Bug Reports Evolution in Open Source Systems

Wajdi Aljedaani[1] and Yasir Javed[2,3]

[1] Al-Kharj College of Technology, Al-Kharj, Saudi Arabia,
[2] Network Security Research Group, Faculty of Computer Science and Information Technology,Universiti Malaysia Sarawak,
[3] Prince Sultan University, Riyadh, Saudi Arabia
`waljedaani@tvtc.gov.sa, yjaved@psu.edu.sa`

Abstract. Open Source Software communities usually utilize open bug reporting system to enable users to report and fix bugs. In addition, the lifetime of most open source system stays for long periods of time. In this work, we comprehensively examine the evolution of bug reports in four different open source systems from various languages. The selected project are analyzed since 2004 in order to find how many bugs are reported compared to their resolution. We report our results and some recommendations to the open source community.

Keywords: Bug repository, bug report, open-source system bugs,

1 Introduction

Large software systems are becoming essential to people daily lives. A big portion of these software systems is attributed to their maintenance. Several studies demonstrated that more than 90% of the software development cost is dedicated to maintenance and evolution tasks [1]. With a huge number of software systems, understanding and fixing bugs come to be a puzzling task. As bug reports (BRs) can facilities useful information to fix bugs, researchers have focused on how to leverage them to ease the task of fixing these bugs. The bug reports quality is very essential in easing the fix for that bug. Hooimeijer and Weimer [2] formed a descriptive model for bug reports lifetime. Bettenburg et al. [3] examined what makes a good bug report. Several researchers also explored different aspects of bug reports such as duplicate detection [4], fixer recommendation [5], feature prediction [6], and bug localization [7]. Bhattacharya et al. [8] investigated the fixing time of bug in Android apps. They studied both quality of bug reports and the bug-fixing process. A large amount of the software maintenance is attributed to bug fixing. Normally, bugs are reported, fixed, verified and closed. Nevertheless, in some scenarios, bugs have to be re-opened, which will increase maintenance costs, degrade the overall user-perceived quality of the software and lead to unnecessary rework.

Bug fixing is a very important essential activity in software development and maintenance. Bugs are reported, recorded, and managed in bug tracking systems such as Bugzilla. A bug report contains many fields, which provide insights into the bug and help in fixing it. Open source software (OSS) development is a collaborative large team activity contributed by many globally distributed developer community. Many open source software teams depend on freeware tools to plan, code, test, track-report-fix bugs and market product(s). Open source projects embrace open bug repositories such as Bugzilla to support its development and maintenance in managing and maintaining bugs. Geographically-distributed users and developers can report software bugs. This will help in addressing these bug reports. In this way, open source software is iteratively developed and the quality of the produced software can be improved [4]. A bug triage is the person who will decide to whom a bug reported is assigned.

There is a number of reasons for bugs in open source systems but their resolution requires a proper attention from developers and contributors of the project. In order to understand how bugs are reported over time and how bugs are resolved the authors tend to perform this research. This research will also highlight the time taken to resolve the bugs that might vary from days to months but in some case more than a certain number of years. The authors will also propose a model to avoid a situation where a low priority bug always sits unresolved. Sometimes bugs are often misreported and thus require no attention, this study will also look into how many of this kind of case exists.

2 Literature review

A number of researchers have studies the qualitative analysis of bug repositories on a survey, and open software system [9], [10], [11], [12]. Zaman et al. [13] examined bugs performance in two software systems Firefox and chrome to study the collaboration among project team members to identify performance bugs. Their investigation centered done on Firefox for bugs that on Firefox and Firefox core component, and the entire bug reports that were submitted to chrome system.

