INTI JOURNAL | Vol.2024:27
eISSN:2600-7320

Effective Bug Triage for Software Development and Maintenance

S Sai Kishore [1*], Chitra K [1], Nur Fatin Liyana Binti Mohd Rosely[2]

[1] DSATM Kanakapura Road, UDAYAPURA, Bangalore Karnataka, India
[2] Faculty of Data Science and Information Technology, INTI International University, 71800 Nilai, Negeri Sembilan, Malaysia

[*]**Email**: saikishoreraju238@gmail.com, fatinliyana.rosely@newinti.edu.my

Abstract

Software businesses allocate about 45% of their budget to resolving issues. Bug triage is an essential step in the bug-fixing process that aims to effectively provide a developer with information about a new bug. This research focusses on the issue of data minimization in bug triage, which involves reducing and enhancing the quality of bug data. Utilize instance and feature selection techniques to simultaneously decrease the size of both the word and data dimensions related to bugs. The objective is to construct a prediction model for a novel bug data set by utilizing qualities from previous bug data sets. Additionally, we aim to assess the comparative significance of employing feature and instance selection in the sequence in which they are implemented. Empirically evaluate the effectiveness of data reduction by analyzing a total of 600,000 bug reports from two significant open-source projects, Mozilla and Eclipse. The findings indicate that our data reduction technique has the potential to effectively decrease the bulk of data while enhancing bug triage accuracy. Our study effort presents a methodology for utilizing data processing techniques to provide superior, sparsely populated bug data for the sake of software development and maintenance.

Keywords

Mining software repositories, data preprocessing, reduction of bug data, feature selection, triage of bugs

Introduction

Software companies allocate around 45 percent of their total expenses towards rectifying bugs and other issues [S. Amershi et al., 2019; Z. Jia et al., 2022]. Once an issue is identified, the method should immediately go on to bug triage, where efforts will be made to assign a developer to the bug in a suitable manner. The automated bug triage approach effectively utilized text categorization technologies to reduce the workload and costs associated with human labour [X. Xia, et al., 2017; H. Wu et al., 2017]. In this study, we address the challenge of data reduction for bug triage, which refers to the problem of improving data quality while simultaneously decreasing the total number of defects. Currently, we are focusing on the precise details of how to accomplish each of these objectives. We combine the selection of occurrences and the selection of features to make the data more concise in terms of both the word and bug dimensions simultaneously. Our

Submission: 4 May 2024; **Acceptance:** 6 August 2024

Copyright: © 2024. All the authors listed in this paper. The distribution, reproduction, and any other usage of the content of this paper is permitted, with credit given to all the author(s) and copyright owner(s) in accordance to common academic practice. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license, as stated in the website: https://creativecommons.org/licenses/by/4.0/

team has created a model to predict a new set of bug data. Subsequently, we analyze previous bug data sets to extract the attributes that are associated with those bugs, with the intention of including them into the model. Hence, we can ascertain the optimal order in which to offer the instance selection and feature selection choices. To undertake an empirical investigation on the effectiveness of data reduction, we utilized almost 600,000 bug reports obtained from two prominent open-source projects, Mozilla and Eclipse [Z. Hassan, et al., 2021]. Based on the findings, our approach to decreasing data volume could both enhance bug triage precision and decrease the overall data quantity. This work focusses on utilizing pre-existing data processing techniques and showcases a strategy that can enhance the precision of bug data. By concurrently reducing the amount of data linked to software development and upkeep. The research findings provide a technique for achieving this goal.

Related Works

Previous research on modelling bug data, triaging bugs, and enhancing defect prediction's ability to predict bugs was examined.

A. Modeling Bug Data

Guo, S. constructs a bug report network to examine the interdependencies among problem reports for the purpose of analyzing the associations within bug data [Guo, S. et al., 2020]. Hong et. al. not only examined the connections between bug reports, but also created a developer social network to study developer collaboration using the bug data from the Mozilla project. Gaining insights into the developer community and the progression of projects is facilitated by utilizing this developer social network. Xuan et al. identified the process of developer prioritization in open-source issue repositories by establishing a correlation between bug priorities and developers. Developer prioritization can facilitate the identification of developers for software maintenance activities.

In their study, Soltani et al. (2020) devised questionnaires to assess the quality of bug data in three open source projects. These questions were administered to both developers and consumers. The researchers establish criteria for evaluating a bug report's quality by analysis of questionnaires. Additionally, they develop a classifier to assess whether a bug report requires improvement. Duplicate bug reports diminish the quality of bug data by delaying the cost of defect resolution. He proposed a strategy for identifying duplicate bugs by improving a retrieval function that considers many aspects. Additionally, he developed a natural language processing method that matches the execution information.

