

Received 3 December 2024, accepted 13 December 2024, date of publication 17 December 2024,
date of current version 30 December 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3519418

RESEARCH ARTICLE

Implementing and Evaluating Automated Bug Triage in Industrial Projects

HYUN-TAEK HONG^{1,2}, DAE-SUNG WANG¹, SE-JIN KIM¹, HOON SUNG¹,
CHANG-WON PARK², HO-HYUN PARK³, AND CHAN-GUN LEE¹

¹Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea

²Vehicle Solution Company, LG Electronics, Seoul 07796, South Korea

³School of Electrical and Electronics Engineering, Chung-Ang University, Seoul 06974, South Korea

Corresponding author: Chan-Gun Lee (cglee@cau.ac.kr)

This work was supported in part by the Chung-Ang University Research Scholarship Grants in 2022, in part by the National Research Foundation of Korea (NRF) grant funded by the Korea Government Ministry of Science and ICT (MSIT) under Grant RS-2024-00342059, and in part by the LG Electronics.

ABSTRACT Resolving bugs on time is essential for software development and is critical in industrial projects because it directly affects businesses. Automatic bug triage has been investigated to increase software productivity, and research has become more active as machine learning techniques have improved. However, most research has focused on open-source projects, whereas studies on industrial projects remain limited. The research gap in previous studies is that the research has directly triaged developers, reducing accuracy in industrial projects where organizational structures frequently change. Moreover, developers often move between teams, making this approach less effective. The research in this article applies automatic bug triage to industrial projects by adapting the characteristics of industrial projects. Addressing these limitations establishes an approach that is better suited to industrial projects and has enhanced accuracy. Based on this background, we propose a novel approach to triage developers associated with component-based developer lists. Each component has an associated list of developers, and the triage results of the model are limited to selecting from among the listed developers, enhancing triage accuracy. The proposed approach reflects the characteristics of industrial projects and addresses the dynamic workload adjustments in a component-based team structure. The proposed approach improves the results by 6.2 percentage points over human triage for top-1 results, suggesting that this approach could be further expanded for broader application in industrial contexts. Future research should focus on refining the proposed method with real-time feedback and experiment with a broader dataset for generalizability and scalability.

INDEX TERMS Bug triage, industrial project, software engineering, pretrained language model, component.

I. INTRODUCTION

As software size increases exponentially, the number of bugs also rises significantly. These bugs affect the software delivery schedule; therefore, how quickly and effectively these bugs are resolved significantly contributes to the timely distribution of the software. In particular, in industrial projects related to business, failing to meet the delivery timeline can result in direct burdens, such as cost penalties, critically affecting the business. Bug fixing is time-consuming, and it is difficult to estimate the overall effort.

The associate editor coordinating the review of this manuscript and approving it for publication was Mahmoud Elish.

Weiss et al. [1] and Giger et al. [2] have attempted to estimate the effort. However, generalizing their findings is complicated because the effort depends on several factors, such as the quality of the bug report.

Most software development projects use application lifecycle management (ALM) systems, such as Jira,¹ Codebeamer,² and Polarion,³ to manage artifacts, including requirements, design, test cases, and bug reports. Using ALM ensures that 1) requirements are accurately reflected

¹<https://www.atlassian.com/software/jira>

²<https://codebeamer.com/>

³<https://polarion.plm.automation.siemens.com/>

in the design, 2) corresponding test cases are created, and 3) schedules are managed effectively based on these outputs, increasing software productivity. Therefore, in an ALM system, the timely resolution of bugs is essential, and bug triage activities, such as analyzing bug descriptions, setting priorities, and assigning the appropriate developer, are crucial.

Automatic bug triage techniques have been studied to facilitate bug fixing activities. Anvik et al. [3] conducted the initial study, and following that study, several automatic bug triage studies [4], [5] have been published. However, automatic bug triage has not been widely adopted in industrial projects due to several difficulties such as accuracy concerns and frequent organizational changes.

However, with recent advancements in natural language processing and machine learning techniques, interest in the effectiveness of automatic bug triage is growing. For example, deep learning-based natural language processing models, such as the bidirectional encoder representations from transformers (BERT) [6] and the robust optimized BERT approach (RoBERTa) [7], which understand and analyze the overall context, have significantly improved triage performance [8], [9] compared with earlier word-based research [10], [11]. Such approaches typically rely on access to large datasets for model training and validation, so they are primarily conducted with open-source projects. However, research opportunities for industrial projects are limited [12] due to the challenge of accessing data.

The developer organization is typically structured globally, creating a matrix structure comprising project- or feature-specific combinations. Each suborganization operates in an agile scrum [13], [14], allowing different developers in the same scrum to resolve similar bugs. Developers often move between teams according to their personal or company reasons. Therefore, triaging to a specific developer may not be suitable for industrial projects. Regarding human organization, industrial projects often manage a professional pool of developers and testers with high domain expertise, resulting in more accurate bug reports and analyses in verification and development.

Industrial projects generally have shorter durations, with most spanning under 100 months (e.g., Project A: 44 months, B: 45 months, and C: 28 months, as examined in this study), whereas open-source projects often extend more than 100 months, such as Mozilla Core (217 months) and Mozilla Firefox (204 months). Bugs in industrial projects are fixed faster than in open-source projects (Tables 1 and 2). In addition, the quality of bug reports is often higher in industrial projects than in open-source projects [10] because professional testers author them. Moreover, continuous maintenance of essential reporting items is enforced in the organization.

In summary, several aspects should be considered in industrial projects compared with open-source projects: 1) higher accuracy and human expertise to meet shorter project durations and 2) a team structure-based approach. Section II

details the characteristics of industrial projects compared to open-source projects.

The component field in the bug tracking system is typically predefined and helps triagers identify where the bugs are most likely to be. Wang et al. [15] tried to predict the component of a bug using bug report mining to determine the correct component rather than using the user-selected component. Using stack traces and categorical attributes, Sabor et al. [16] attempted to predict bug report fields, including the component field. These studies focused on predicting the component field but did not consider that multiple developers are assigned to each component.

Based on this background, the proposed approach combines the component information for automatic bug triage in industrial projects. This approach triages developers based on a pretrained language model and matches the results with the component-level developer list. Managing the developer list at the component level reduces the scope of the candidate developer list, improving triage accuracy.