Banerjee et al. [9] studied two bug repositories of Mozilla and Eclipse in order to compare the similarities and differences between each of them. Their goal was to evaluate user behavior, the structure of the repository and the type of bugs. They also looked into the duplicating groups as well as the frequency of the reported bugs. Our study is similar to their in the frequency of reported bug. They concentrated only on two open source software Mozilla and Eclipse, however, we studied four open source software Ant, K3B, Kate, and OpenSSH. We analyzed the evolution of bug repository for these projects over the time reporting bug per year. Bugs evolve over time and in order to make them maintainable and making sure that they stay with passage over time require the resolution of bugs as mentioned by [14]. The authors in [15] worked in the classification of bugs in order to find out the impact of bugs but were limited to manual checking and only Apache projects. The authors in [16] have worked on the system of bounty for awarding problem solvers to award them with some gifts or monetary values. It also highlights the need for funding for open source projects to sustain in the market and thus this study will highlight the issues in terms of days to resolve the project. It will also highlight that commitment in projects make the project running else open source projects community consist of a number of project but are not being used due to lack of support.

3 Bug Tracking System

When a developer or a user encountered a bug while using the software or wants to request an enhancement, he or she usually open a bug report in the open bug repository. Many open bug repositories (e.g., Bugzilla, Jira, Gnats, and Trac) have been adopted in open source projects. We only explore projects that use Bugzilla as their bug tracking system. This Section introduces background material necessary for this work.

In Bugzilla, bugs are kept in the form of bug reports which consist of predefined fields, text description, and attachments. Bugzilla keeps a record of several information about bug reports in its database. The fields are usually found are:

- Bug ID: Unique identifier for bug reports.
- Description: Textual description of the bug.
- Opened: Date of the report.
- Status: Status of the report. (new, assigned, reopened, needinfo, verified, closed, resolved and unconfirmed).
- Resolution: Action to be performed on the bug (obsolete, invalid, incomplete, notgnome, notabug, wontfix, and fixed.
- Assigned: Name and/or e-mail address of the developer in charge of fixing this bug.
- Priority: Urgency of the error. (immediate, urgent, high, normal and low).
- Severity: How this error affects the use and development of the software. (blocker, critical, major, normal, minor, trivial and enhancement).
- Reporter: Name and e-mail address of the bug reporter.
- Product: Software that contains the bug.
- Version: Version number of the product.
- Component: a Minor component of the product.
- Platform: Operating system or architecture where the error appeared.

Figure 1 shows the bug life cycle adapted form Bugzilla, whenever a new bug is reported, it report is forwarded to the developer for deciding whether the bugs need further details or whether it will be fixed. Once the bug is accepted the Bugs is fixed and the report goes to the reporter. If the solution is accepted then the bug is considered to be resolved else the bug is reopened and goes back to developer.

4 Data Collection

For a selected software program, we extracted the vital data from the bug repository. From that point, we identify the bug reports that were relevant to our software system and perform an extraction of metric for the entire bug reports in

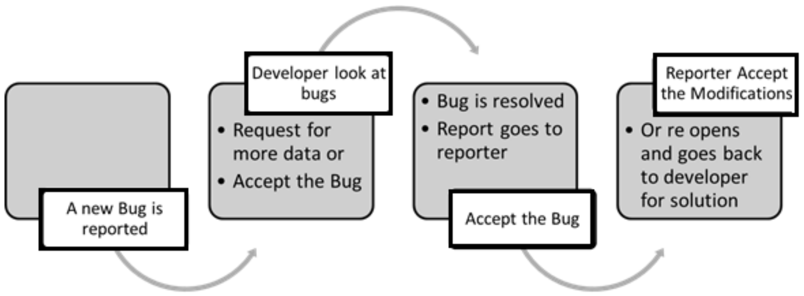
A new Bug is reported
Developer look at bugs
• Request for more data or
• Accept the Bug
• Bug is resolved
• Report goes to reporter
Accept the Bug
Reporter Accept the Modifications
• Or re opens and goes back to developer for solution

Fig. 1. Bugs life cycle as reported by Bugzilla

all of the four projects. This research concentrated on the bug reports metrics to identify the similarities and differences between each project in our dataset. We gathered the data of all bug reports in all projects that were submitted to bug repository Bugzilla in the period from 2004 to 2017 and retrieved almost 13,047 bugs. In this study, we collect all types of bugs, for example, closed, opened, wont fix, etc. First, we check all the available bug repositories for all examined software system in the Bugzilla bug tracker website. For each software system, we extract the essential data from the bug repository (i.e., bugID, description, status, resolution, priority, and reporter). As shown in Figure 2 we have used Bugzilla from three major sources (1) Eclipse (2) KDE and (3) OpenSSH. This research then extracted all the bugs to conduct the study.

