Process Mining for Process Conformance Checking in an OSS Project: An Empirical Research

Elia Kouzari(✉), Lazaros Sotiriadis, and Ioannis Stamelos

Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
ekouzari@csd.auth.gr

Abstract. With almost 20 years of research, Process Mining can now be considered to be in a mature phase allowing its application to a variety of sectors. In this article, the bug closure process that is followed by a community of an open source software project is investigated in order to perform process conformance checking. Actual data that reveal the process steps have been extracted from the project's Bugzilla database and have been used as input in Disco process mining tool. The data includes extracted information for more than 19,000 bugs for the past 15 years in a csv form, formatted appropriately to construct an event log suitable for process mining. The extracted models have been compared to the process described in the project's blogs and wikis by the community. The same models are also compared to the bug closure process that Bugzilla suggests to be used by the projects using this software for bug tracking purposes. The findings reveal that indeed the process followed in the OSS project is very similar to the declared one but variations do occur under specific circumstances. However, the process is not identical to the one proposed by Bugzilla suggesting that each OSS project can customize its processes in order to better address the needs of the project and the community. This empirical research highlights the importance of process mining in OSS projects in order to investigate the processes followed and identify outliers helping to standardize and improve the processes and enhance the collaboration among the members of the communities.

Keywords: Open source software · Process mining
Open source communities · Software event logs · Process conformance

1 Introduction

Scientific research has been focusing in the field of Process Mining, developing platforms, tools and algorithms for almost 20 years now. Process Mining is extensively applied in a variety of fields and sectors like healthcare, insurances, software etc. [1–5]. Dozens of algorithms can be applied in the context of process mining either on a standalone basis or through the platforms that have been designed to facilitate this.

The majority of applied research in this field focuses on process discovery since this minimizes the cost of understanding the current 'As-is' process [6]. However, a lot of

I. Stamelos et al. (Eds.): OSS 2018, IFIP AICT 525, pp. 79–89, 2018.
https://doi.org/10.1007/978-3-319-92375-8_7

interest arises in the aspect of process conformance as well. According to van der Aalst [7] process discovery is the technique that takes an event log and produces a process map explaining the behavior recorded in the log. On the other hand, process conformance is the procedure where an existing process model is compared with an event log of the same process to indicate whether the reality, as recorded in the log, meets the proposed model and vice versa.

Kouzari and Stamelos [8] suggested that the application of process mining in open source software could reveal not only the variety of processes followed by open source communities but it could also help standardize or improve core activities like the way tasks are shared and bugs are closed. Given the fact that in Open Source Software there is a huge amount of process-relevant data publicly available online, OSS communities are a great opportunity to discover and analyze software processes. These data can be usually extracted from mailing lists, discussion forums, source repositories and binary release sections [9].

In this article, the authors proceed to process conformance checking of the bug reporting and closing process of a large Open Source Software project by examining the documentation of the project and extracting the process model that is believed to be followed against the 'As-is' process that is extracted from the event log. In addition, this process is also compared with the general Bugzilla guidelines for bug reporting and resolution.

The rest of the article is structured as follows: Sect. 2 describes the background of this work and poses the research question. Section 3 explains the methodology followed and Sect. 4 presents the findings of the research. Finally, Sect. 5 contains the discussion, conclusions and future work.

2 Background Work

2.1 Bugzilla

Bugzilla is an open source software system for bug tracking. It is currently the most widely used bug tracking system [10]. This bug tracker allows open source communities to handle the discovered bugs by facilitating the communication of problems effectively throughout the data management chain. This ensures that each reported bug is stored in the project's Bugzilla database along with each detail regarding its state and the steps taken or not towards its resolution.

The lifecycle of a bug in Bugzilla is illustrated in Fig. 1. This workflow can be customized to meet the needs of every community and every project using it as a back-tracking system. In Fig. 1 only the default bug statuses are shown.