This research addresses the three research questions below:

- **RQ 1:** Is the assignee prediction using the proposed method better than the human triager in industrial projects? This approach improves the results by 6.2%p for top-1 results, proving that the proposed method enhances the performance of automatic bug triage.
- **RQ 2:** How is the bug triage performance of industrial and open-source projects differentiated? This work compares the differences and analyzes the context according to model performance and report quality.
- **RQ 3:** What are the special considerations when applying automatic bug triage to industrial projects? This study demonstrates the value of further research by incorporating component prediction and extracting information from images.

This paper applies a novel triage approach to industrial projects, examines the differences in industrial projects compared with existing open-source project-based research, draws implications, and presents a foundation for additional related research. The contributions of this paper are as follows:

- We propose a novel approach, component-based bug triage for industrial projects, examining the results.
- We analyze the characteristics of industrial projects compared with open-source projects in the context of automated bug triage.
- We exploit the related artifacts, such as unified modeling language (UML) images, frequently found in industrial projects and their effectiveness on automated bug triage.

The findings are significant for industrial projects. First, this study demonstrates that using automatic triage first and human triage afterward can maximize the results in industrial projects. Second, research on automatic bug triage based on industrial projects is limited. However, this study suggests a practical method for industrial project research. Third, differences from open-source projects that should be

considered for applying automatic bug triage to industrial projects are presented to facilitate future research. These findings significantly contribute to expanding automatic bug triage for industrial projects where 1) global development is occurring, 2) organizational changes occur frequently, and 3) a dedicated human triager is assigned.

The rest of the paper is organized as follows. Section II summarizes the characteristics of bug handling in industrial projects, and Section III describes the related work, including automatic bug triage and industrial projects. Next, Section IV details the hybrid method for this research, and Section V presents the background information of the experimental environment. Section VI explains the results, and Section VII summarizes the discussion items. Finally, Section VIII presents the conclusions.

II. BUG HANDLING IN INDUSTRIAL PROJECTS

Several characteristics of industrial projects are briefly provided in Section I, and this section explains further information as the basis for this research. There are three primary differences between industrial and open-source projects regarding bug handling.

First, from a timeline perspective, industrial projects often have tight schedules, meaning the duration for fixing bugs is limited. In particular, schedule management is highly stringent for software projects integrated with embedded hardware, as delays in software delivery can cause serious problems due to the tightly managed hardware supply chains. Therefore, even when significant software design changes are necessary, workarounds with minimal changes are sometimes chosen to meet deadlines, and resolving low-priority bugs is often postponed.

Table 1 presents the bug processing duration of the industrial projects in this study. Table 4 details the dataset statistics. Project A takes 18.28 days, the shortest in the dataset, whereas the other two projects take 20.47 days and 27.63 days, respectively. However, Project B takes 54.18 days, three times longer than Project A. This discrepancy is because Project B requires approval from an external certification agency, a requirement that does not apply to Project A.

Consequently, it took significant time to receive final confirmation even after the bug fix was ready. For fixes based on open-source platforms, such as Linux or Android, it took a long time for open-source changes to be officially integrated and included in the release, followed by additional time for recertification. Another reason is that a separate software update schedule is established for low-priority bugs, and bug fixes are distributed later, separate from the software release schedule aligned with the hardware schedule, resulting in a longer resolution period.

Table 2 compares the first action days of open-source projects. Although it is not the bug fixing duration but just the first action, a similar number of days as in Table 1 is listed for industrial projects.

TABLE 1. Average bug processing duration in ascending order of processing day (Source: industrial projects in this study).

Project	Processing days
A	18.28
D	20.47
C	27.63
B	54.18

TABLE 2. Bug processing first action.

Project	First action days
Eclipse	16.7
Mozilla	26.1

Note: open-source projects [5]

Another aspect of this tight timeline characteristic also affects the direction of bug fixing. For instance, the first assignee faces significant pressure to resolve the bug immediately, which may lead the assignee to prioritize fixes that can be implemented independently rather than considering a collaborative approach with other developers. Consequently, the assignee may become the sole assignee responsible for similar bugs in the future, regardless of whether this fix is the most effective solution. These kinds of practical difficulties occur due to the tight timeline.

Second, from a quality perspective, the quality of the bug report is essential to fixing bugs on time. Bettenburg et al. [17] reported that a good bug report includes three critical pieces of information: steps to reproduce, stack traces, and test cases. Bug report quality is generally higher in industrial projects than in open-source projects because professional testers with strict definitions of mandatory information and continuous maintenance in the company control the quality of bug reports. Industrial project testers write bug reports with a rich background, communicating directly with developers and applying their deep understanding of the specialized verification environment.

In contrast, in open-source projects, novices or careless individuals often report bugs without providing sufficient information for resolution. For instance, they might only provide a simple bug description without log files or screen captures. However, an internal escalation process is in place in industrial projects to immediately modify and update bug reports, ensuring consistently high-quality bug reports. Although Bettenburg et al. [18] reported that duplicate bug reports can help resolve problems, they can also consume developers' workloads. The duplicate bug check process is in place in industrial projects, so testers can revise previous bug reports with further descriptions.

In addition, as mentioned in Section I, the amount of information from bug reports is relatively minimal compared with open-source projects, posing a challenge for implementing an efficient machine learning model. Therefore, overcoming the lack of information would enhance the performance of automatic bug triage.

Third, from an organizational perspective, industrial projects, especially large-scale projects, must be managed

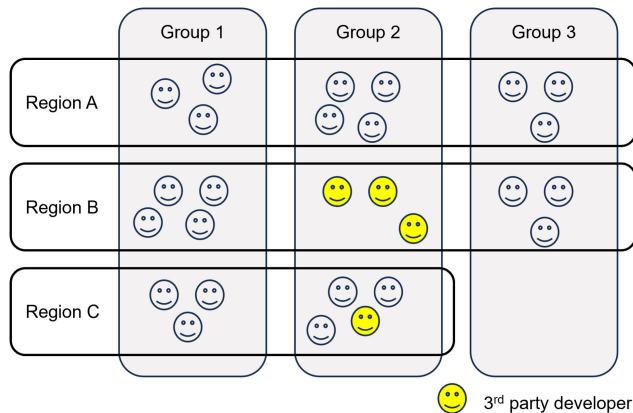


FIGURE 1. Typical group structure for industrial projects.

by a professional pool of developers and testers with high domain expertise. However, the members are sometimes changed for personal or company reasons. Organizational changes typically occur annually, and setting developers' names is challenging, especially for third-party companies and overseas regions. Another reason for organizational change is balancing the development load between groups.

