

SPRINT: An Assistant for Issue Report Management

Ahmed Adnan

University of Dhaka

Dhaka, Bangladesh

bsse1131@iit.du.ac.bd

Antu Saha

William & Mary

Williamsburg, Virginia, USA

asaha02@wm.edu

Oscar Chaparro

William & Mary

Williamsburg, Virginia, USA

oscarch@wm.edu

Abstract—Managing issue reports is essential for the evolution and maintenance of software systems. However, manual issue management tasks such as triaging, prioritizing, localizing, and resolving issues are highly resource-intensive for projects with large codebases and users. To address this challenge, we present SPRINT, a GitHub application that utilizes state-of-the-art deep learning techniques to streamline issue management tasks. SPRINT assists developers by: (i) identifying existing issues similar to newly reported ones, (ii) predicting issue severity, and (iii) suggesting code files that likely require modification to solve the issues. We evaluated SPRINT using existing datasets and methodologies, measuring its predictive performance, and conducted a user study with five professional developers to assess its usability and usefulness. The results show that SPRINT is accurate, usable, and useful, providing evidence of its effectiveness in assisting developers in managing issue reports. SPRINT is an open-source tool available at github.com/sea-lab-wm/sprint_issue_report_assistant_tool.

I. INTRODUCTION

Issue reports are essential artifacts to identify, track, and resolve issues in software systems [1–4]. Given their importance, managing issue reports is critical for maintaining and evolving software systems [5, 6]. Key issue management activities include labeling issues that report similar problems, categorizing them based on severity, and identifying potential buggy locations in the code [1, 7, 8]. While important, manual issue management is a time-consuming and challenging process, especially for projects that typically receive hundreds of issues affecting various software components.

To help developers manage issues, researchers have proposed a variety of techniques to automate issue report management tasks [9]. Practitioners have also proposed a variety of tools [10–14], standalone or integrated into existing systems (*e.g.*, issue trackers), to support these tasks. However, most of these tools are designed to support specific tasks and their underlying models are not easy to update. As such, these tools do not integrate different techniques and support for various issue management tasks in a single, easy-to-extend, comprehensive solution.

To address these limitations, we introduce SPRINT, an integrated GitHub application for issue management. SPRINT is designed as a comprehensive solution that can be easily extended and adapted to support multiple issue management tasks. In its current version, SPRINT leverages state-of-the-art deep learning techniques to assist developers in (i) identifying issues similar to the newly reported issue, (ii) predicting issue severity, and (iii) localizing the potential code files that require modification to solve the reported problem. SPRINT gives the developers suggestions as comments in issue reports and as

labels attached to the issues to facilitate issue management. SPRINT is extensible due to its modularized plugin-based architecture, which includes well-designed APIs and scripts that enable easy feature integration and extensibility. SPRINT is also scalable, allowing multiple users and repositories to leverage its features concurrently.

We evaluated the predictive performance of SPRINT’s underlying state-of-the-art models by replicating the evaluations of their respective papers. Additionally, we conducted a user study with five professional developers experienced in issue management, who found SPRINT easy to use, useful, and practical for issue management and resolution. SPRINT is an open-source tool hosted on GitHub [15] and can be easily installed in any GitHub repository.

II. SPRINT: AN ISSUE REPORT MANAGEMENT TOOL

A. Supported Issue Management Tasks

SPRINT is an issue management assistant for developers, project managers, computer science students, and educators. SPRINT currently supports three issue management tasks.

1) *Issue Severity Prediction*: SPRINT classifies the reported issues based on their severity level. After a user creates a new issue, the tool tags it with one of five labels: Blocker, Critical, Major, Minor, or Trivial. This helps project managers and developers prioritize the issues that require immediate attention.

2) *Similar Issue Identification*: SPRINT suggests similar issues as soon as a new issue is submitted, tagging the new issue with a “Duplicate” label. Similar issues are suggested in a comment in the issue report. This feature minimizes redundant issue management efforts by suggesting related issues to developers and reporters, who are meant to inspect the results to determine if the new issue was reported before.