The main process followed in Fig. 1 is as follows: When a bug is first reported in the system, its status changes to "UNCONFIRMED". The bug remains in this state until it receives a specific number of votes by community members that is indeed reproduced. By the time the votes are sufficient, the bug status changes to "NEW". A bug can automatically at first be set to "NEW" when the user/developer reporting it has the right to change its status. Following, the bug can be assigned to a specific developer or can be left

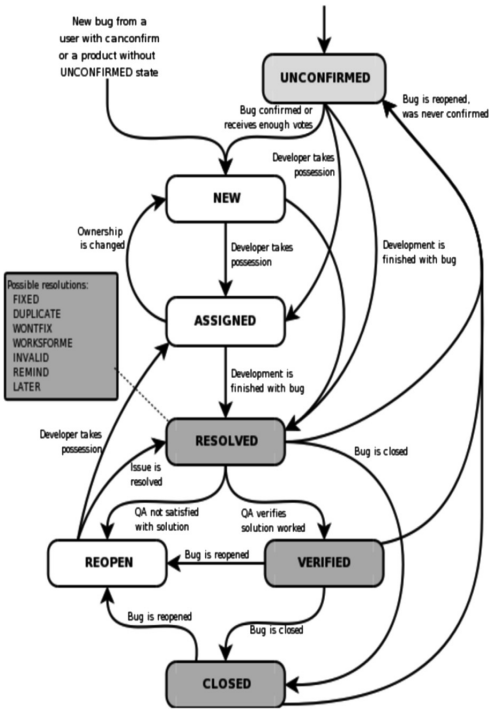

Fig. 1. The lifecycle of a bug in Bugzilla (source: https://wiki.documentfoundation.org/QA/Bugzilla/Fields/Status)

open for anyone to resolve it. At this moment, the status of the bug is set to "ASSIGNED". The developers resolve the bug, and a patch is sent for Quality Assurance (QA). When the QA tests are successful, the bug becomes "RESOLVED" and "VERIFIED". In case QA tests fail, the bug can be set to "REOPEN" and short after this to "ASSIGNED" or "RESOLVED" once again. A bug must be "CLOSED" by the person who first reported it. It is obvious that a lot of variations are also possible depending on the nature of the bug and the authority of the person who first reports it [11].

2.2 Koha Open Source Integrated Library System

Koha is a web-based open source ILS written in Perl and distributed under GNU General Public License[1]. With a very active community of developers around the world, Koha was first released in 2000 and since then it has gained wide acceptance, as it is utilized in hundreds of organizations (both in the governmental and private sector) around the world [12]. Koha interacts with a MySQL database and supports a variety of

[1] https://koha-community.org.

library activities with the most widely used being cataloguing, acquisition, circulation and administration [13]. Being able to be installed and function in all Operating Systems (Linux, Unix, Windows, MacOS), Koha offers a very configurable and adaptable user interface, allowing each organization to set it up accordingly to the procedures and policies followed in each case.

For bug tracking purposes, Koha maintains its own Bugzilla database. Since the project has an active community, there are wikis[2], IRC[3] (chats) and blogs[4] that describe the procedures followed in the project and facilitate the communication and coordination of the community.

The Koha blog describes the bugs workflow process as follows: A user or a developer can report a bug through Bugzilla. Everyone should file a bug report, even developers who intend to immediately fix one. The Koha project does not actively use the Priority field in Bugzilla. However, severity field is indicated to be important. For this reason, along with the bug report the user has to also assign the severity of a bug. For unclear situations "normal" severity is suggested. Once a bug is reported it is assigned to a developer. At this stage, a bug can be left idle, if no one takes action on it, and it can also be reassigned. Once a person decides to work on a bug, he has to first accept it and update its status from New to Assigned. When a solution to a bug is available, a patch is submitted turning the status of a bug to "Patch-Sent". The Koha Release Manager evaluates the submitted patch and if the patch works it is marked as "pushed". A resolved bug is altered from "Assigned" to "Resolved - Fixed". Koha Wiki clearly suggests that the user should review the Bugzilla Bug Writing Guidelines prior to submitting a bug to Koha.

Based on all these characteristics and its large and active community of developers and users, Koha is considered at the time as an ideal candidate in the context of this research.