Large-scale software project development encompasses hundreds of people. Third-party developers may participate, and multinational development may be conducted depending on the situation. In this environment, changes in the developer list between the training and application stages due to resignations or work transfers caused by organizational restructuring can significantly reduce the accuracy of bug triage. This differences between stages directly affects the automatic bug triage performance because the model was set based on the current developer assignment.

Fig. 1 presents a typical industrial group structure, including third-party developers and multinational regions. Many multinational software development teams are structured similarly, optimizing the overall development costs. Group 1 comprises three regional developers, whereas Group 3 comprises two regional developers. Group 2 includes third-party developers from Regions B and C. In this situation, it is beneficial to triage from the group's viewpoint first and then triage developers inside the group to increase accuracy. For some projects, this group is formed as an agile scrum, typically comprising four to 18 developers [13]. Third-party development can be categorized into two areas: 1) specific intellectual property oriented to their expertise and 2) engineering service focusing on development for designed components.

Third-party developers are essential to large-scale projects, but their specific characteristics must be considered. Third-party developers focusing on a specific technology have limitations in managing intercomponent resolutions for intellectual property reasons and resolving the other components instead of modifying them to maintain their original design. This situation introduces assignment uncertainty. Second, company contracts with a fixed duration do not allow

the developers to engage for the whole project timeline, introducing instability in third-party development.

This section highlights three primary differences between industrial and open-source projects regarding bug handling: timeline, bug report quality, and organizational structure. Industrial projects often operate under tight timelines, necessitating minimal changes and leading to delayed resolutions for low-priority bugs. The quality of bug reports in industrial projects is often higher due to professional testers and strict internal processes. Last, organizational changes and third-party developers pose challenges for consistent and effective bug triage in industrial projects. Addressing these differences is crucial to improving automatic bug triage performance in industrial contexts.

- The tight development timeline triggers the necessity of automatic bug triage.
- The quality of bug reports is well maintained; thus, a high degree of accuracy in automated bug triage can be anticipated.
- Organizational changes are inevitable in business environments; therefore, the triage approach must be capable of adapting to these changes.

III. RELATED WORK

This section presents previous studies on bug reports, automatic bug triage, optical character recognition (OCR), BERT, RoBERTa, and industrial projects. Several studies have addressed bug handling on industrial projects, and we investigate and compare them with this research.

A. BUG REPORT

Bug reports are a critical factor affecting software productivity. They consist of an *ID* identifying the item, *title* summarizing the bug, *description* with detailed information, *assignee* specifying the developer in charge, *priority* identifying the urgency, and *status*. Additionally, a *comment* records the discussion history, and the *reported date*, *last modified date*, and *external links* can also be a part of the bug report. The *component* is widely used to identify the bug report category and is usually selected by the bug reporter from a predefined component list (e.g., Bluetooth and life cycle).

Bettenburg et al. [17] introduced three conditions for a good bug report: steps to reproduce, stack traces, and test cases. Bettenburg et al. [18] also studied duplicated bug reports, which are treated as harmful, but they revealed that these can help resolve a bug because they provide further information.

Lamkanfi et al. [19] aimed to predict the severity of a bug, which is critical to meet the given timeline. In addition, Huang and Ma [20] introduced the multiplex network to predict bug fixers: an tossing graph and an email communication graph. From a bug report viewpoint, there is no specific difference between open-source and industrial projects, but industrial projects are more often customized

to adapt to the project-specific context, such as the *customer field*.

B. AUTOMATIC BUG TRIAGE

Automatic bug triage has been studied for a long time, and new approaches are being attempted, as machine learning techniques have recently evolved dramatically. The benefits of automatic bug triage can be defined in two ways. First, it reduces the time wasted by tossing bugs between developers by accurately assigning the right developer. Second, automatically assigning bug reports to someone on the development team helps prevent the human triager from delaying the overall software development process.

Ahsan et al. [21] proposed an automatic software bug triage system based on latent semantic indexing and the support vector machine (SVM). Anvik and Murphy [22] introduced an SVM-based technique, and Jeong et al. [5] suggested a tossing graph-based approach. Mani et al. [8] applied a word-based recurrent neural network and presented differences compared to existing classifier-based studies. Zaidi et al. [23] used BERT, a sentence-based approach, which performs better than the word-based approach.

C. OPTICAL CHARACTER RECOGNITION

The OCR [24] technique extracts text from images and can be used in industrial projects when the absolute number of bug reports is insufficient to enrich the data.

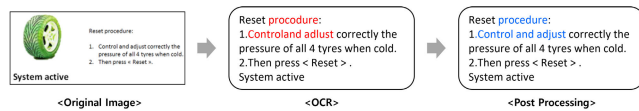


FIGURE 2. Optical character recognition with postprocessing.

Fig. 2 illustrates the typical OCR approach. The output of OCR can be extracted using already-released tools, but it is insufficient to use directly because errors can occur or corrections may be needed. Nguyen et al. [25] clarified the enhancing quality of OCR by studying its effects and defined the post-OCR processing problem, illustrating its typical pipeline, and reviewing state-of-the-art post-OCR processing approaches. Paddle OCR,⁴ an open-source project that can be accessed easily, has been widely used recently. Manipatruni et al. [26] recently leveraged artificial intelligence for paddle OCR-based text extraction.

D. BERT AND ROBERTA

Google introduced BERT [6], a pretrained language model based on text data using a transformer to understand the context. The BERT model is trained on unlabeled textual data and can be pretrained and fine-tuned. In addition, BERT has positional embedding, 12 transformer layers, 768 hidden states, and 12 attention heads. Although BERT uses large-scale data requiring considerable computing resources and a long learning time compared with existing models,

it performs better than previous models by fine-tuning pretrained language models in complex problems. Moreover, BERT processes text bidirectionally, considering the entire sentence, which is crucial in bug triage and requires understanding the context of the words.

Thus, follow-up studies have been conducted. For example, RoBERTa [7] is based on BERT but has more optimization. The RoBERTa model introduces dynamic masking during training, where the tokens selected for masking change at each epoch, in contrast to BERT's static masking approach. Further, RoBERTa also improves on BERT by training on a significantly larger dataset, providing more contextual information.

