G-Issue: Analyzing Lifetime and Evolution of Issue-related Artifacts from Open Source Repositories

Sayed Mohsin Reza
University of Texas at El Paso
El Paso, Texas, USA
sreza3@miners.utep.edu

Saif Uddin Mahmud
University of Texas at El Paso
El Paso, Texas, USA
smahmud4@miners.utep.edu

Omar Badreddin
University of Texas at El Paso
El Paso, Texas, USA
obbadreddin@utep.edu

ABSTRACT

Software developers or contributors report issues related to bugs, errors, and missing documentation during community-based software development. These issues are treated as feedback and are crucial to enhancing software new features, documentation, and quality. If software issues are not being addressed with a correct developer, software quality degrades and is unable to use in the end. Hence, it is essential to analyze the software issue-related artifacts to understand the behavior of the software. This paper investigates the performance of the proposed issue-related artifacts mining tool G-Issue with other state-of-the-art tools. We also investigate issue lifetime and evolution of issues over time among well-known and maintained repositories. The results show that G-Issue is faster in mining issue-related artifacts but takes more memory than general Python API during mining issue mining. The results depict that we can prioritize issues based on issue lifetime and evolution. Such results may provide a new horizon about issues that can help in issue management, developer assignment, and quality management.

G-Issue URL: https://www.smreza.com/projects/modelmine/issues.php

CCS CONCEPTS

• **software and its engineering → Maintaining software**.

KEYWORDS

Mining Software Issues, Issue-related Artifact Mining, Software Engineering, Software Maintenance

1 INTRODUCTION

Software development becomes distributed nowadays, and developers from anywhere can contribute towards the software development [1]. Towards this development, some software manages technical artifacts like commits, issues, and milestones which enables a social community that attracts many developers to work on and deliver projects within timeline [2, 30]. BitBucket [11], GitHub[5], GitLab [24] is the leader in distributed version control and source code management (SCM), which combines the ability to develop, secure, and operate software in a single application.

Source code management software is growing in features that allow faster development through bug identification, error reporting, or other issues. One of the features is an issue tracking system, often used to get user feedback related to proposed features, bugs, errors, and problems. Also, the service allows the developers to assign an issue to a developer [20] and automatic labeling issues to prioritize it better [18]. In summary, this tracking system enhances the code quality and increases the software lifetime.

Software maintenance is a costly and largely unpredictable human-intensive activity in the software development life cycle. High maintenance efforts and expertise often eclipse the cost and sometimes become the reason for unsustainable software [16]. Moreover, if issues are not well managed during this maintenance, the software becomes smelly and may introduce bugs, and obsolete in the long run [25]. To solve such issues, developers worldwide may provide feedback on an issue and can contribute to fixing that. Therefore, source code management with issue tracking can provide collaborative pathways to manage software, reduce software failures and improve software quality.

Very few research efforts have been conducted on mining [8, 17, 29], analyzing [6] and visualizing [10] issues in open source communities. These efforts include issue title prediction [34], automatic issue labeling [31, 32] and sentiment analysis of issues [7, 15]. However, there is a missing effort on mining issues faster, analyzing issue timelines, and evolving issues over time.

In this paper, we investigate the performance of issue mining of an in-house developed tool, called G-Issue, and compare performance with other state-of-the-art tools [13, 28] in terms of execution time and memory usage. Moreover, we investigate the average issue lifetime in popular open source repositories and analyze the evolution of issues over time among repositories to see the behavior of each repository.

This study is structured as follows: Section 2 discusses the related research works on issue tracking and its management; Section 3 discusses the study design with research questions. Section 4 shows results against each research questions and discuss elaborately and finally we conclude in Section 5.

2 RELATED WORK

Software development through source code management and its associated artifacts are available on an open-source platform. Several studies have been conducted research on such artifacts from different perspective such as sentiment analysis [17], label prediction[19], issue management [4] & mining [34].