3) *Buggy Code Localization*: SPRINT suggests potential buggy code files based on the textual similarity between the reported issue and the code file of the system’s latest version. When a new issue is reported, SPRINT fetches the files of the latest system version and ranks them as a list of potential buggy code files. This feature suggests users code files that might require modification to solve the issues.

B. Usage Scenario & Graphical User Interface

SPRINT can be easily installed in any repository. A user only needs to visit the installation website [16], click the ‘Install’ button, and select the repositories where the tool will be used without requiring any configuration.

Steps to Reproduce Calculating Window Bug #70

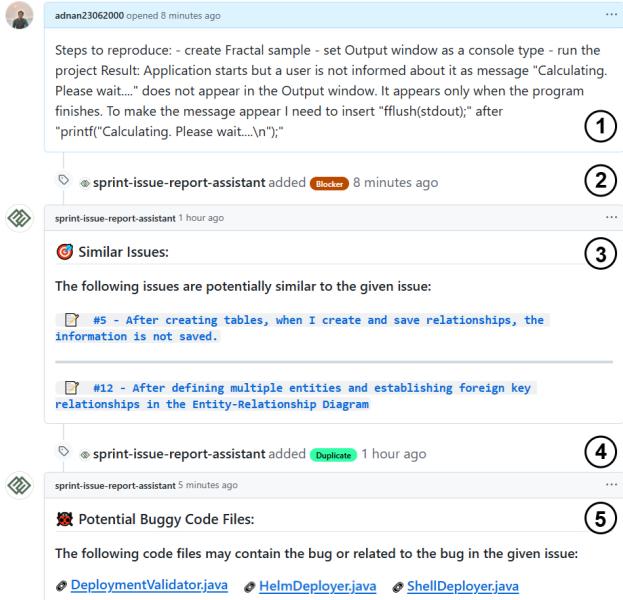


Fig. 1: SPRINT’s GUI with a Usage Scenario

After installation, when a user reports a new issue (see ① in fig. 1), SPRINT’s first step is to fetch the issue’s title and description, and the code files of the system’s latest version. SPRINT then generates comments with feedback to the user by analyzing this information using state-of-the-art models (described in section III).

First, SPRINT analyzes the new issue and classifies it into one of five severity levels (Blocker, Critical, Major, Minor, or Trivial [17]), assigning a label with predicted severity ②. This label follows a color-coding format from red to yellow, where red suggests the issue is very severe and yellow indicates the issue is trivial. Second, SPRINT analyzes the new issue and all the existing issues in the project, suggesting which of these are similar to the new issue. Then, it generates a comment with a list of suggested similar issues ③, each including its issue ID, title, and URL. If one or more similar issues are found, the new issue is labeled as “Duplicate” ④. Third, SPRINT identifies potential buggy code files by analyzing the reported issue’s information and the paths and names of the repository’s code files. SPRINT generates a comment displaying the list of files (with URLs) that may need modification to solve the issue ⑤. SPRINT’s features are independent of one another: no feature is dependent on the execution of others.

SPRINT suggests severity labels, similar issues, and potential buggy code files for the issues created after installation, whenever a new issue is submitted. SPRINT does not generate comments or labels for the issues existing before installation because these suggestions might conflict with comments and labels manually created by developers and reporters. SPRINT currently handles the code files of the system’s latest version. A future tool improvement is to identify the affected system

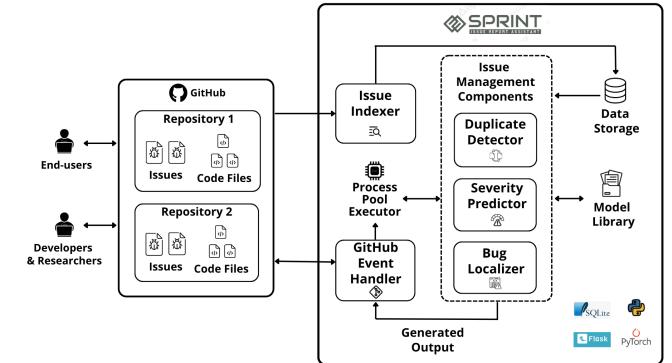


Fig. 2: Overview of SPRINT’s Architecture

version specified in the issue and perform bug localization on that version’s code files.