E. BUG TRIAGE STUDY IN INDUSTRIAL PROJECTS

Relatively few studies have assessed bug triage in industrial projects. This section analyzes notable examples from an industrial characteristic perspective.

Hu et al. [4] introduced BugFixer based on historical bug-fix information. They evaluated BugFixer on three open-source projects and two smaller industrial projects. However, the number of data was small: 1008 bug reports with 19 developers and 686 bug reports with 11 developers; hence, this study does not efficiently represent the characteristics of an industrial project.

Jonsson et al. [27] proposed a bug triager with a machine learning ensemble and measured performance on five industrial projects. They considered only the top-1 accuracy, and three reasons were presented. First, the automatic triage system would not assign a bug report to multiple teams. Second, the top-1 approach is a conservative choice. Third, they used Weka, which does not support the top-*k* approach. We support this approach because the industrial team expects the highest top-1 accuracy, as they treat the other cases as false triages.

Dedfík and Rossi. [28] compared open-source project results with those of industrial projects. They noted that industrial projects have fewer developers who fix fewer bugs than in open-source projects and suggested having the team triage in parallel with the individual developer triage. This suggestion also supports the motivation that team-level consideration is critical for industrial projects, especially long life-cycle projects with a high probability of changing developers as time passes compared with the initial project stages.

Lee et al. [10] applied convolutional neural networks (CNNs) to industrial projects based on bug titles and descriptions, which performed better than open-source projects. They observed that combining the CNN and human triagers achieved the best performance in reducing triage overhead. The characteristics of industrial project bugs are organized into three categories: bug report quality, size of the developer pool, and data imbalance. These characteristics remain valid and meaningful.

Additionally, organizational characteristics must be addressed considering the latest software development trends.

⁴<https://pypi.org/project/paddleocr/>

As software complexity and size increase, functional matrix organizations expand alongside the agile framework, making it challenging to increase expertise through traditional project-based organizations. Another organizational aspect to consider is global development, which has recently spread to enhance regional expertise and competitiveness in development costs. This approach allows developers on a global team to address similar bugs, regardless of location.

Sarkar et al. [29] applied recent bug triage research for nine large projects at Ericsson. Although their results demonstrated accuracy comparable to other large-scale bug triaging, it was insufficient for regular use at Ericsson. Thus, they introduced high-confidence prediction, which still requires manual triage effort, proving that using an automatic bug triage system alone is tough, but human triaging must be collaborative, as explained in Section VI-A.

Zhang [30] addressed the characteristics of industrial projects. First, internal developer transfer happens frequently. Second, the team is responsible for a software component rather than a specific developer. These findings are aligned with the observations in the current study.

Aktas and Yilmaz [31] presented another industrial case from a bank that receives an average of 350 bug reports daily from the field. The identified items must be considered when applying automatic triage to an industrial project. First, designing or changing the assignment process is necessary. Second, the accuracy of the triage does not have to be higher than that of the manual triage. Third, additional functionalities, such as detecting deterioration and creating human-readable explanations, are required. These findings are critical in an automatic bug triage system because most stakeholders do not want just a bug system but also a holistic approach.

Recently, studies have aimed to apply automated bug triage to industrial projects (Table 3). However, some studies have focused on team triage rather than individual developer triage. Even applying developer triage relies on word-based methods rather than context-aware pretrained language models. Among these studies, Zhang [30] suggested that a component-based approach benefits industrial projects, and they examined component prediction. We also advocate a component-based approach and propose a novel method incorporating component information for developer triage.

IV. METHOD

As Section III-B described, automatic bug triage is crucial to accurately identifying the responsible developer and facilitating an immediate analysis and reassignment among developers, preventing delays in bug activity until triage is complete. Even if a bug is incorrectly assigned, the developer who received it can analyze and reassign the bug to a more appropriate developer. Without automatic bug triage, a human triager must wait for each bug to be logged (even if occurring during off hours), manually analyze the bug description, and then triage it. Until this process is completed, no action can

be taken regarding the bug. Hu et al. [4] expressed a similar observation.

This activity is significant in industrial projects, as multinational development is becoming common to reduce software development and verification costs. For example, if a tester on one side of the world registers a bug and a human triager on the opposite side must assign it, no action would be taken on the bug for half a day. This advantage of minimizing delay is linked to the first characteristic of industrial projects described in Section II and contributes to a tight development schedule.

The proposed automatic bug triage approach is based on a pretrained language model, similar to the research by Wang and Lee [9]. The proposed approach applies a pretrained language model to triage developer and matches the result with a developer list for a specified component of the bug report. As explained in Section II. This research differs from the existing studies in two significant aspects. First, developer allocation is accomplished by considering the component. Second, as developers often change in industrial projects, triage is performed by considering the developers' change history.

As described in Section II, the team is organized by functional units or agile scrums, and developers usually resolve similar bugs in these functional units. Thus, achieving precise triage on the first attempt is challenging. The third characteristic identified in Section II, regarding third-party development, supports this difficulty, as developers commonly change during the development cycle. Given this context, a practical approach is to narrow down using the component information and allow the scrum team to handle the proper assignment according to the situation in the agile scrum.

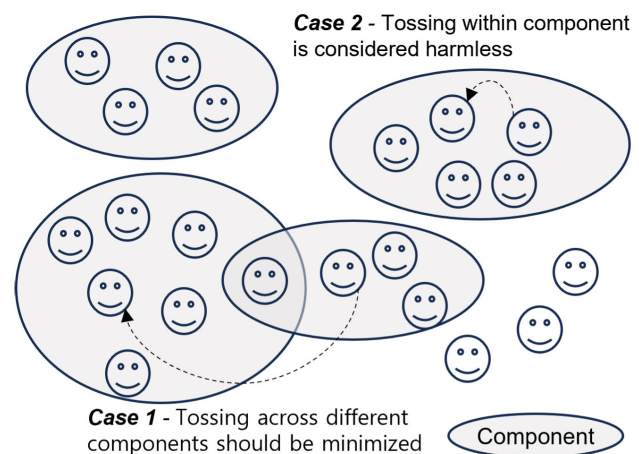
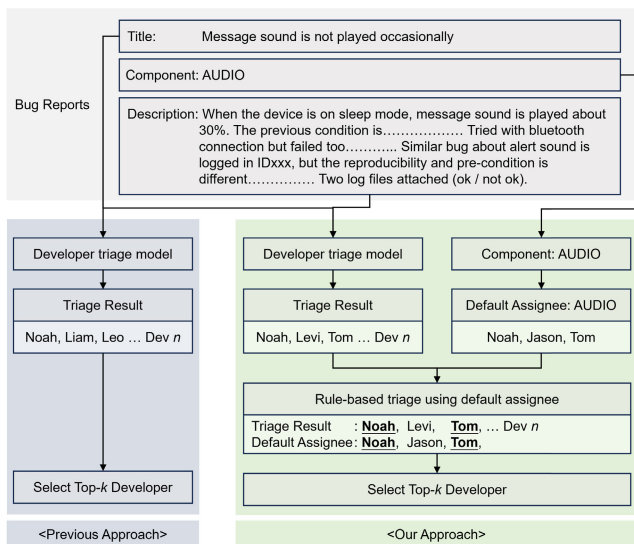


FIGURE 3. Reassignment (tossing) differentiation in the context of the component boundary.