2.1 Issue-related Artifact Mining

Software artifact mining has improved software quality, bug identification, and network analysis. Several studies have uncovered interesting and actionable artifacts from software data. Several mining tools have emerged to enable such research, and discovery [22, 26, 28]. For example, PyDriller, a python framework for mining software repositories, can extract recent information from open source repositories such as commits, developer information, modifications, differences, and source codes [28]. However, the tool has no feature to extract issues. MetricMiner is another application

suitable for mining software repositories for metrics calculation, data extraction, and statistical inference [27]. These tools focus on extracting data primarily from either code or commit history, with limited support for mining issue-related artifacts.

2.2 Issue-related Artifact Analysis

Issue-related artifact analysis has gained popularity last few years. Several studies have researched on issue lifetime [19, 21], how long it will take to close an issue, and empirical studies on the life expectancy of issues based on labels. Kikas et al. conducted research on 4000 repositories to find temporal dynamics of issues in GitHub [19]. The study found that the projects with a shorter observation time tend to have higher volumes of open issues. In addition, Kikas proposed a prediction model trained from static, dynamic, and contextual features to predict the lifetime of an issue. The results showed that the average issue lifetime for community-created issues is 39 days, but the team-created issues are 5.9 days.

2.3 Issue-related Artifact Visualization

Visualization techniques have been popular in textual & network data. As issue-related artifact has textual(issue title, body) and network (issue assignee, label) data, several research have been conducted on issue-related artifact visualization [4, 20]. Liao et al. applied the visualization technique on issue-related behavior by analyzing seven projects with 98,074 issues in total [20] and proposed an SRF to measure the importance of user behaviors. The results found that issue-related user behaviors are critical, and not all issues that are assigned labels could be closed rapidly. Another empirical research by Bissyand et al. investigated the adoption of an issue tracker based on the projects, developers, and type of issues [4]. The results found that small-sized team projects are less likely to have issues and visualized the top 10 labels in GitHub, where bug and feature requests are at the top with 18.36% and 10.29%, respectively.

2.4 Issue-related Other Research

There are other research on issues conducted in aspects of sentiment [7, 17, 33], bug and issue-tracker analysis [25]. Jurado et al. conducted a study on 10,829 issues from 9 well-known & open source repositories to do sentiment analysis [17]. The study proposed a new technique to identify the underlying sentiments in the text found in issues and their comments. Results showed that issues' titles and text leave underlying sentiments, which can be used to analyze the development process. Another research on issues-related artifacts is automatic labeling of GitHub Issues [3, 12, 18]. Kallis et al. introduced a tool called Ticket Tagger, which is developed using Node.js and de-facto server-side JavaScript and uses machine learning approaches on issue artifacts to label an issue automatically [18].

In summary, there is insufficient research on issue-related artifact mining, analysis, and visualization through a web application. Hence, this scope study is unique in terms of faster mining without any hassle of processing, analyzing, and visualizing the artifacts.

3 STUDY DESIGN

The study aims to analyze issue-related artifacts from open source repositories with the purpose of mining, pre-processing, and visualizing the issues which can be effectively used in practice. The perspective is of both researchers and practitioners who are interested in analyzing the issues in terms of issue expectancy and evolution of issues. Specifically, we aim to address the following research question:

3.1 Research Questions

This section discusses the research questions we used and how we plan to answer these research questions. We are motivated to find the answer to the following research questions:

RQ1. What is the performance of the G-Issue tool compared to the state-of-the-art tools in mining issue-related artifacts?

The RQ focuses on the performance evaluation of G-Issue and is motivated by the fact that issue-related artifacts are crucial in repositories compared to code itself and tend to be significantly larger in terms of text size and issue comments. This often translates to complexity in identifying and extracting issue-related artifacts. For reference, we compare ModelMine with state of the art tools Python API [17], GHTorrent [13, 14], PyDriller [28], G-Repo [26] for mining issues from GitHub. To answer this research question, we choose three individual tasks that are common for the majority of mining research with available support in mining tools. The tasks are as follows:

(1) **Task 1 (Size related):** Retrieve the list of 1000 issues that include at least one open state issue, and the total number of issues is more than 1000.
(2) **Task 2 (Time-related):** Retrieve the list of 1000 issues that include at least one open state issue and created before January 2019.
(3) **Task 3 (State related):** Retrieve the list of 1000 issues now in a closed state.