B. Bug Triage

The objective of bug triage is to determine the most suitable developer to handle a new bug, or to make a decision regarding who should handle it. In order to reduce the expenses associated with manual bug triage, "ubrani" and "Murph" initially implemented an automated system for triaging problems. Goyal and Sardana (2021) employ text categorisation algorithms to predict forthcoming advancements. Goyal, A., & Sardana, N. examine several bug triage procedures, including data preprocessing and traditional classifiers. Goyal and Sardana have expanded on the previous work by creating recommenders that are focused on development and aimed at reducing the effort required for bug triage.

Based on the findings of other researchers, human bug triage has reallocated over 37% of problem complaints. They recommend employing a scatter plot method to minimize reassignment in bug triage. To prevent the submission of low-quality bug reports during the bug triage process, Xuan et al. developed a semi-supervised classifier by merging both unlabeled and labelled problem reports. Park et al. convert bug triage into an optimization problem and suggest a collaborative filtering strategy to decrease the time required for bug fixes.

The bug triage challenge in data mining is connected to the issues around expert identification and ticket routing. Bug triage is a specialized process that specifically deals with assigning developers to problem reports, as opposed to the more general topics of expert discovery or ticket routing. In addition, bug triage involves transforming bug reports into documents and employs content-based classification, which differs from the sequence-based classification used in ticket routing.

Methodology

A. Agile Methodology

The agile technique is a project management approach that divides projects into smaller, more manageable segments called "sprints." Uninterrupted communication and collaboration among all stakeholders are necessary for ongoing progress at all levels. Upon commencing their work, teams promptly initiate a process that encompasses the stages of planning, executing, and assessing the work they have accomplished. For success to be ensured, it is imperative to have continuous communication among team members and individuals who have a personal interest in the project.

Agile project management is a type of project administration that focusses on collaboration and making progress in small steps. The core principle of the agile project management technique is the belief that it is possible to make gradual enhancements to a project during its duration by making necessary modifications. This concept is the basis for the agile software development methodology. Agile project management is quickly gaining popularity as a preferred method for project management. The effectiveness of agile project management may be primarily attributable to its adaptability, extensive client engagement, and willingness to embrace change, which are defining characteristics of agile project management.

The Scrum technique distinguishes itself from other techniques by several key aspects. These include its iterative and incremental development process, its emphasis on self-organising teams, its focus on delivering value to the customer, and its ability to adapt to changing requirements.A heuristic approach prioritizes acquiring knowledge through practical experience and adapting behavior in response to changing circumstances. Heuristic frameworks are constructed using knowledge gained from practical experience and systematic testing. Scrum is an example of such a framework. It recognizes the team's limited knowledge at the start of a project and anticipates the discovery of new information as they progress towards completing the assignment. The primary objective of Scrum is to enable teams to easily adapt their procedures to meet constantly changing needs and an expanding set of expectations. Incorporating reprioritization and maintaining short release cycles will enable any team to continually acquire new knowledge and improve.

Three essential components of a scrum team will always maintain their importance and require our unwavering dedication and exertion. The Product Backlog, a crucial inventory of work requiring completion, is overseen by either the product owner or the product manager, depending on the individual responsible for the product. The sprint backlog is populated with a dynamic assortment of features, requirements, enhancements, and bug fixes. The team's "To Do" list is essentially a collection of items that meet that specific need.

The Sprint Backlog is a curated inventory of features, user stories, or bug patches that the development team has chosen to focus on during the ongoing sprint. The list can be located in the sprint planning document of the product. The team has assessed the following components, determined their relative importance, and ranked them accordingly. During the pre-sprint meeting called "sprint planning," the team determines the specific items from the product backlog that they will focus on for the upcoming sprint.

An increment is the outcome of a completed sprint that is fully developed and can be immediately utilized. A sprint objective is a synonym for an increment. The term "increment" is rarely frequently used, as it is more typical to refer to "Done" in relation to the team's definition, a milestone, the sprint objective, a complete version, or a released epic. Consequently, the term "increment" is less probable to be utilized. Both the significance of the term "Done" and the arrangement of your sprint's objectives are quite significant.

B. Data Flow Diagrams

A data flow diagram is a visual representation of the system used to manage or store data. This specific numerical format is often utilized in computer systems. To obtain the most accurate and valuable interpretation of the data, it is essential to consider the data methodology, data sources, and the required explanations. A model of an object can be created using a software application called DFD, which is specifically designed for modelling purposes. The DFD employs a diverse range of diagram forms, including event diagrams, activity diagrams, transition diagrams, and class diagrams, to depict the connections between data in various ways. The figure 1 below depict several instances of information flows by 3 different level.