As described in Fig. 3, developers are assigned to each component; in some cases, some are responsible for multiple components. Assuming that the scrum is organized on a component basis, reassigning (tossing) bugs between

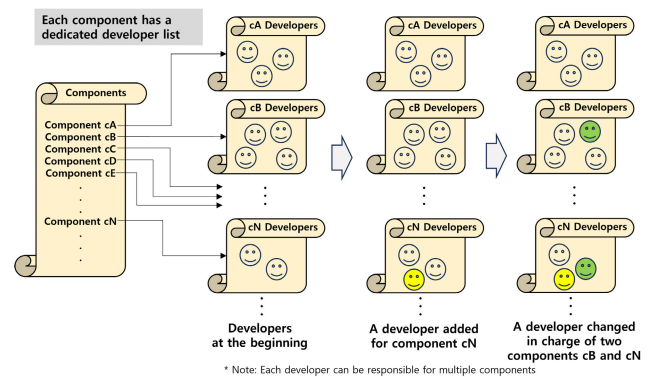
TABLE 3. Summary of bug triage study in industrial projects available in literature.**Notes:** (M: month). # Labels refer to the number of classes (e.g., developer, team, or component).

Paper	Industry	# Bugs	# Labels	Duration	Triage	Approach
Hu <i>et al.</i> , 2014 [4]	software	1008	19	7M	Developer	Apply component information Similarity DCB (Dev.-Comp-Bug) network
	software	686	11	4M	Developer	
Jonsson <i>et al.</i> , 2016 [27]	automation	15,113	67	140M	Team	Ensemble learner Stacked Generalization (SG)
	telecom 1	>9,000	28	>60M	Team	
	telecom 2	>8,000	36	>60M	Team	
	telecom 3	>3,000	17	>60M	Team	
	telecom 4	>10,000	64	>60M	Team	
Dedik <i>et al.</i> , 2016 [28]	software	2926	110	35M	Developer	SVM + TF-IDF performs the best
Lee <i>et al.</i> , 2017 [10]	electronics 1	1,257	65	17M	Developer	Apply CNN (Convolutional Neural Network)
	electronics 2	6,256	165	17M	Developer	
	electronics 3	14,583	82	27M	Developer	
	electronics 4	3,152	22	22M	Developer	
Sarkar <i>et al.</i> , 2019 [29]	telecom	11,570 (9 products)	43	24M	Team	Triage combined with the confidence level
Zhang <i>et al.</i> , 2020 [30]	software 1	71,829	199	144M	Component	Apply DNN (Deep Neural Network) introducing the component approach is beneficial
	software 2	30,662	487	60M	Developer	
Aktas <i>et al.</i> , 2020 [31]	bank	86,363	69	13M	Team	Provides considerations for actual deployment

**FIGURE 4.** Overview of the previous and proposed approaches.

components, as in Case 1, decreases the triage accuracy. However, if reassignment occurs within the same component, as in Case 2, it is considered a normal activity as load balancing is conducted on a scrum epic basis. Guo et al. [32] categorized five reasons for bug tossing: finding the root cause, determining ownership, poor bug report quality, difficulty determining the proper fix, and workload balancing. These reasons are also valid for industrial projects, but we assume that tossing within a team or scrum is not harmful because it is intentional.

Based on these characteristics, this study exploits the developer and component information to triage the proper developer accurately. This approach achieves higher accuracy due to the dataset quality and performance of pretrained language models. Furthermore, based on component information, the proposed rule-based technique narrows the scope of developer triage compared with targeting all developers. Additionally, this component-based method allows for bug

**FIGURE 5.** Component-based management of developer lists.

tossing between developers in the same group, making the process more manageable during development.

Fig. 4 illustrates the difference between the previous and proposed approaches. The bug report provides the title, description, and component information. The typical machine learning-based approach employs the title and description [8], which is the same for the proposed approach. As described in Section III-D, the BERT-based model was selected for developer triage, which is suitable for bug triage because it is a context-based approach. Zaidi et al. [23] summarized the benefit of BERT compared with the previous deep learning-based models, CNN and Recurrent Neural Network (RNN). The result of the developer triage model is compared with the default assignee list of the specified component. This comparison narrows the scope of candidate developers, improving the overall accuracy.

Fig. 5 illustrates the component-based developer list. The component list on the left identifies the targeting component and has developer lists connected to each component. When a developer is assigned to a specific component cN , the new developer is added to the cN developer list (highlighted in yellow). If the developers for specific components cB and cN change, the corresponding entries in the cB and cN developer

TABLE 4. Dataset statistics. Notes: (M: month). # Labels refers to the number of classes, indicating the number of developer classes for all projects except for project E, which refers to the number of component classes.