These tasks are implemented using the following frameworks/tools: (1) G-Issue, (2) Python API, (3) GHTorrent, (4) PyDriller, and (5) G-Repo. To compare the tools, we use two performance metrics: (1) Execution Time and (2) Max Memory (MM). Such performance metrics are used in evaluating different software artifacts mining tools [9, 22, 28]. The evaluation checks how fast and how much memory the tool takes to mine issue-related artifacts.

RQ2. What is the average issue lifetime among different repositories?

This RQ describes the analysis of the time it takes to solve an issue for each repository in our dataset on average. After collecting issues using G-Issue, we will find closed state issues, its created time, and when it is closed. We reveal the average issue lifetime among different repositories based on those data.

RQ3. What is the evolution of issues over time among repositories?

This RQ shows the evolution of open and closed state issues among repositories. To prepare the results, we need to extract yearly issues and their state from all the issues. We also plan to show Kernel Density Estimation (KDE) as a part of the probability density function on our ongoing issue creating time variables.

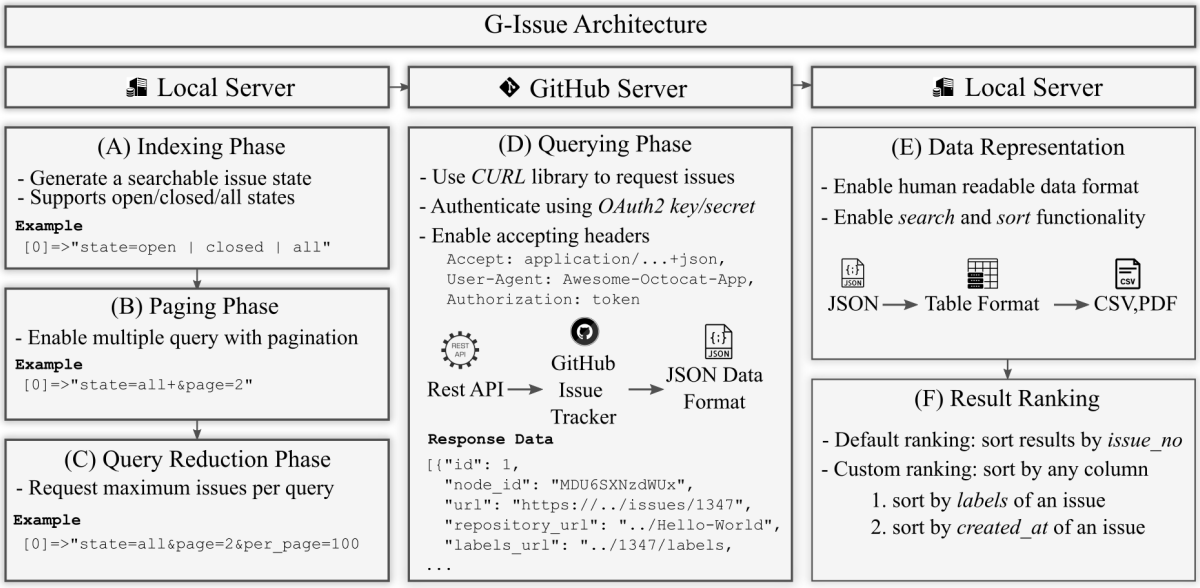

Figure 1: Architecture of G-Issue Tool

With these research questions, we aim to provide a more profound knowledge of the capabilities of G-Issue in mining and analysis of issue-related artifacts. The following subsections report the architecture of G-Issue and the steps that we conducted to collect the dataset.