2.3 Research Question

In Sect. 2.2 the Koha bugs workflow is presented as declared in the official pages of Koha project. In terms of process mining, the abovementioned process is considered the "As-Is" process followed. Since Koha is an open source project its data and code are available through the tools used by the community. It would be of great interest to mine the actual process followed to resolve the bugs using data from the project's Bugzilla database and perform process conformance checking. As a result, the research question investigated in this paper is the following:

RQ: Does the community of Koha Open Source ILS conforms to the bug resolution process described in the project's pages or not? If not, what is the actual process followed?

[2] https://wiki.koha-community.org.

[3] https://koha-community.org/get-involved/irc/.

[4] https://www.myacpl.org/koha/.

3 Methodology

Anyone can register in the Bugzilla page for Koha project in order to view all the bugs listed along with the history of each bug. The main Bugzilla page illustrates all current bugs with their current state. In order to get all the actions performed for the resolution of a bug, one must locate a bug and then using the bug id provided he may visit the specific page that illustrates in tabular format the actions taken so far for it.

The methodology shown in Fig. 2 was used to gather all information for all bugs listed in Koha project so far for cleaning/transformation of data for process mining.

Fig. 2. The methodology followed in this article for process conformance check

3.1 Locate Relevant Data

First, the relevant data was located in the Bugzilla database. Using Bash scripting and Wget tool, the data was extracted in html form and appended in a single html file. This procedure gathered historical data for 19,311 bugs between 15/06/2002 and 14/12/2017.

3.2 Data Preparation

Html2text was used to remove any text formatted from the gathered data. In addition, to be able to further format the data in a csv form and to create a tabular representation suitable of an event log for process mining, further scripting was required (grep, icony, recode, sed, tr, cat, uniq, sort, echo). As a result, a csv file with 359,395 records was created containing information for 19,311 bugs.

3.3 Clean Data

For each record, the following columns were present: Event_Id, Bug_Id, Bug_Description, User_Email, Action_datetime, Action_Type and Action_Data. Although this file contained all the required columns to be used as an event log [14], further cleaning of the file had to be performed in order to keep those data that would reveal the process followed for bug resolution. While the first 5 columns reveal their role, columns Action_Type and Action_Data were used to keep the most important information. For every variable mentioned in column Action_Type, a different set of values was used in column Action_Data. After analysis of each variable in the Action_Type column and taking in mind the described process followed for bug resolution in Koha Blogs and Wikis, the csv file was filtered to keep the records for the following variables of Action_Type column: Status, Priority, Severity, Resolution.

Table 1 below presents the Action_Data values per Action_Type selected for filtering.

Table 1. Action_Data values per Action_Type variable

Action_Type variable	Corresponding Action_Data values
Priority	P1, P1-high, P2, P3, P4, P5, P5-low, PATCH-Sent, PATCH-Sent-P5
Resolution	–, DUPLICATE, FIXED, INVALID, LATER, MOVED, REMIND, WISHLIST, WONTFIX, WORKSFORME
Severity	blocker, critical, enhancement, major, minor, newfeature, normal, trivial
Status	ASSIGNED, BLOCKED, CLOSED, REOPENED, RESOLVED, UNCONFIRMED, VERIFIED Failed QA, InDiscussion, Needs Signoff, NEW, Passed QA, Patch doesn't Apply, Pushed by Module Maintainer, Pushed for QA, Pushed to Master, Pushed to Stable, Signed Off

Finally, a csv file with 97,372 records for 19,311 bugs was extracted that was used as an event log for Process Mining. The rest of the steps that concern the process mining and the process conformance check are presented in Sect. 4.

4 Findings

4.1 Process Mining

For Process Mining, Disco[5] process mining tool was used. Initially the event log was used as input to Disco. Prior to the process mining, the columns of the event log had to be assigned as Case_ID (Bug_Id), Resource (User_Email), Timestamp (Action_Date-time), and Activity (Action_Type and Action_Data). Event_ID and Bug_Description were ignored as they had no role in affecting the bug resolution process.

The event log was then used for process mining revealing a "spaghetti model" that it was impossible to highlight useful information regarding the process mined.