III. SPRINT’S ARCHITECTURE & IMPLEMENTATION

A. Architecture

SPRINT’s architecture, shown in fig. 2, consists of three main components: (1) the Issue Indexer, (2) the GitHub Event Listener, and (3) the Issue Management Components.

1) Issue Indexer: When SPRINT is installed in one or more repositories, this component fetches all the existing issues from those repositories using GitHub Webhooks [18] and stores them in a local relational database. This component applies page-based indexing to partition issues into manageable groups for efficient fetching. The database is meant to facilitate quick access and analysis of issues. This database is permanently synchronized with GitHub to provide 24/7 support since new issues are reported continuously and concurrently.

2) GitHub Event Handler: This component is responsible for listening to and handling the GitHub repository events when a new issue is submitted. This component fetches a newly reported issue along with the latest version’s code files and sends them to the Issue Management Components for further analysis. After the analysis, this component processes the feedback, formats it appropriately, and posts the generated comments and labels to the reported issue as the final output.

3) Issue Management Components: There are three components for issue management in the current version of SPRINT.

Similar Issue Detection: This component takes a newly created issue (title & description) from the GitHub Event Handler and compares it with each issue stored in the database by analyzing textual similarity between them. For this task, SPRINT uses the RTA classification model [19], fine-tuned on the RTA duplicate bug report training dataset [20]. SPRINT uses a Process Pool Executor [21], a multiprocessing component, to analyze multiple issue pairs concurrently. After the analysis, this component returns the duplicate issues to the GitHub Event Handler.

Issue Severity Prediction: This component receives a newly reported issue (issue title & description) from the GitHub Event Handler and utilizes the RTA classification model [19], fine-tuned on RTA severity prediction training dataset [20], to classify the issue into one of five severity levels: blocker, critical,

major, minor, and trivial. This component returns the predicted issue severity level to the Event Handler. For similar issue detection and severity prediction tasks, we selected RTA [19] because of its state-of-the-art predictive performance on large open-source issue reports and its speed and efficiency in issue analysis. It learns the fundamental representation of bug reports via a dynamic masked language model and contrastive learning objectives in a self-supervised manner.

Bug Localization: This component receives a newly created issue (title & description) and the code files of the latest system version from the GitHub Event Handler. For bug localization, SPRINT uses the bug localization approach proposed by Bogomolov *et al.* [22]: a Llama-2-7b-chat [23] model, fine-tuned on their bug localization dataset [24]. The bug localization component constructs a prompt including the issue contents and the list of code file paths and names. With this prompt, the model generates a ranked list of potential buggy code files that might need further inspection to solve the issue and returns it to the Event Handler. The prompt asks the model to compare the textual semantics between the issue content and repository code file paths/names to predict potential buggy code files.

B. SPRINT’s Extensibility & Scalability

SPRINT’s architecture separates the issue management components from the components used for GitHub integration and event handling. The features are implemented as modular APIs with comprehensive documentation to simplify tool feature addition and enhancement. Our tool also conforms with the plugin architecture [25] to enable easy extensibility. The event handler serves as the host component, while the issue management component APIs serve as the plugin interface. As per plugin architecture, SPRINT’s GitHub application supports dynamic loading and communication protocols, ensuring isolation between the host and plugins. Moreover, in the tool repository [15], we provided scripts that can be used to fine-tune other transformer-based models for issue management tasks. With this, project owners can easily integrate the resulting models into our tool by replacing the model paths in the tool’s configuration file.

SPRINT’s backend utilizes Python’s Process Pool Executor, which applies multiprocessing to concurrently handle multiple issues from various repositories. Project owners can configure the pool size based on their computational resources and workload requirements, by changing the configuration file, allowing SPRINT to be responsive even during simultaneous requests. This multiprocessing technique also facilitates duplicate detection by notably reducing processing time and ensuring fast responses since pairwise issue comparison is computationally intensive.