3.2 G-Issue Architecture

In this section, we discuss the architecture of the issue mining tool G-Issue that we built in-house lab setup and hosted on the online platform. The tool adopts several approaches (indexing, paging, query reduction, querying, data representation, and results ranking) to mine issue-related artifacts of repositories from open source repositories. The overall architecture of the G-Issue tool is visualized in Figure 1.

In G-Issue, we provide a user interface with the mining capability to request GitHub for issue-related artifacts and process that data. This service is under the parent tool called ModelMine [22]. This tool provides a simple, extensible user interface to mine issue-related artifacts of repositories. It has a different way of searching to ensure the possibility of different mining types of datasets for MSR research.

Software issues have multiple types of artifacts, including state, milestone, assignee, and G-Issue, allowing researchers to investigate specific state-based issue searches. This feature allows researchers to analyze the different states of the issues in repositories and the behavior of software code issues of different projects. The user interface of the G-Issue tool is visualized in Figure 2.

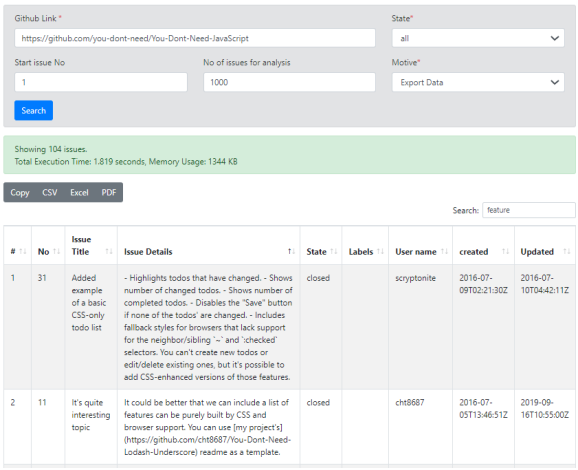

Figure 2: Search & result screenshot of G-Issue Tool

3.3 Data Collection

One of the challenges in software research is identifying code repositories that have been actively maintained for an extended period. We identify some characteristics that may give us actively maintained repositories to search such code repositories. The characteristics are as follows: a repository with a minimum of 5000 commits, at least 100 active contributors, a minimum of 3000 stars, and 500 forks. We use the ModelMine tool [22] which is capable of retrieving repositories with the mentioned criteria. A high number of stars and forks imply the popularity of the repositories, and a high number of commits imply maintenance throughout the software development life cycle. We choose the top ten repositories from the results provided by the ModelMine tool. Overall, the selected repositories have code changes in commits that will help us to extract the different source code metrics to reduce threats to the generalizability of this study. In this study, we have mined repositories and created a dataset

Table 1: Selected repositories with metadata information

Serial	Repository name	Commits	Contr.	Stars	Forks	Time Selection	No. Open Issues	No. Closed Issues	Total Issues
1	Spring framework	22,208	531	41,400	28,800	2004-05 to 2022-08	1,391	24,593	25,985
2	Junit-5	6,621	161	4,400	991	2015-01 to 2022-08	135	2,828	2,963
3	Apache kafka	8,590	762	18,000	9,600	2012-08 to 2022-08	1,002	11,477	12,479
4	Apache lucene-solr	34,789	232	4,100	2,700	2016-01 to 2022-08	255	2,411	2,666
5	Dropwizard	5,702	361	7,900	3,300	2011-03 to 2022-08	26	5,518	5,544
6	Checkstyle	9,922	254	5,800	7,700	2013-09 to 2022-08	697	11,287	11,984
7	Hadoop	24,612	339	11,300	7,000	2014-09 to 2022-08	681	3,681	4,362
8	Selenium	26,532	558	19,800	6,200	2013-01 to 2022-08	117	10,597	10,715
9	Skywalking	6,242	315	16,100	4,700	2015-11 to 2022-08	62	8,525	8,587
10	Signal android	7,015	223	19,800	4,700	2011-12 to 2022-08	242	10,049	10,291

composed of ten open source repositories. Then we use the G-Issue tool to mine issue-related artifacts. The whole dataset is now published in GitHub https://github.com/sayedmohsinreza/CSIQ and available online [23]. The detailed summary of the ten open source repositories and issues in each repository are reported in Table 1.