The extracted model highlighted the most used statuses but it was hard to analyze and compare to the suggested process. To extract safest conclusions, 47,9% of the most common activities of the diagram were filtered creating a new, clearer process model.

Despite the second process model was significantly more clear than the first one, it was clearly affected by the status of the first 283 bugs in the event log. This was proved by Status/Closed that was shown to be the first step of the process (and the last) for the majority of the cases. With further analysis of the event log, the authors noticed that for the first 283 bugs only a record indicating that a bug's status was set to "CLOSED" existed. This could affect the actual process followed since no history for the resolution process of these bugs was available. In the next session, this is further explained.[6]

[5] https://fluxicon.com/disco/.

[6] Additional information on the first process models extracted by the event log and discussed in this section are available online at http://switch.csd.auth.gr/.

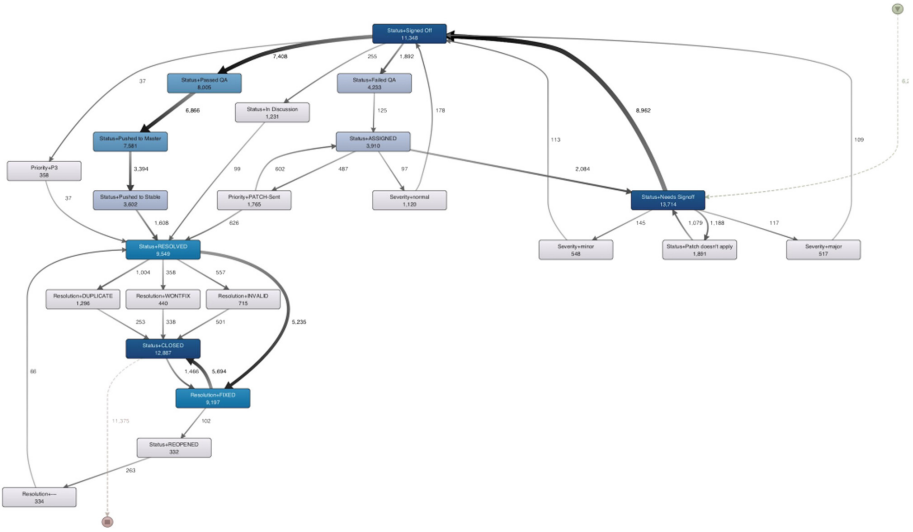

Fig. 3. The process model extracted when the first 283 bugs are removed from the event log

A new event log, ignoring the first 283 bugs was used as a new input with the same parameters in Disco. The process map created for the new event log is shown in Fig. 3.

In the diagram of Fig. 3 a clearer process is extracted. The dark process steps indicate the most common statuses and the darker arrows indicate the most frequent transitions from these process steps. One can identify from this diagram that the most common process path followed is the following:

Needs Sign Off -> Signed Off -> Passed QA -> Pushed to Master -> Pushed to Stable -> Resolved -> Resolution/Fixed -> Closed.

All of the process steps mentioned are associated with the value "Status" of the Action_Type field, except from Fixed that is associated with the value "Resolution" of the Action_Type field.

The same event log was inserted into Disco, keeping only dimensions "Status" and "Resolution" of the column "Action_Type". To extract a more precise model, focused on the resolved bugs, an endpoint filter was applied to the event log. The end event value was set to "FIXED" keeping 10% of the cases and 17% of events. As a result, the process model of Fig. 4 was produced.

In this model, the process is almost identical to Fig. 3 with an exception towards the end of the process. However, the bug resolution process remains unaltered even though a significant number of cases are not included indicating that the community is consistent to the steps taken to resolve a bug.