C. Implementation

SPRINT is implemented using Python’s Flask framework [26]. SPRINT is developed as a GitHub Application, using GitHub’s Webhooks that allow integration with GitHub’s issues tracker [18]. For data storage, SPRINT uses the relational database system, SQLite3. Currently, SPRINT is running on a

modest server and can support five to six repositories concurrently. For production, it needs to be deployed in a robust server, possibly in a cloud infrastructure. Though SPRINT is tailored for handling GitHub issue management, most of its backend components can be easily adapted for other issue trackers.

IV. SPRINT’S EVALUATION

We conducted a preliminary evaluation to measure SPRINT’s models’ predictive performance as well as SPRINT’s usability and usefulness. SPRINT’s GitHub repository provides details of the evaluation methodology, results, and necessary data to replicate and verify the evaluation [15].

A. Model Evaluation

To select appropriate models for our features, we conducted an extensive literature review, experimented with various models, and chose those that displayed the best predictive performance. Our approach was to replicate the original evaluation of the models by following the methodologies and test datasets provided in their respective papers. For the similar issue detection feature, we chose the RTA model [19] fine-tuned in RTA’s duplicate issues training dataset [20]. Their test dataset [20] has 15,288 issue pairs (approximately 60% of the pairs were duplicates, and the remaining were non-duplicates) from 6 large open-source projects. Overall, we obtained 97.3% accuracy, 97.5% precision, and 98.8% recall. For issue severity prediction, we selected the RTA model [19] fine-tuned in RTA’s issue severity training dataset [20]. For this task, the test dataset has 15,510 issues of 6 projects with 5 severity classes. The distribution of the severity classes was between 15% and 24%. Overall, we obtained 65.6% accuracy, 67.3% precision, and 64.6% recall. For the bug localization feature, we fine-tuned the Llama-2-7b-chat model on the dataset provided by Bogomolov *et al.* [22], and we evaluated the model’s prediction capability on their 150 test issues [24]. We achieved 34% accuracy, 20% Recall@2 (R@2), 31% Precision@2 (P@2), and a MAP of 29%.

B. User Study

We conducted a user study involving five professional developers with three to eight years of issue management experience who work on different projects at Samsung Research Bangladesh. The goal of the study was to evaluate (i) SPRINT’s usefulness/usability (RQ1/RQ2) and (ii) the predictive accuracy of SPRINT’s suggestions (RQ3).

1) Methodology: For the similar issue detection feature, we first selected 2 issues as queries and their respective duplicates and 8 non-duplicate issues from the RTA [19] model’s duplicate bug report test dataset [20]. These issues come from the OpenOffice project [27]. To evaluate the severity prediction feature, focusing on the same project, we first categorized the RTA [19] model’s issue severity test dataset [20] into 2 groups: (i) issues for which the model predicted the severity correctly and (ii) issues where the model mispredicted the severity. Then, we chose one issue from each group randomly. For the bug localization feature, we chose one of the projects from Bogomolov *et al.*’s dataset [24]: wso2/testgrid [28]. Each

issue in this project had two buggy code files on average. Then, we ran the bug localization model on each issue three times since LLM outputs can vary slightly each time and selected the common suggestions that occurred in all three runs. After that, we divided these issues into two groups: (i) issues for which the model predicted at least one correct buggy code file (*i.e.*, present in the ground truth) in its top-5 suggestions and (ii) issues where the model failed to predict any buggy code files in the top-5 suggestions. We selected one issue from each group as queries. Our goal in selecting both successful and unsuccessful cases across various features was to ensure a fair evaluation, allowing participants to experience these scenarios and better assess the tool’s usefulness and usability.

In the study, the participants first were introduced to the tool with detailed guidelines. Then, they engaged in answering a questionnaire that assessed the accuracy, usefulness, and usability of the tool’s suggestions. The study survey questions were a combination of Likert-scale and open-ended questions. We provided participants with ground truth data for reference.

2) *RQ1/RQ2 Results:* The participants (*a.k.a.* users) evaluated how easy-to-use and practical SPRINT is.