3.4 Terminology

The software issues have some particular terminology we need to discuss to understand the results. Occasionally, issue-related artifacts include reporting bugs, requesting new features, refactoring code, and enhancement ideas. Also, the artifacts are typically created by anyone with title details and consist of the person's information, created time, and labels associated with the issues. If the issue is closed or modified, that record is also documented in the specific issue.

Here are the details of some terminologies used in this study.

- **Issue lifetime** - Time from the first opening of the issue to the first closing of the issue.
- **Opened issue** - Newly created issue. Each issue is opened only once during its lifetime.
- **Closed issue** - issue that is marked closed in the issue tracking system. In practice, an issue might be reopened and closed again, but here we use only the last closing event.

4 RESULTS & DISCUSSION

In this section, we report the results and analysis of the research questions mentioned in Section 3.1.

4.1 Performance Evaluation

This section discusses the results of the performance of G-Issues compared to other state-of-art-tools. The performance evaluation results among the tools are visualized in Table 2. Such a result provides an idea of which tool performs better during mining issue-related artifacts and how much fast and memory the tool takes to mine selected repositories. All these results are produced with the setup to mine 1000 issues from repositories.

Table 2 shows that in each task, G-Issue mines a list of 1000 issues with the lowest execution time while GHTorrent mines with the highest execution time. Python API has the lowest memory utilization during mining, and GHTorrent has the highest utilization. Among state-of-the-art tools, PyDriller and G-Repo have focused on mining software repositories and have no feature to mine issue-related artifacts.

Table 2: Performance evaluation results

Tasks	Metrics	G-Issue	Python API	GHTorrent	PyDriller	G-Repo
Task 1*	ET**	**12.1s**	18.2s	46.2s	Not	Not
(Size)	MM***	18223KB	**10211KB**	67033KB	supported	supported
Task 2	ET	**30.22s**	41.7s	88.3s	Not	Not
(Time)	MM	20340KB	**16547KB**	74031KB	supported	supported
Task 3	ET	**11.8s**	15.5s	102.3	Not	Not
(Issue-related)	MM	19967KB	**14566KB**	63654KB	supported	supported

* Task details are listed in Section 3.1
** ET - Execution Time
*** MM - Max Memory

4.2 Analysis of Issue Lifetime

In this section, the results of issue lifetime among repositories are discussed and portrayed in Table 3. The table shows the average days it takes to solve an issue among repositories. Here "Average days to solve" means how many days it takes to close the issue by developers since the issue creation date.

Table 3: Statistics on days it takes to solve an issue

Project Name	Mean (days)	Minimum (days)	Maximum (days)
1. spring-framework	1220	0	5491
8. selenium	551	0	2574
10. signal-android	215	0	3010
2. junit-5	162	0	2144
3. apache-kafka	104	0	2467
5. dropwizard	101	0	3221
7. hadoop	98	0	2158
4. apache-lucene-solr	91	0	2030
6. checkstyle	57	0	2496
9. skywalking	37	0	1840

From Table 3 results, we can see that *Spring Framework* project has the highest on average of 1220 days to solve an issue where *skywalking* developers use only 37 days. *spring-framework* commit count is less than *haddop* project but average issue lifetime in *haddop* is twelve time less than *spring-framework*. For each repository, the minimum issue lifetime day is zero, which implies that within the issue created date, developers solve the issue and close that.

However, Figure 3 visualizes the boxplot of issue lifetime among repositories. From the figure, it is noticeable that *spring-framework*

and *selenium* has the highest mean of days to solve an issue. Among all repositories, one issue from *spring-framework* has taken more than 5000 days / 13 years to solve. Here, we need to keep in mind that some issues are closed and reopened later on to receive more feedback on that issue.