Project	# Bugs	# Labels	Duration
<i>A</i>	16,374	332	44M
<i>A'</i>	20,489	395	58M
<i>B</i>	11,158	418	45M
<i>C</i>	6,627	195	28M
<i>D</i>	3,104	107	18M
<i>E</i>	34,419	117	63M
Google Chromium	263,936	1,032	96M
Mozilla Core	186,173	591	217M
Mozilla Firefox	138,093	170	204M

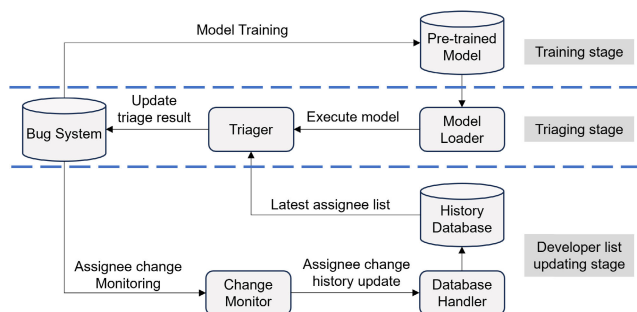


FIGURE 6. Overall procedure.

lists are replaced (highlighted in green). A developer may be responsible for multiple components; thus, multiple updates may occur. The proposed approach employs component information to narrow the candidate developer list. For example, if the model selects Developers A, C, and E as candidate assignees for component *cB*, the proposed algorithm verifies whether Developers A, C, and E are part of the *cB* developer list. If Developer C is not included in the *cB* developer list, the algorithm selects Developers A and E as candidate developers for component *cB*.

Fig. 6 depicts the overall procedure. During the training stage (top), the RoBERTa pretrained language model is fine-tuned to construct a prediction model. During the developer list update stage (bottom), the bug system is monitored, and the developer list described in Fig. 5 is updated when developer assignments change. At the triaging stage (middle), actual triage is performed based on the outcomes of these two stages. Finally, the results are reflected in the bug system.

V. EXPERIMENTAL ENVIRONMENT

A. DATASET

We used datasets from industrial projects for the experiment, as listed in Table 4. Project A' has a project duration of almost five years and the most bug reports, totaling 20,489. We extended Project A into A' because we gathered additional data after the first evaluation of Project A. Project B also has a project duration of almost four years and includes 11,158 bug reports. Project C has a duration of 2.5 years and includes 6,627 bug reports. Project D was also selected, with 3,104 bug reports, to check whether the information

gathered from images enhances performance. Project E was used to evaluate the component prediction and was trained with 34,419 bug reports.

The number of labels for Projects A, A', B, C, D, and E are 332, 395, 418, 195, 107, and 117, respectively. Despite having the most bug reports, Project A' has fewer assignees than Project B. Therefore, we selected Projects A and A' as the primary datasets to examine triage performance. The number of labels for Project E is the number of components, whereas the others are the number of bug reports. All obtained data underwent the same preprocessing steps, such as HTML removal, code snippet removal, and corpus-based stopword processing. Three well-known open-source projects were used as a comparison group for the industrial datasets: Google Chromium, Mozilla Core, and Mozilla Firefox, from Mani et al. [8].

B. EXPERIMENTAL SETTINGS

Consistent experimental settings were applied across all datasets for fair comparisons and reproducibility. All datasets, including Projects A, A', B, C, D, and E as well as Google Chromium, Mozilla Core, and Mozilla Firefox, were divided into training, testing and validation sets with a training/testing/validation ratio of 8:1:1. The splitting process was designed to maintain a representative label distribution in the sets. The training set was applied to optimize the model weights, and the testing set served as an evaluation set to assess model performance on unseen data.

The hyperparameters were carefully configured for each of the parameters. For Projects A and A', the learning rate was set to 0.001, whereas Projects B and C employed a learning rate of 0.005. Mozilla Firefox and Google Chromium required a slightly higher learning rate of 0.01. Project D, with its smaller dataset size, applied a learning rate of 0.002, whereas Project E was configured with a learning rate of 0.003 to handle its unique component-based classification task. The batch size was set to 64 for Projects A, A', and B, whereas Projects C and D used a batch size of 128. Mozilla Firefox, Mozilla Core, and Google Chromium, due to their larger dataset sizes, employed a batch size of 32. All models were trained for 30 epochs, with early stopping applied if the validation loss did not improve for more than five consecutive epochs. The AdamW [33] optimizer was applied for all datasets, with the default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$).

The experiments were conducted in a standardized environment to ensure consistency across all datasets. The hardware included an Nvidia RTX 3090 GPU with 24 GB of RAM, and the software environment consisted of PyTorch 1.10.0 and Python 3.8.

C. EVALUATION METHOD

Top- k is a machine learning and information retrieval technique that selects the top- k items from a list based on criteria or scores. *Top* refers to selecting the highest-ranking

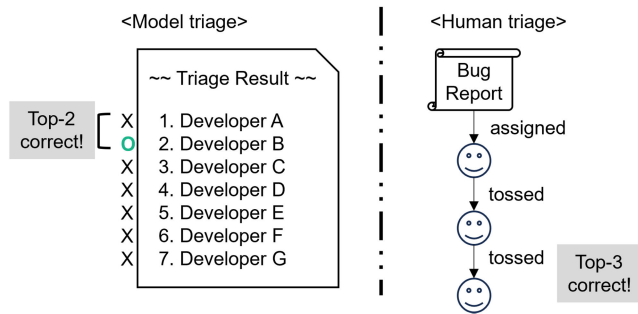


FIGURE 7. Triage comparison.

TABLE 5. Prediction model comparison for team triage (RQ 1).

Project	Model	Top-1	Top-3	Top-5	Top-10
A	BERT	47%	64%	70%	78%
	RoBERTa	53%	70%	76%	84%
B	BERT	49%	62%	67%	74%
	RoBERTa	51%	66%	71%	77%
C	BERT	61%	71%	71%	85%
	RoBERTa	66%	76%	82%	88%

items, and k denotes the number of items to retrieve, usually a fixed, positive integer (e.g., 1, 5, or 10). In machine learning tasks, top- k is widely applied in recommendation systems, ranking algorithms, and neural network predictions to filter and deliver the most relevant k items of a more extensive set.

This section presents a method for comparative evaluation of the human and automated triage approaches developed in this study. For model triage, the resulting list is generated at once, and the top- k position is determined based on its list rank. For human triage, the evaluation is based on the number of tossings. Fig. 7 presents an example where the model triage correctly identifies the bug in the top-2 positions, as the second item on the list is correct, which is Developer B. If Developer C is correct, it is top-3; if Developer A is correct, it is top-1. In contrast, the human triage of Fig. 7 is considered top-3 correct because it took two tossings to reach the correct assignment. Similarly, if no tossing occurs, it is considered top-1; if tossing does occur, it is considered top-2.

VI. RESULTS

According to the characteristics of industrial projects as described in Section II, this study addresses the following research questions:

- **RQ 1:** Is the assignee prediction using the proposed method better than the human triager in industrial projects?
- **RQ 2:** How is the bug triage performance of industrial and open-source projects differentiated?
- **RQ 3:** What are the special considerations when applying automatic bug triage to industrial projects?