4.2 Process Conformance Check

By observing Figs. 3 and 4 one can conclude that the discovered process is very similar to the one described in the community's blogs and wikis. By these means we can

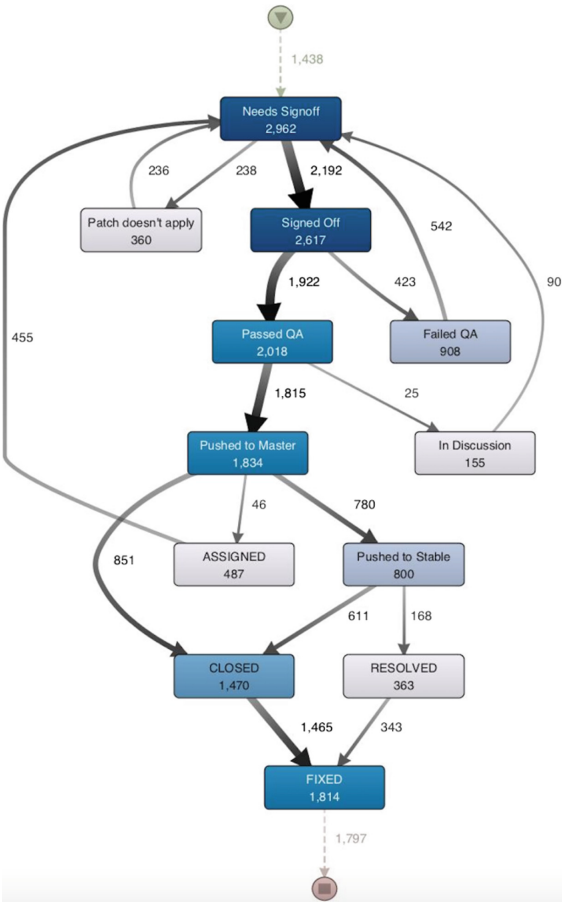

Fig. 4. A process model extracted for all bugs that are FIXED

clearly say that the Koha community does conform to the process suggested for bug resolution.

Focusing on Fig. 3, where the model contains more cases since it includes all the bugs and not just those that have been fixed, it is obvious that there is a main process, identical to the one proposed by the community but this is not exclusive. There are some exceptions, some process variations that are obvious. Disco has the ability to reveal these paths along with their frequency. As a result, Table 2 presents the 5 most frequent process variances that do not follow the proposed guidelines for bug resolution.

4,14% of the time, a bug is automatically Resolved and Closed. This happens either because it is an old bug and no other relevant historical data is available, either because the person who identifies a bug automatically fixes it. However, it is recommended by the community to report each and every bug even if that means the bug is instantly resolved. For this reason, the abovementioned path does not conform to the bug resolution process.

Table 2. The 5 most frequent process variances that do not conform to the process guidelines

Process Variance	Frequency
RESOLVED -> FIXED -> CLOSED	4,14%
PATCH-Sent -> RESOLVED -> FIXED -> CLOSED	1,64%
ASSIGNED -> PATCH-Sent -> RESOLVED -> FIXED -> CLOSED	1,42%
Needs SignOff -> Failed QA	0,37%
ASSIGNED -> RESOLVED -> FIXED -> CLOSED	0,32%

The same seems to happen with the rest of the process variances. In all of the variances, either the bug is not set to Needs SignOff from the beginning of the bug resolution process or several process steps are omitted until the bug is set to Closed.

In the Discussion session below this is further discussed and compared to the Bugzilla suggested process for bug resolution.

5 Discussion and Future Work

The event log created by the Bugzilla database of Koha OSS ILS revealed that the community does follow in practice the bug resolution process described in the official documentation of the project. However, there are some cases where this is not true. It was mentioned earlier that the first 283 bugs did not include any other information rather than a record indicating they were closed. This indicates that 15 years back, when the project was not in a mature phase the community was using another bug resolution process. This might have happened due to a limited number of bugs or due to the lack of mature bug tracking tools. Starting from bug with bugID = 284 there is precise information about the bug and all of its states throughout the resolution process.

At the same time, one can observe that although the community follows a specific process for bug resolution, there are some process variances that do not conform to the proposed guidelines. This illustrates that in an open source software community there is freedom to act and modify procedures that are not as strict as in proprietary software. A developer might decide to act based on his knowledge and experience in a specific situation.