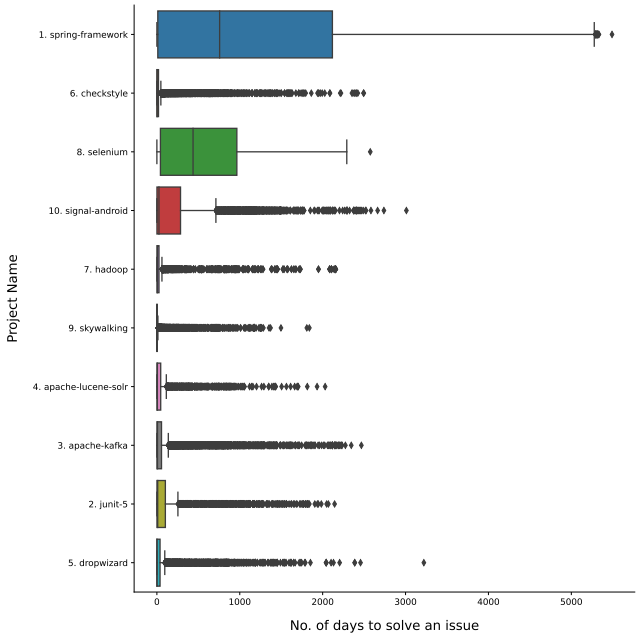

Figure 3: Box plot of days it takes to solve issues among repositories

4.3 Evolution of Issues

In this section, we discuss the evolution of issues among repositories. The results of the evolution of issues are visualized in Figure 4 showing a histogram of issue count per year in terms of open or closed state among repositories.

In every case, the graph implies that new issues are increasing in number during the software evolution. This number increases and becomes higher when the close-state issue rate declines. *spring-framework*, *junit-5*, *checkstyle* and *signal-android* shows a recent decline in the rate of closed-state issues and an upward trend of new issues. The KDE density value represents an increasing number of issues reported by developers or contributors.

Also, we have seen a pattern of the zigzag move of issues over the years among the repositories. It implies that when new issues are introduced within that year, it tries to be solved and closed the issue. Hence, the continuous maintenance through issue-related artifact analysis prepares software for the subsequent releases with improved software quality and minimized bugs in reporting.

5 CONCLUSION

Software maintenance is crucial during software development. If the maintenance efforts are not made correctly, the software quality degrades over time and is hard to fix at one point. To do software

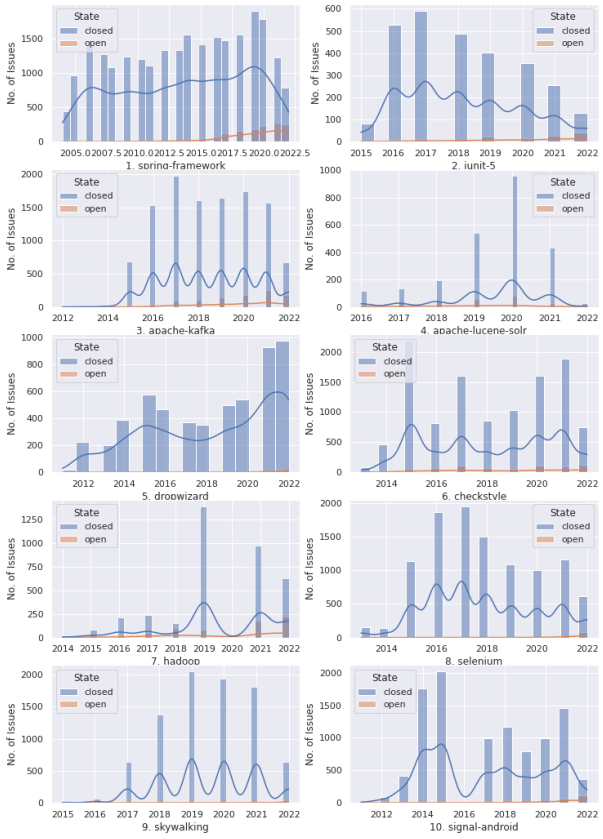

Figure 4: evolution of issue-related artifacts over time among repositories

maintenance, developers need feedback in the form of issues. Most source code management software nowadays provides issues to report bugs and share ideas for new features.