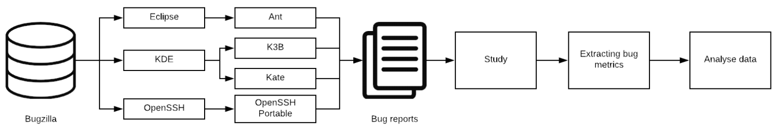

Fig. 2. Process of how this reserach is conducted

5 Research Methodology

This research focused on open source projects from various languages that are Ant [17], K3B[18], Kate [19] and OpenSSH [20]. We have looked over these projects from their first bug reporting year. These projects are evaluated in term of Reported bugs, Number of unsolved bugs, Number of latest commits, Number of Duplicated Bugs and Average time to fix the bugs.

Each project bug has been extracted over the years till date using the crawlers built by us. It is seen that OpenSSH started with the highest number of bugs but has decreased a lot to 213 while for ANT has reduced the number of bugs to half. The most number of bugs were for ANT was 2402 in 2008. Table 1 shows the detail about the system that was selected.

System	**Platform**	**Language**	**#Bug Reports**
Apache	Ant	Java	3,608
KDE	K3B	C++	1,838
KDE	Kate	C++	4,902
OpenSSH	OpenSSH Portable	C	2,699
Total	**13,047**		

Table 1. Project details along with their Bugs.

Table 2 shows the number of bugs collected for each project from the year 2004 to 2017 in terms of a total number of bugs for each year and number of fixed bugs for each year.

Year	**# Bug Reports**				**# Fixed Bugs**			
	Ant	**K3B**	**Kate**	**OpenSSH**	**Ant**	**K3B**	**Kate**	**OpenSSH**
2004	102	7	366	764	94	0	209	379
2005	49	272	171	32	44	115	89	16
2006	21	410	99	263	17	265	40	129
2007	10	105	149	18	3	55	77	4
2008	2402	72	163	216	1044	25	77	137
2009	252	88	310	103	106	56	111	43
2010	175	434	696	157	87	43	299	94
2011	96	59	534	164	30	25	217	80
2012	112	29	570	22	50	4	214	1
2013	97	13	400	48	42	5	151	21
2014	133	224	293	89	72	11	123	35
2015	60	22	404	279	14	0	103	118
2016	45	51	450	331	21	15	102	147
2017	54	52	297	213	13	21	40	2

Table 2. The Bugs Reported and Bugs Solved since 2004 for all the projects.

6 Analysis and Discussion

As shown in Figure 3, the total numbers of resolved bugs are mostly done for Ant except for the year 2008, where actually the numbers of reported bugs were too much. A number of solved bugs reveal that resolution and dedication of committers with the project and overall success of the project.