Comparing the bug resolution process of Koha with the proposed guidelines for bug resolution by Bugzilla it is obvious that there are a lot of differences. With a closer look a correlation is identified in some of the states. The RESOLVED status of Bugzilla corresponds to Needs SignOff of Koha and VERIFIED status of Bugzilla corresponds to Signed Off of Koha. This reveals that open source software tools can be freely customized to address the needs of each community of users and developers. At the same time, each OSS community is free to investigate how OSS tools can be modified to be used in favor of their own processes.

As stated earlier, Koha is a very active and serious community that is well organized. In addition, the project contains excellent documentation that the users can address in order to find answers to their questions. For even better communications there are IRC chat rooms where users and developers can further discuss any related

issues. The documentation of the project and the good communication between the members of Koha community is illustrated in the process conformance check performed in this article.

However, other communities, less structured and active might face a variety of problems in the management of their processes. Process mining not only can highlight problems in the existing processes followed by OSS communities but it can also use projects like Koha as a benchmark to reveal aspects of processes that can be further improved in other projects. Nonetheless, the extracted process models can be used for predictions regarding the future of a given project in terms of survival, software quality and maturity.

Recently, Bugzilla released newer documentation containing a newer version of the lifecycle of a Bugzilla bug. It would be of great interest to investigate projects following the new guidelines for bug resolution and also see which of the projects currently following the former guidelines will decide to modify their processes and the effect this will bring to process efficiency. Future work includes further empirical research on process conformance for OSS projects. Especially for bug management, the newly released Bugzilla process needs to be taken into account.

References

1. Rojas, E., Munoz-Gama, J., Sepulveda, M., Capurro, D.: Process mining in healthcare: a literature review. J. Biomed. Inf. **61**, 224–236 (2016)
2. Partington, A., Wynn, M.T., Suriadi, S., Ouyang, C., Karnon, J.: Process mining for clinical processes: a comparative analysis of four Australian hospitals. ACM Trans. Manag. Inf. Syst. **5**(4), 1–19 (2015)
3. Delias, P., Doumpos, M., Manolitzas, P., Grigoroudis, E., Matsatsinis, N.: Clustering healthcare processes with a robust approach. In: 26th European Conference on Operational Research, November 2015, pp. 1–6 (2013)
4. De Weerdt, J., Schupp, A., Vanderloock, A., Baesens, B.: Process mining for the multi-faceted analysis of business processes—a case study in a financial services organization. Comput. Ind. **64**(1), 57–67 (2013)
5. Rubin, V.A., Mitsyuk, A.A., Lomazova, I.A., van der Aalst, W.M.: Process mining can be applied to software too! In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, p. 57. ACM, September 2014
6. Rozinat, A., Gunther, C.W.: The added value of process mining. BPTrends (2014). https://www.bptrends.com/the-added-value-of-process-mining/. Accessed 16 Jan 2018
7. Van Der Aalst, W.: Data science in action. Process Mining, 2nd edn, pp. 25–52. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4_2
8. Kouzari, E., Stamelos, I.: Process mining in software events of open source software projects. In: 2nd International Symposium & 24th National Conference on Operational Research, HELORS 2013, 25–27 September 2013, Athens, Greece (2013)
9. Jensen, C., Scacchi, W.: Data mining for software process discovery in open source software development communities. In: Proceedings of Workshop on Mining Software Repositories, pp. 96–100, May 2004
10. Barnson, M.P., Steenhagen, J., Weissman, T.: The Bugzilla Guide-2.17.5 Development Release. The Bugzilla Team (2003)

11. Akbarinasaji, S., Caglayan, B., Bener, A.: Predicting bug-fixing time: a replication study using an open source software project. J. Syst. Softw. **136**, 173–186 (2018)
12. Kouzari, E., Stamelos, I.: Process Mining applied on library information system usage-A case study (2017). Manuscript submitted for publication
13. Macan, B., Fernandez, V.G., Stojanovski, J.: Open source solutions for libraries: ABCD vs Koha. Program **47**(2), 136–154 (2013)
14. Van Der Aalst, W.M., Dustdar, S.: Process mining put into context. IEEE Internet Comput. **16**(1), 82–86 (2012)