A. RESEARCH QUESTION 1

As described in Section IV, the pretrained language models BERT and RoBERTa were applied. Table 5 presents the top-1, 2, 5, and 10 scores for the BERT and RoBERTa models. In all cases, RoBERTa performs better than BERT, indicating

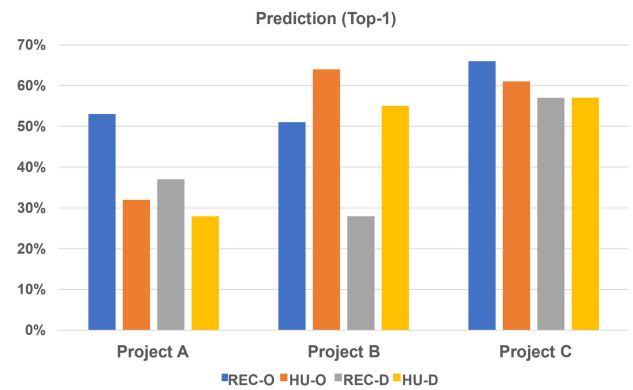


FIGURE 8. Prediction comparison with a human triager (RQ 1).

that RoBERTa, which optimizes BERT, is also suitable for this research. Thus, all subsequent experiments in this paper were conducted based on RoBERTa.

Considering the characteristics of industrial projects, the top-1 score is the most important. Project A's top-1 score is the lowest at 47%, whereas Project C's top-1 score is the highest at 61%. Table 5 presents team-level predictions to compare the performance of BERT and RoBERTa, which is why the scores are relatively high. Developers in industrial projects frequently change; hence, we aimed to triage at the team level, providing a higher level of abstraction.

As described in Section II, the triage method is applied to the organization and individual developers to compare the results, considering the frequent changes in developers typical of industrial projects. Triageing at the organizational level makes the assignment unit coarser than the developer level, and reassignment within the organization becomes less costly, even if developers change.

Fig. 8 presents 12 results, each representing a combination of organization recommendation vs. individual recommendation and model recommendation vs. human recommendation for each of the three industrial projects. The records of the model triage for the organization and developer are REC-O and REC-D, respectively. The records of human triage for the organization and developer are denoted by HU-O and HU-D, respectively. These data are for the top-1 recommendation, which is considered a significant performance measure in industrial projects.

Overall, model performance is better in Projects A and C, whereas the performance of the human triager is better in Project B. Quantitatively, Project B has the most assignees with 418 developers, increasing the difficulty for the model. From a qualitative perspective, the proficiency of the human triager in Project B is high because it has a previous generation, enriching the triager's expertise.

In the case of Project C, the overall performance of the model and human triager is high, as the number of assignees and bug reports is relatively small (195 and 6,627, respectively), and the project complexity is lower

TABLE 6. Assignee prediction on project A' (RQ 1).

Top- <i>k</i>	Model	Human
Top-1	34% (6,694/20,489)	27.8% (5,676/20,489)
Top-3	53.8% (11,049/20,489)	73% (15,011/20,489)
Top-5	61.3% (12,581/20,489)	88.7% (18,183/20,489)

than in Projects A and B. Naturally, organizational-level triage performs better than assignee-level triage, but it is challenging to use practically in an industrial environment, as each organization typically consists of more than a few dozen people. Therefore, this research narrowed the scope by applying component-level triage.

Assignee prediction for Project A' is listed in Table 6 from model and human perspectives, yielding notable results. The top-1 score of the model is 34%, which is better than that for human performance at 27.8%, a 6.2%p improvement. According to Table 6, 1,018 (6,694-5,676) bugs were triaged more accurately than those by the human triager. This performance improvement enhances accuracy and reduces the need for re-triaging bugs, which can be a significant cost in industrial projects.

Lee et al. [10] investigated the cost of re-triaging bugs and presented a range of estimates for the humans/day effort depending on the project size and complexity. In their analysis, 0.2 humans/day were identified as the lowest cost per bug for re-triaging, which this study adopts for a conservative estimate in the calculations. Based on this assumption, the proposed approach results in a cost savings of approximately 200 humans/day throughout the project duration, underscoring its efficiency in reducing the re-triaging workload compared with human triaging. This result indicates that the automatic triager performs better than the human triager for top-1 predictions. However, as the K-index increases, the human triager performs better. For top-3, the model achieves 53.8% performance compared with the human triager at 73%, a difference of about 20%. For top-5 predictions, the model reaches 61.3% compared with the human results at 88.7%, a more than 25%.

This discrepancy arises because a human with high domain knowledge based on experience, as described in Section II, has higher accuracy than an automatic triager. Additionally, the model predicts top-1, 3, and 5 at once, whereas the human makes sequential predictions, performing up to five top-5 predictions, leading to better performance by the human triager. In industrial projects, the development timeline can be shortened depending on how quickly the first action is taken. Therefore, it is practical for industrial projects to combine the initial triage with a model with further reassignment by a human triager.

B. RESEARCH QUESTION 2

The results of a quantitative comparison between an open-source project (Mozilla Core) and an industrial project (Project A) using the same model are presented in Table 7. Project A has 16,324 bug reports, 169,799 fewer than Mozilla

TABLE 7. Comparison with open-source project (RQ 2).

Project	# Bugs	# Dev.	Top-1	Top-3	Top-5	Top-10
A	16,324	332	37%	51%	58%	66%
Mozilla Core	186,173	302	17%	30%	36%	47%
Difference	-169,799	+30	+20%p	+20%p	+22%p	+19%p

TABLE 8. Open-source prediction (RQ 2).

Dataset	Top-1	Top-3	Top-5	Top-10
Google Chromium	20%	34%	41%	50%
Mozilla Core	17%	30%	36%	47%
Mozilla Firefox	14.3%	27%	33%	41.4%

Core at 186,173. The number of assignees for Project A is 332, compared to 302 for Mozilla Core, meaning Project A has more assignees despite having significantly fewer bug reports, making Project A more challenging. For instance, in Mozilla Core, each assignee is responsible for an average of 616.5 bugs (186,173/302), whereas in Project A, each assignee is responsible for only 49.1 bugs (16,324/332), almost 12 times fewer. This difference in the responsibility for average bugs indicates that Project A requires more responsibility for bugs, demanding a more precise model.