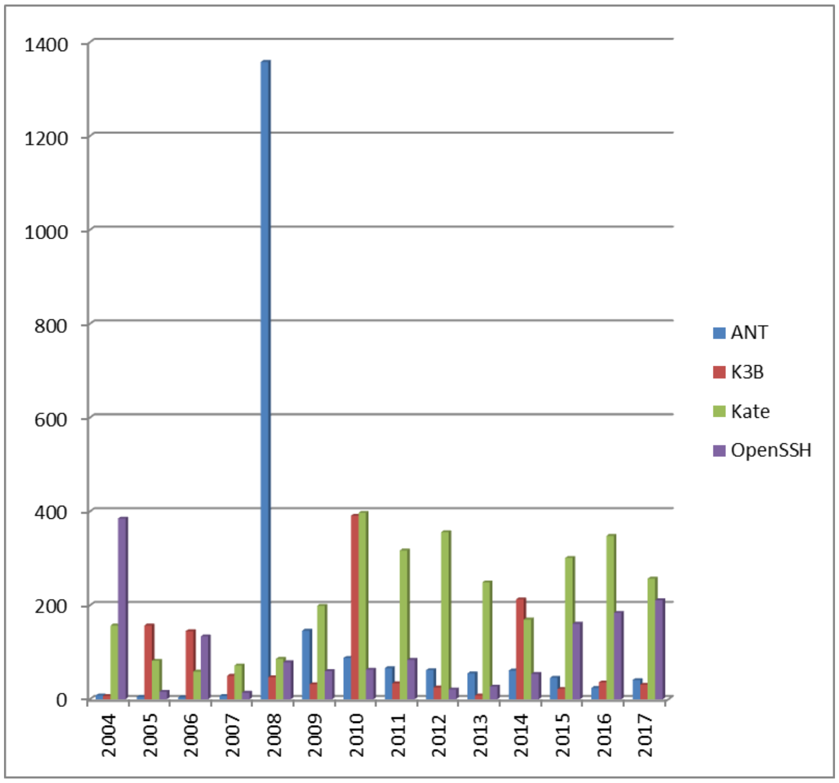

Fig. 3. Number of resolved bugs over the year

Figure 3 shows only the bugs resolved, but most of these bugs were resolved in short time while other were closed as they were wrongly reported. As shown in Table 3, most numbers of bugs are not open except ANT that has 10 opened in 2009. This shows that committers tend to solve most bugs and fix them. [14] also shows that bugs are sometimes misreported and requires a proper mechanism for developers to close this kind of issues. It also shows that some bugs need not be fixed and these fixings requires either some additional effort and will be fixed in upcoming releases. Except Ant and Kate, OpenSSH and K3B have less won't fixed bugs referring to their popularity and their commitment to solve problem. Moreover, the community in OpenSSH is big it also refers to the quick solution of bugs as more dedication is offered by a most number of peoples.

Year	**# Opened Bugs**				**# Won't Fixed Bugs**			
	Ant	**K3B**	**Kate**	**OpenSSH**	**Ant**	**K3B**	**Kate**	**OpenSSH**
2004	0	0	0	0	1	0	17	65
2005	0	0	0	1	1	5	11	1
2006	0	0	0	1	0	11	5	23
2007	0	0	0	0	1	1	11	0
2008	9	0	0	0	220	1	16	32
2009	10	0	0	1	11	0	7	15
2010	1	0	0	1	6	0	20	18
2011	1	0	0	2	6	0	43	27
2012	1	0	0	0	9	0	101	1
2013	1	0	0	2	21	0	25	1
2014	3	2	1	1	5	4	18	0
2015	1	0	0	7	9	0	113	19
2016	1	2	6	2	0	1	45	40
2017	2	1	3	2	4	0	7	1

Table 3. Number of opened as well as won't fixed bugs.

A number of duplicated bugs refers to the point where bugs might be reported differently by two or more people. The opened bugs are referred to committers and they mostly decide whether the bug is duplicate or not. A number of duplicated bugs also refers to how many people faced the same issue and may require more attention than other bugs as shown in Figure 4.

It is seen that ANT has the only spike of high duplicated bugs with Kate have the same issue at number of years but is tend to reduce in current years that shows the awareness among developers regarding the solution of the problems as shown in Equation 1. Equation 1 allows to find a predictor based on trend about number duplicated bugs in upcoming years.