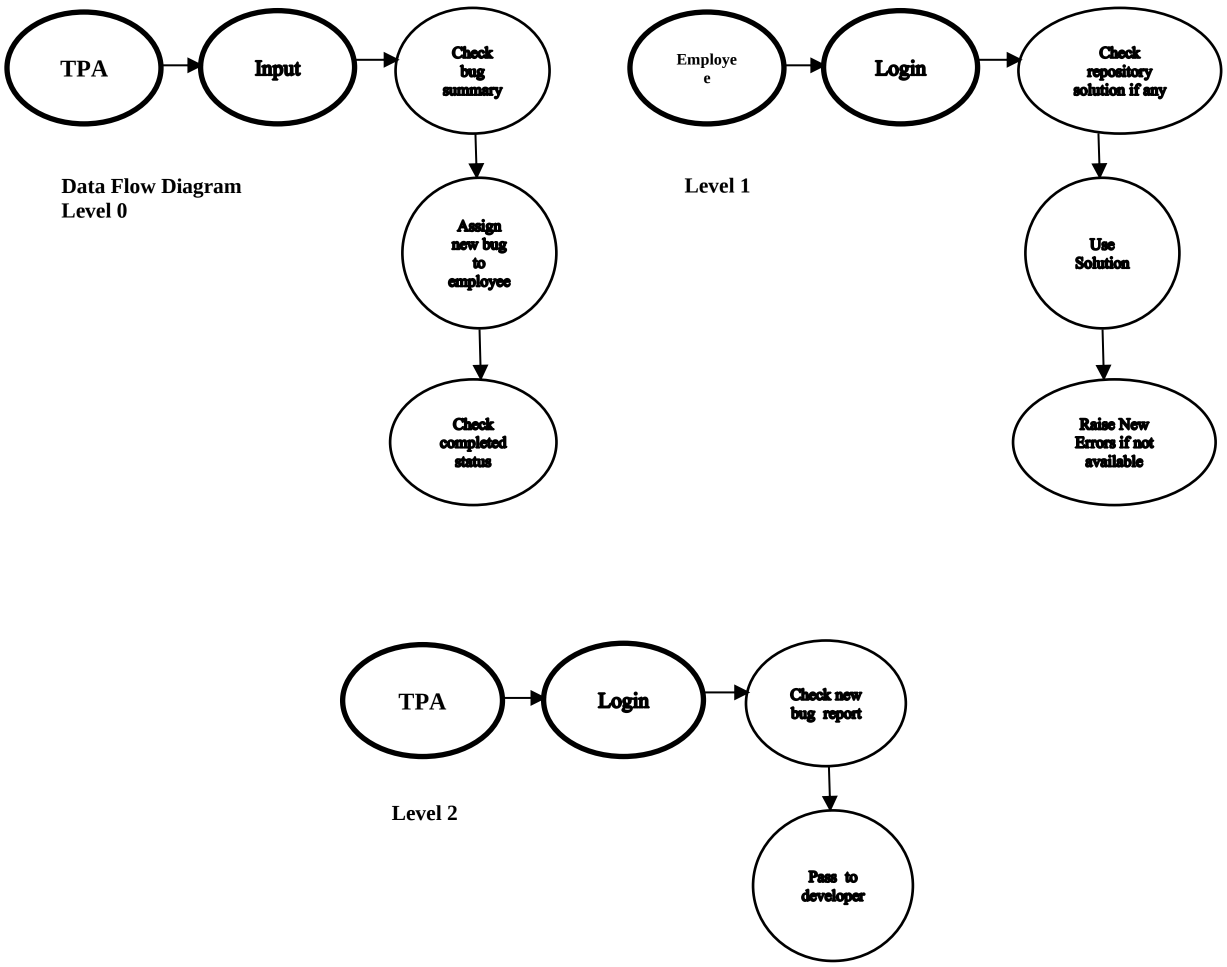

Figure 1: Several instances of information flows by 3 different level

Results and Discussion

i. **Bug understanding:** By applying software reduction techniques, developers can isolate and extract the relevant portions of the codebase associated with a particular bug. This helps in gaining a better understanding of the bug's root cause, making it easier to analyze and fix the issue.

ii. **Faster bug reproduction:** Software reduction techniques can reduce the size and complexity of the software involved in a bug, making it easier to reproduce the bug in a controlled environment. This can significantly speed up the bug reproduction process, enabling developers to investigate and diagnose the issue more efficiently.

iii. **Reduced debugging time:** With a simplified and reduced codebase, developers can focus their efforts on the essential parts of the software related to the bug. This targeted approach can save considerable debugging time by eliminating irrelevant code and reducing the overall complexity of the system.