Readability of SPRINT’s Suggestions: Of five users, three rated SPRINT’s suggestions for the three features as “very easy” to understand, while two rated them as “moderately easy” (on a 5-point Likert scale, these options are the most positive).

SPRINT Overall Usability: Four users found SPRINT easy to use, while one user was unsure. All the users agreed that SPRINT’s feedback comments were easy to understand and displayed useful information.

SPRINT Responsiveness: SPRINT’s average response time for showing results for all three features was approximately 90 seconds (for a repository with ≈ 20 issues and ≈ 50 code files). Four users found SPRINT “very responsive” and one user found it “moderately responsive” (on a 5-point Likert scale, these are the most positive options).

SPRINT Usefulness for Issue Management: Four users “completely agreed,” while one user “somewhat agreed” with the statement: “The combination of SPRINT’s three features is helpful for issue management”.

New feature suggestions: The participants suggested additional features for SPRINT, such as automated issue content identification (*e.g.*, identifying the reproduction steps), quality assessment, automated program repair, and developer recommendations for issue resolution.

3) *RQ3 Results:* The participants evaluated the accuracy of the suggestions made by the SPRINT’s three main features.

Similar Issue Identification: All the users agreed that SPRINT correctly suggests all the similar issues for the two queries. However, the reporters found that SPRINT suggests one or two extra non-duplicates for the first query—though, it perfectly predicts the two expected similar issues for the second query.

Severity Prediction: Three of five users agreed that SPRINT correctly predicted the severity of the issues, whereas two users were unsure about the predictions. They stated that SPRINT correctly identified the severity class for one query and mispredicted the other, but the misprediction was close to the actual severity class (SPRINT predicted ‘Trivial’ instead of ‘Minor’).

Bug Localization: All five users identified correct and incorrect predictions, aided by the ground truth references provided in the study survey. Two users identified one correct prediction in the top-5 suggestions sufficient, while three others suggested improving SPRINT’s accuracy. Overall, all participants valued the feature’s responsiveness and potential usefulness.

V. RELATED WORK

Existing tools support individual issue management tasks. Find Duplicates [10], Probot [11], NextBug [12] are plugins for GitHub, Jira [29], and Bugzilla [30] that identify related issues. These tools rely on information retrieval or classical machine learning models that process issue text to determine issue similarity. Priority Scheduler [13] is a Jira [29] plugin that assigns priority based on project deadlines. BugLocalizer [31] is a Bugzilla [30] extension that analyzes bug reports and source code similarities to identify buggy files. PR-Agent [14] is a multi-featured paid tool that applies LLMs, *e.g.*, ChatGPT [32], to perform tasks such as pull request change classification, automatic code review, and documentation. Additionally, there are tools for reporting [33, 34], identifying bug report components [35], and assessing bug reproduction steps [36, 37]. Researchers have proposed automated techniques for duplicate issue detection [19, 38, 39], severity prediction [19, 40, 41], bug localization [22, 42], issue categorization [43, 44], and other issue management tasks [45, 46]. We selected state-of-the-art models, namely RTA [19] and LongCodeArena [22] for SPRINT’s three features, based on a rigorous literature review found in our replication package [15].

Compared to prior tools, SPRINT stands out as an open-source, easy-to-install solution that seamlessly integrates with GitHub and consolidates multiple features, making it a comprehensive assistant for issue management. Moreover, SPRINT leverages state-of-the-art models that have demonstrated superior performance compared to prior proposed approaches.

VI. CONCLUSIONS & FUTURE WORK

SPRINT is an integrated open-source GitHub application that leverages state-of-the-art models to identify similar issues, predict issue severity, and localize potential buggy code files for a new issue. It aims to support developers in managing and resolving issues. Evaluation results indicate that SPRINT provides valuable assistance in terms of predictive accuracy and user experience. For future work, we plan to enhance the performance of SPRINT’s bug localization feature by incorporating more advanced models. Additionally, we aim to extend SPRINT’s compatibility to platforms beyond GitHub.