In this study, we investigated the process of mining, analyzing, and visualizing issue-related artifacts through a developed tool called G-Issue. The study primarily compares the performance of the G-Issue tool with state-of-the-art tools. Moreover, we investigate the lifetime and evolution of issues in well-known open source projects.

The results show that the G-Issue tool performs a minimum of 33% faster than other state-of-the-art tools. However, in memory management, G-Issue is higher than the Python API but lower than other tools. Besides, the results show that highly popular & forked repositories have more issues; on average, it takes more days to solve an issue. In terms of evolution, if the rate of the closed issue is declining, there is a high chance of introducing new issues. Such

results may provide new knowledge about issues-related artifacts and help team leaders with issue assignments for better software development.

In future research, we plan to analyze the issue text and apply natural language processing to identify issue labels, improving the issue label tracking system.

REFERENCES

[1] Andrew Begel, Jan Bosch, and Margaret-Anne Storey. 2013. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *IEEE software* 30, 1 (2013), 52–66.

[2] Dane Bertram, Amy Voida, Saul Greenberg, and Robert Walker. 2010. 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*. 291–300.

[3] Shikhar Bharadwaj and Tushar Kadam. 2022. Github issue classification using bert-style models. In *2022 IEEE/ACM 1st International Workshop on Natural Language-Based Software Engineering (NLBSE)*. IEEE, 40–43.

[4] Tegawendé F Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillere, Jacques Klein, and Yves Le Traon. 2013. Got issues? who cares about it? a large scale investigation of issue trackers from github. In *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*. IEEE, 188–197.

[5] John D Blischak, Emily R Davenport, and Greg Wilson. 2016. A quick introduction to version control with Git and GitHub. *PLoS computational biology* 12, 1 (2016), e1004668.

[6] Mário André de F. Farias, Renato Novais, Methanias Colaço Júnior, Luís Paulo da Silva Carvalho, Manoel Mendonça, and Rodrigo Oliveira Spínola. 2016. A systematic mapping study on mining software repositories. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 1472–1479.

[7] Jin Ding, Hailong Sun, Xu Wang, and Xudong Liu. 2018. Entity-level sentiment analysis of issue comments. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*. 7–13.

[8] Santiago Dueñas, Valerio Cosentino, Gregorio Robles, and Jesus M Gonzalez-Barahona. 2018. Perceval: software project data at your will. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 1–4.

[9] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N Nguyen. 2013. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 422–431.

[10] Aron Fiechter, Roberto Minelli, Csaba Nagy, and Michele Lanza. 2021. Visualizing github issues. In *2021 Working Conference on Software Visualization (VISSOFT)*. IEEE, 155–159.

[11] John Fisher, D Koning, and AP Ludwigsen. 2013. *Utilizing Atlassian JIRA for large-scale software development management*. Technical Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

[12] Mehdi Golzadeh, Alexandre Decan, Damien Legay, and Tom Mens. 2021. A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *Journal of Systems and Software* 175 (2021), 110911.

[13] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: GitHub's data from a firehose. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 12–21.

[14] Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. 2014. Lean GHTorrent: GitHub data on demand. In *Proceedings of the 11th working conference on mining software repositories*. 384–387.

[15] Emitza Guzman, David Azócar, and Yang Li. 2014. Sentiment analysis of commit comments in GitHub: an empirical study. In *Proceedings of the 11th working conference on mining software repositories*. 352–355.

