplicate some of the activities. The issues once *Fixed* were again *Assigned* thus wasting efforts of the software developers and adding to the organizational cost. Few cases were trapped in between causing delays in resolving the issue. Some cases were *Reported* but not *Assigned* to anyone. Few issues were not assigned priorities causing a risk of an important and severe issue not being resolved on time. The analysis indicates inefficiencies and inconsistencies in the process thus adding to the expenses of the organization. Such analysis may be used by Chromium team to remove the inefficiencies and improve the process. This will both save the efforts of the development team and help manage the organizational cost and reputation.

5 Conclusion

In this paper, detailed analysis has been performed on the ITS of Chromium. The paper provides insights on how process mining of software repositories can help unveil useful insights into the software development activities. If the organizations adopt an approach to continuously monitor their software repositories, they can gain useful insights of their processes. Such practice would improve the quality of the processes thereby reducing overall operational cost of the software development that is incurred by the organizations.

References

1. Poncin, W., Serebrenik, A., Brand, M.V.D.: Process Mining Software Repositories. In: 15th European Conference on Software Maintenance and Reengineering, pp. 5–14. IEEE, Germany (2011).
2. Sunindyo, W. Moser, T. Winkler, D., Dhungana, D.: Improving Open Source Software Process Quality Based on Defect Data Mining. In: Biffl, S. Winkler, D. and Bergsmann, J. (eds.) LNBIP, vol. 94, pp. 84–102. Springer Berlin Heidelberg (2012).
3. Gupta, M., Surekha, A.: Nirikshan: Mining Bug History For Discovering Process Maps, Inconsistencies and Inefficiencies. In: International Conference on Software Engineering, pp. 1. ACM, Chennai (2014).
4. Gupta, M. Surekha, A., Padmanabhuni, S.: Process Mining Multiple Repositories for Software Defect Resolution from Control and Organizational Perspective. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 122–131. ACM, Hyderabad (2014).
5. Zimmermann, T. Nagappan, N. Guo, P.J., Murphy, B.: Characterizing and Predicting which Bugs Get Reopened. In: Proceedings of the 34th International Conference on Software Engineering, pp. 1074–1083, IEEE, Zurcih (2012).
6. Kagdi, H., Collard, M. L., Maletic, J. I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of software maintenance and evolution: Research and practice. 19(2), 77–131 (2007).
7. Chen, T. H., Thomas, S. W., Hassan, A. E.: A survey on the use of topic models when mining software repositories. Empirical Software Engineering, 21(5), 1843–1919 (2016).