ACKNOWLEDGMENTS

This work is supported by U.S. NSF grants CCF-2239107 and CCF-1955853. The opinions, findings, and conclusions expressed in this paper are those of the authors and do not necessarily reflect the sponsors’ opinions.

REFERENCES

- [1] R. K. Saha, J. Lawall, S. Khurshid, and D. E. Perry, "On the effectiveness of information retrieval based bug localization for c programs," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 161–170.
- [2] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng, "Detecting missing information in bug descriptions," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017, 2017, p. 396–407.
- [3] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, pp. 308–318.
- [4] A. Saha and O. Chaparro, "Decoding the issue resolution process in practice via issue report analysis: A case study of firefox," in *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering (ICSE'25)*, 2025.
- [5] N. Strauß and J. Jonkman, "The benefit of issue management: anticipating crises in the digital age," *Journal of Communication Management*, vol. 21, pp. 34–50, 02 2017.
- [6] J. Zhang, X. Wang, D. Hao, B. Xie, and Zhang, "A survey on bug-report analysis," *Sci. China Inf. Sci.*, vol. 58, no. 2, pp. 1–24, 2015.
- [7] N. Bettenburg, R. Premraj, T. Zimmermann, and . Sunghun Kim, "Duplicate bug reports considered harmful ... really?" in *Proceedings of the IEEE International Conference on Software Maintenance*, 2008, pp. 337–345.
- [8] A. F. Otoom, S. Al-jdaeh, and M. Hammad, "Automated classification of software bug reports," in *proceedings of the 9th international conference on information communication and management*, 2019, pp. 17–21.
- [9] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu, "How practitioners perceive automated bug report management techniques," *IEEE Transactions on Software Engineering*, vol. 46, no. 8, pp. 836–862, 2020.
- [10] "Find duplicates: Detect similar issues, find related issues," <https://marketplace.atlassian.com/apps/1212706/find-duplicates-detect-similar-issues-find-related-issues?tab=overview&hosting=datacenter>, 2024.
- [11] "Probot/duplicate-issues," <https://github.com/probot/duplicate-issues>, 2024.
- [12] H. Rocha, G. d. Oliveira, H. Marques-Neto, and M. T. Valente, "Nextbug: a bugzilla extension for recommending similar bugs," *Journal of Software Engineering Research and Development*, vol. 3, pp. 1–14, 2015.
- [13] "Priority scheduler: Easy issue priority management: Change priority by date." <https://marketplace.atlassian.com/apps/1233172/priority-scheduler?hosting=cloud&tab=overview>.
- [14] "Pr-agent," <https://github.com/Codium-ai/pr-agent>, 2024.
- [15] "Sprint github repository," https://github.com/sea-lab-wm/sprint_issue_report_assistant_tool, 2025.
- [16] "Sprint: Issue report assistant tool," <https://github.com/apps/sprint-issue-report-assistant>, 2024.
- [17] "Severity & priority types of bug reports," <https://www.javatpoint.com/severity-and-priority-in-testing>.
- [18] "About writing code for a github application," <https://docs.github.com/en/apps/creating-github-apps/writing-code-for-a-github-app/about-writing-code-for-a-github-app>.
- [19] S. Fang, T. Zhang, Y. Tan, H. Jiang, X. Xia, and X. Sun, "Representthemall: A universal learning representation of bug reports," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 602–614.
- [20] "Duplicate bug report dataset by rta," <https://drive.google.com/drive/folders/1gPnZbgOO4XiBBsyF27jS-XwhHaInxlQ>, 2023.
- [21] "Severity & priority types of bug reports," <https://www.geeksforgeeks.org/processpoolexecutor-class-in-python/>.
- [22] E. Bogomolov, A. Eliseeva, T. Galimzyanov, E. Glukhov, A. Shapkin, M. Tigina, Y. Golubev, A. Kovrigin, A. van Deursen, M. Izadi *et al.*, "Long code arena: a set of benchmarks for long-context code models," *arXiv preprint arXiv:2406.11612*, 2024.
- [23] "Llama-7b-chat-hf;" <https://huggingface.co/meta-llama/Llama-2-7b-chat-hf>.