[16] Les Hatton, Diomidis Spinellis, and Michiel van Genuchten. 2017. The long-term growth rate of evolving software: Empirical results and implications. *Journal of Software: Evolution and Process* 29, 5 (2017), e1847.

[17] Francisco Jurado and Pilar Rodriguez. 2015. Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues. *Journal of Systems and Software* 104 (2015), 82–89.

[18] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket tagger: Machine learning driven issue classification. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 406–409.

[19] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. 2016. Using dynamic and contextual features to predict issue lifetime in github projects. In *2016 ieee/acm 13th working conference on mining software repositories (msr)*. IEEE, 291–302.

[20] Zhifang Liao, Dayu He, Zhijie Chen, Xiaoping Fan, Yan Zhang, and Shengzong Liu. 2018. Exploring the characteristics of issue-related behaviors in github using visualization techniques. *IEEE Access* 6 (2018), 24003–24015.

[21] Mitch Rees-Jones, Matthew Martin, and Tim Menzies. 2017. Better predictors for issue lifetime. *arXiv preprint arXiv:1702.07735* (2017).

[22] Sayed Mohsin Reza, Omar Badreddin, and Khandoker Rahad. 2020. Modelmine: a tool to facilitate mining models from open source repositories. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–5.

[23] Sayed Mohsin Reza, Saif Uddin Mahmud, Khandoker Rahad, and Omar Badreddin. 2022. *CSIQ: A Synthesized Dataset of Code Smells, Issues and Quality related Artifacts from Open Source Repositories*. https://doi.org/10.17632/77p6rzb73n

[24] Julio César Cortés Ríos, Kamilla Kopec-Harding, Sukru Eraslan, Christopher Page, Robert Haines, Caroline Jay, and Suzanne M Embury. 2019. A methodology for using GitLab for software engineering learning analytics. In *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 3–6.

[25] Gema Rodríguez-Pérez, Jesús M Gonzalez-Barahona, Gregorio Robles, Dorealda Dalipaj, and Nelson Sekitoleko. 2016. Bugtracking: A tool to assist in the identification of bug reports. In *IFIP International Conference on Open Source Systems*. Springer, 192–198.

[26] Simone Romano, Maria Caulo, Matteo Buompastore, Leonardo Guerra, Anas Mounsif, Michele Telesca, Maria Teresa Baldassarre, and Giuseppe Scanniello. 2021. G-Repo: a Tool to Support MSR Studies on GitHub. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 551–555.

[27] Francisco Zigmund Sokol, Mauricio Finavaro Aniche, and Marco Aurélio Gerosa. 2013. MetricMiner: Supporting researchers in mining software repositories. In *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 142–146.

[28] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. Pydriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 908–911.

[29] Xiaobing Sun, Bixin Li, Hareton Leung, Bin Li, and Yun Li. 2015. MSR4SM: Using topic models to effectively mining software repositories for software maintenance tasks. *Information and Software Technology* 66 (2015), 1–12.

[30] José Apolinário Teixeira and Helena Karsten. 2019. Managing to release early, often and on time in the OpenStack software ecosystem. *Journal of Internet Services and Applications* 10, 1 (2019), 1–22.

[31] Jun Wang, Xiaofang Zhang, and Lin Chen. 2021. How well do pre-trained contextual language representations recommend labels for GitHub issues? *Knowledge-Based Systems* 232 (2021), 107476.

[32] Jun Wang, Xiaofang Zhang, Lin Chen, and Xiaoyuan Xie. 2022. Personalizing label prediction for GitHub issues. *Information and Software Technology* 145 (2022), 106845.

[33] Bo Yang, Xinjie Wei, and Chao Liu. 2017. Sentiments analysis in GitHub repositories: An empirical study. In *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*. IEEE, 84–89.

[34] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, DongGyun Han, David Lo, and Lingxiao Jiang. 2022. itiger: An automatic issue title generation tool. *arXiv preprint arXiv:2206.10811* (2022).