$$y = -25.91 \ln x + 60.98 \tag{1}$$

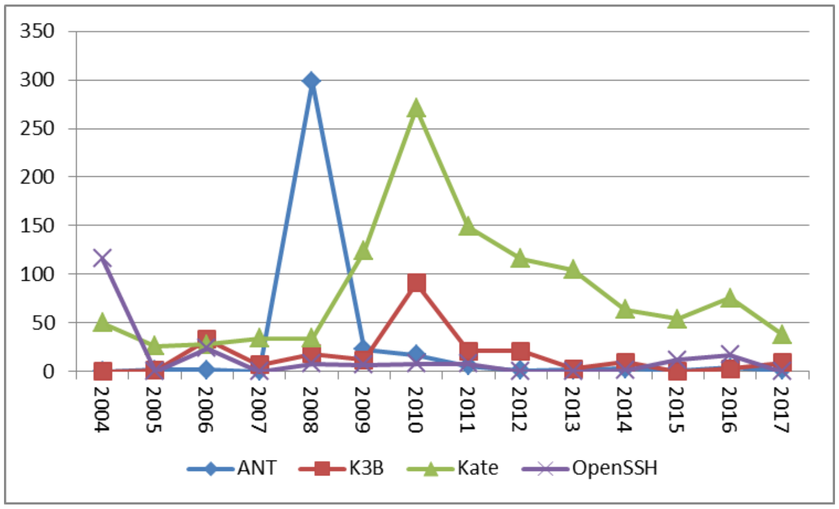

Fig. 4. The number of duplicated bugs over the years for all four projects

A total number of commits represent the dedication of developers and contributors dedication toward the success of the project as shown in Figure 5, shows the number of bug reporters has decreased over the current years that also make sure about the project stability and less number of bugs. It may also depict that project are being fixed consistently and thus require less number of bugs.

The number of days per solution to bug varies from 0 to all four projects to above 5K days that may be due to non-priority of the bug or the bugs require fixing some menu issues and that is solved after the solution of each bug. The average of each bug solving is highest in ANT that is around 1125 days while lowest in K3B that means each bug is solved in a less than year calendar while Kate is also solved in an year as shown in Table 4.

Platform	**#Minimum**	**#Average**	**#Maximum**
Ant	0	1125.061	4721
K3B	0	274.54	2854
Kate	0	370.22	4857
OpenSSH	0	572.94	5470

Table 4. Summary of fixing time of the projects (in #days).

While looking at above results it is proposed that a complete reporting should be made that can help with following:

– Auto classification of bug when reported. This will require two approaches to be followed (1) the proposed system can ask reported to bug to select

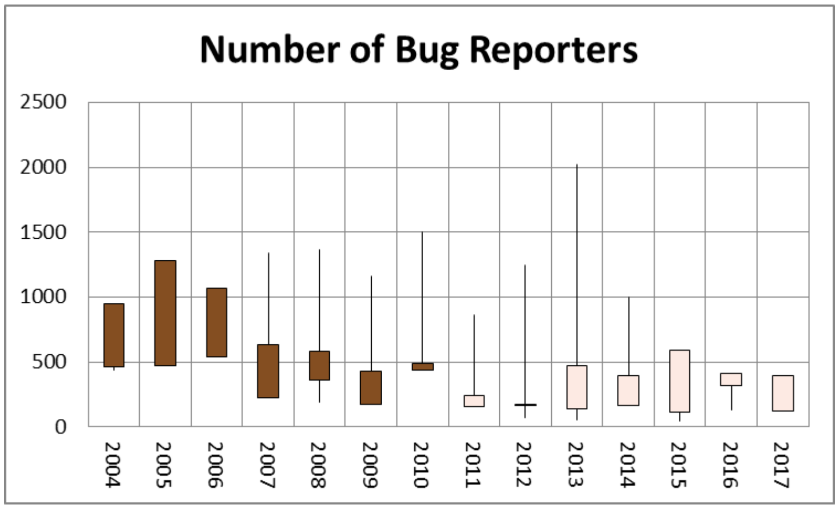

Fig. 5. The number of bug reporters each year combined for a project

the type of bug (2) second and most important approach is text mining and auto classification of bug according the reported issue. This means that whole reported text will be matched against bug classification corpus already built and will be assigned a weight for each category. The category with highest weight will be selected.

- Assigning it to relevant developer automatically that means checking about which developer can fix similar kind of bug, which developer have less load, which developer needs less time to fix the bug thus will be dependent on number of factors and all will be taken into account.
- Assigning the time-line to fix the bug depending on its category like a programmatic ally bug error will have a small fixing time compared to cosmetic changes.
- Propose developers already known solution for the reported kind of bug in order to expedite the process. This will help the developers in doing the quick fix and will allow the developer to accept or reject the solution in order to help other developers whether the suggested solution is better or not.
- Once the bug is solved, a message about fixing the bug will be sent to reported in order to verify and close the ticket. This is will an accountability and easy quality assurance system.