iv. **Enhanced bug triage prioritization:** Software reduction techniques help in identifying the critical components and dependencies affected by a bug. This information allows for better bug prioritization, enabling developers to allocate resources effectively and address high-impact issues first.

v. **Increased collaboration efficiency:** By reducing the codebase size, software reduction techniques facilitate better collaboration among developers, testers, and other stakeholders involved in the bug triage process. Smaller code snippets are easier to share, understand, and discuss, leading to more effective collaboration and faster bug resolution.

vi. **Resource optimization:** Effective bug triage using software reduction techniques can result in optimized resource allocation. By minimizing the scope of investigation to relevant code segments, developers can avoid spending excessive time and effort on non-essential areas, thereby utilizing their resources more efficiently.

Conclusion

The step in the software maintenance process where bugs are ranked in order of importance is referred to as the bug triage phase, and it is one of the most demanding and time-consuming phases. This work integrates feature selection and instance selection techniques to decrease the quantity of datasets related to problems, while simultaneously improving the quality of the information that can be extracted from the samples. This method involves initially gathering attributes from each unique bug data set and subsequently training a prediction model using previous data sets. It enables us to determine the sequence in which instance selection and feature selection should be implemented on a new bug data set. This allows us to choose the most effective technique for analyzing a new set of bug data. This process could serve as the basis for our methodology in future investigations. This article provides an empirical analysis of the data reduction methods employed in the bug repositories of two renowned open-source projects, Mozilla and Eclipse. Our study presents a methodology that can improve the accuracy of bug data while also lowering the amount of data associated with software development and maintenance. The main emphasis is on the utilization of pre-existing data processing techniques. Our study findings offer a precise approach for attaining this objective. As part of our continuous endeavors, one of our objectives is to enhance the effectiveness of the data reduction employed in the bug triage process. This suggests that we can explore the most efficient approaches for generating a comprehensive and reliable collection of bug data, as well as executing a software task that is specific to a certain field. In order to enhance our ability to accurately forecast reduction orders, it is imperative that we explore potential correlations between reduction orders and the characteristics of bug data sets. This discovery will enhance our capacity to accurately forecast lower-level phenomena.

References:

S. Amershi *et al.*, “Software Engineering for Machine Learning: A Case Study,” *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, May 2019, doi: https://doi.org/10.1109/icse-seip.2019.00042.

Z. Jia *et al.*, “Detecting Error-Handling Bugs without Error Specification Input,” *IEEE Xplore*, Nov. 01, 2019. https://ieeexplore.ieee.org/abstract/document/8952517 (accessed Oct. 13, 2022).

X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, “Improving Automated Bug Triaging with Specialized Topic Model,” *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 272–297, Mar. 2017, doi: https://doi.org/10.1109/tse.2016.2576454.

H. Wu, Y. Ma, Z. Xiang, C. Yang, and K. He, “A spatial–temporal graph neural network framework for automated software bug triaging,” *Knowledge-Based Systems*, vol. 241, p. 108308, Apr. 2022, doi: https://doi.org/10.1016/j.knosys.2022.108308.

Z. Hassan, N. Iqbal, and Abnash Abnash, “Towards Effective Analysis and Tracking of Mozilla and Eclipse Defects using Machine Learning Models based on Bugs Data,” *Soft Computing*, vol. 1, no. 1, pp. 1–10, Feb. 2021.

Guo, S., Zhang, X., Yang, X., Chen, R., Guo, C., Li, H., & Li, T. (2020). Developer activity motivated bug triaging: via convolutional neural network. *Neural Processing Letters*, *51*, 2589-2606.

M. Soltani, F. Hermans, and T. Bäck, “The significance of bug report elements,” *Empirical Software Engineering*, vol. 25, no. 6, pp. 5255–5294, Sep. 2020, doi: https://doi.org/10.1007/s10664-020-09882-z.

Goyal and N. Sardana, “Analytical Study on Bug Triaging Practices,” *International Journal of Open Source Software and Processes*, vol. 7, no. 2, pp. 20–42, Apr. 2016, doi: https://doi.org/10.4018/ijossp.2016040102.

Goyal, A., & Sardana, N. (2021). Bug handling in service sector software. In *Research Anthology on Recent Trends, Tools, and Implications of Computer Programming* (pp. 1941-1960). IGI Global.

A. Yadav, M. Baljon, S. Mishra, S. K. Singh, S. Saxena, and S. K. Sharma, “Developer load balancing bug triage: Developed load balance,” Expert Systems, May 2022, doi: https://doi.org/10.1111/exsy.13006.