- [24] "Bug localization dataset of long code arena by jetbrains," <https://huggingface.co/datasets/JetBrains-Research/lca-bug-localization>, 2024.
- [25] "Severity & priority types of bug reports" <https://medium.com/omarelgabrys-blog/plug-in-architecture-dec207291800>.
- [26] "Welcome to flask's documentation," <https://flask.palletsprojects.com/en/stable/>.
- [27] "Apache openoffice github repository," <https://github.com/apache/openoffice>.
- [28] "wso2/testgrid project repository," <https://github.com/wso2/testgrid>.
- [29] "Jira: Issue & project tracking software," <https://www.atlassian.com/software/jira>.
- [30] "Bugzilla: The software solution designed to drive software development," <https://www.bugzilla.org/>.
- [31] F. Thung, T.-D. B. Le, P. S. Kochhar, and D. Lo, "Buglocalizer: Integrated tool support for bug localization," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 767–770.
- [32] "Introducing chatgpt — openai," <https://openai.com/index/chatgpt>.
- [33] Y. Song, J. Mahmud, N. De Silva, Y. Zhou, O. Chaparro, K. Moran, A. Marcus, and D. Poshyvanyk, "Burt: A chatbot for interactive bug reporting," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2023, pp. 170–174.
- [34] Y. Song, J. Mahmud, Y. Zhou, O. Chaparro, K. Moran, A. Marcus, and D. Poshyvanyk, "Toward interactive bug reporting for (android app) end-users," in *Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering*, 2022, pp. 344–356.
- [35] Y. Song and O. Chaparro, "Bee: A tool for structuring and analyzing bug reports," in *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2020, pp. 1551–1555.
- [36] J. Mahmud, A. Saha, O. Chaparro, K. Moran, and A. Marcus, "Combining language and app ui analysis for the automated assessment of bug reproduction steps," in *Proceedings of the 33rd IEEE/ACM International Conference on Program Comprehension (ICPC'25)*, 2025.
- [37] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. Ng, "Assessing the quality of the steps to reproduce in bug reports," in *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 86–96.
- [38] T. Zhang, D. Han, V. Vinayakarao, I. C. Irsan, B. Xu, F. Thung, D. Lo, and L. Jiang, "Duplicate bug report detection: How far are we?" *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 4, pp. 1–32, 2023.
- [39] I. M. Rodrigues, D. Aloise, E. R. Fernandes, and M. Dagenais, "A soft alignment model for bug deduplication," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 43–53.
- [40] J. Kim and G. Yang, "Bug severity prediction algorithm using topic-based feature selection and cnn-lstm algorithm," *IEEE Access*, vol. 10, pp. 94 643–94 651, 2022.
- [41] A. Ali, Y. Xia, Q. Umer, and M. Osman, "Bert based severity prediction of bug reports for the maintenance of mobile applications," *Journal of Systems and Software*, vol. 208, p. 111898, 2024.
- [42] J. Mahmud, N. De Silva, S. A. Khan, S. H. Mostafavi, S. H. Mansur, O. Chaparro, A. Marcus, and K. Moran, "On using gui interaction data to improve text retrieval-based bug localization," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.
- [43] K. Somasundaram and G. C. Murphy, "Automatic categorization of bug reports using latent dirichlet allocation," in *Proceedings of the 5th India software engineering conference*, 2012, pp. 125–130.
- [44] G. Catolino, F. Palomba, A. Zaidman, and F. Ferrucci, "Not all bugs are the same: Understanding, characterizing, and classifying bug types," *Journal of Systems and Software*, vol. 152, pp. 165–181, 2019.
- [45] B. Chaitra and K. Swarnalatha, "Bug triaging: right developer recommendation for bug resolution using data mining technique," in *Emerging Research in Computing, Information, Communication and Applications: ERCICA 2020, Volume 2*. Springer, 2022, pp. 609–618.
- [46] A. Saha, Y. Song, J. Mahmud, Y. Zhou, K. Moran, and O. Chaparro, "Toward the automated localization of buggy mobile app uis from bug descriptions," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 1249–1261.