7 Conclusion

This paper looks into the number of systems that are Open Source and popularly used by industry and academia. The authors selected top most used system from different categories all built in different languages. These systems are seen for their number of bug reported by people, the time it takes to solve the bugs and how many bugs are solved over the calendar year. The bugs reporting show that there is a number of people that are using the system and efficiently contributing to testing an open source system that shows the reliability of the system. If the bugs are not solved in time the system fails and lacks interests but continuous development and contribution assures that selected system are still in use since 2004. The authors suggest making a system in order to highlight the bugs to committers and contributors. The system will help in classification of bugs as well as make sure that none of the bugs is active and unresolved in a calendar year. The authors introduced an aging concept for bugs so that the lowest priority bugs make a high priority to solve the problem of overlooking. In future, authors plan to look into the bugs and classify them according to their type thus making the proposed system reality. The system will automatically look for solution using crawlers to find the right kind of solution.

Bibliography

[1] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M Ibrahim, Masao Ohira, Bram Adams, Ahmed E Hassan, and Ken-ichi Matsumoto. Studying re-opened bugs in open source software. *Empirical Software Engineering*, 18(5):1005–1042, 2013.

[2] Pieter Hooimeijer and Westley Weimer. Modeling bug report quality. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 34–43. ACM, 2007.

[3] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schroter, and Cathrin Weiss. What makes a good bug report? *IEEE Transactions on Software Engineering*, 36(5):618–643, 2010.

[4] Anahita Alipour, Abram Hindle, and Eleni Stroulia. A contextual approach towards more accurate duplicate bug report detection. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 183–192. IEEE Press, 2013.

[5] Xihao Xie, Wen Zhang, Ye Yang, and Qing Wang. Dretom: Developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th international conference on predictive models in software engineering*, pages 19–28. ACM, 2012.

[6] Yuan Tian, David Lo, and Chengnian Sun. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pages 215–224. IEEE, 2012.

[7] Ripon K Saha, Matthew Lease, Sarfraz Khurshid, and Dewayne E Perry. Improving bug localization using structured information retrieval. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 345–355. IEEE, 2013.

[8] Pamela Bhattacharya, Liudmila Ulanova, Iulian Neamtiu, and Sai Charan Koduru. An empirical analysis of bug reports and bug fixing in open source android apps. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 133–143. IEEE, 2013.

[9] Sean Banerjee, Jordan Helmick, Zahid Syed, and Bojan Cukic. Eclipse vs. mozilla: A comparison of two large-scale open source problem report repositories. In *High Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on*, pages 263–270. IEEE, 2015.

[10] Minhaz F Zibran, Farjana Z Eishita, and Chanchal K Roy. Useful, but usable? factors affecting the usability of apis. In *Reverse Engineering (WCRE), 2011 18th Working Conference on*, pages 151–155. IEEE, 2011.

[11] Andrew J Ko and Parmit K Chilana. Design, discussion, and dissent in open bug reports. In *Proceedings of the 2011 iConference*, pages 106–113. ACM, 2011.

[12] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiß, Rahul Premraj, and Thomas Zimmermann. Quality of bug reports in eclipse. In

Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange, pages 21–25. ACM, 2007.

[13] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 308–318. ACM, 2008.

[14] Yasir Javed and Mamdouh Alenezi. Defectiveness evolution in open source software systems. *Procedia Computer Science*, 82:107–114, 2016.

[15] Jijie Wang, Mark Keil, Lih-bin Oh, and Yide Shen. Impacts of organizational commitment, interpersonal closeness, and confucian ethics on willingness to report bad news in software projects. *Journal of Systems and Software*, 125:220–233, 2017.

[16] Md Rejaul Karim, Akinori Ihara, Xin Yang, Hajimu Iida, and Kenichi Matsumoto. Understanding key features of high-impact bug reports. In *Empirical Software Engineering in Practice (IWESEP), 2017 8th International Workshop on*, pages 53–58. IEEE, 2017.

[17] Apache ant - welcome. http://ant.apache.org/. (Accessed on 02/11/2018).

[18] Github - kde/k3b: K3b is a full-featured cd/dvd/blu-ray burning and ripping application. https://github.com/KDE/k3b, . (Accessed on 02/11/2018).

[19] Github - kde/kate: An advanced editor component which is used in numerous kde applications requiring a text editing component. https://github.com/KDE/kate, . (Accessed on 02/11/2018).

[20] Openssh. https://www.openssh.com/. (Accessed on 02/11/2018).