Despite this higher difficulty level, the overall performance is improved by about 20%p for top-1, 3, 5, and 10 predictions at 20%p, 21%p, 22%p, and 20%p, respectively. This improvement is attributed to the better quality of bug reports in industrial projects, as explained in the second characteristic of industrial projects in Section II.

Table 8 presents the prediction results of three open-source projects using the same model from previous research [34]. Despite having top-10 accuracy, the accuracy for Google Chromium (50%), Mozilla Core (47%), and Mozilla Firefox (41.4%) is lower than the top-3 (53.8%) and top-5 (61.3%) accuracy for Project A' presented in Table 6. Three reasons can explain this. First, we run developer triage model and then assign a developer for that specific component (Fig. 4), enhancing triage performance. The second reason is the higher quality of bug reports, as explained in the second characteristic in Section II. The third reason is the narrower scope of triage due to the tight project timeline (Section II).

C. RESEARCH QUESTION 3

Proper methods must be applied according to the situation or variation of the project. As described in Section II, industrial projects have their own rules and customs. For instance, in a specific project, each group of developers is aligned with a component. Notably, components change less frequently during development than developers. For example, the component change rate in Project E is only 0.41%. This minimal change suggests that it is natural to predict components precisely because developers are directly mapped to components.

Additionally, in projects with considerable image information, essential details, such as requirements, specifications, and UML diagrams, might be included in the images.

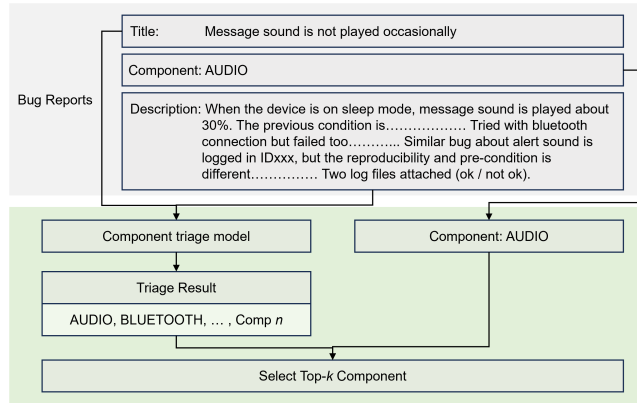


FIGURE 9. Component Prediction (RQ 3).

TABLE 9. Component prediction comparison (RQ 3).

Project	Top-1	Top-3	Top-5	Top-10
<i>E</i>	74%	87%	91%	93%
Bugzilla Firefox	N/A	N/A	40%	43%
Eclipse Platform	N/A	N/A	90%	93%
Eclipse Foundation	N/A	N/A	81%	88%

Thus, extracting valuable information from images in these cases must be considered. These cases require specific customization to evaluate the suitability on a project-by-project basis.

We applied the component prediction technique (Fig. 9) separately to Project E assigning developers based on component information, which was trained on a dataset of 34,419 entries collected from July 2017 to October 2023, to confirm the effectiveness of this method. Table 9 indicates that the top-10 performance is 93%, compared to the open-source project results from Wang and Lee [9]. The results demonstrate performance similar to the Eclipse platform, proving that component prediction also works well in industrial projects.

Although we tested only 122 bug reports, these were entirely new data not involved in the training process, verifying whether the model trained on past data can be applied to current bugs. Therefore, we used bugs that arose after the model training was completed for testing.

Industrial projects often involve images, such as screenshots, user scenarios, and flow diagrams. Critical bug reports typically come from end users and include screenshots of rarely reproducible symptoms. The quality assurance team also commonly studies abnormal behavior by comparing user scenario diagrams with actual behavior. These bug reports may also include flow diagrams to explain reproducible procedures. Thus, these image files assist developers in resolving bug reports.

As described in Table 4, the total number of bugs in industrial projects is relatively small compared with open-source projects. To address this, we enriched the dataset by extracting information from images. Table 10 presents the

TABLE 10. Applying image information for project D (RQ 3).

Model	Top-1	Top-5
Without Image Info	70.3% (2,183/3,104)	87% (2,701/3,104)
With Image Info	73.7% (2,289/3,104)	88.7% (2,754/3,104)

experiment results for Project D. Although the dataset is not large enough to train the model, it demonstrates a 3.4%p improvement in top-1 performance and 1.7%p improvement in top-5 performance. These results are sufficiently meaningful though image information was added to only 300 of the 3104 data points (9.66%).

As expressed in Fig. 2, basic text was extracted using paddle OCR, but there were challenges in accuracy and efficiency when using this information directly; hence, we tailored the OCR by adding filtering and postprocessing steps to improve accuracy. Some images contain multiple languages, making the language model unable to recognize them effectively; thus, languages other than English were filtered out, and if the number of letters was too few to help the model learn, less than 100 characters were removed. A deep learning-based space removal model was applied for postprocessing, and spell correction was applied when spelling errors occurred. The result would not have improved without these enhancements.

VII. DISCUSSION

Automatic bug triage has not been widely adopted in industrial projects; hence, some have doubts about its effectiveness compared with human triagers and have concerns regarding the effects of an incorrect triage. However, given the tight development cycles and multinational collaboration, automatic bug triage is critical for enhancing software productivity.

Research has focused on identifying the proper developers for bug triage. However, considering the recent trend toward functional-based organizational structures and the varying workloads of developers, a more practical approach would be using intermediate techniques, such as component-based triage, as proposed in this paper. This research examines the feasibility of such an approach.

The proposed approach improved results by 6.2%p for the top-1 score, which is meaningful (Table 6). However, the top-3 and top-5 results reveal the opposite, which indicates that the best result only occurs by combining the model and human triage. In contrast, the model-only approach is almost impossible because human interpretation is sometimes inevitable.

This study initially attempted to apply logic directly to the ALM system. However, due to the initial lack of completeness of the model and concerns about accuracy, comments were added to each bug ticket rather than directly changing the assignee. This situation demonstrates that achieving a consensus with stakeholders involved in bug handling is crucial to applying successful automatic bug triage to industrial projects.

There are also nontechnical challenges in expanding automatic bug triage to industrial projects. First, new techniques are resisted due to a tendency to maintain existing, well-established methods. The unfamiliarity with the new technique causes this resistance. It is effective for engineers who thoroughly understand the technique to participate in the project and directly support the relevant stakeholders.

Second, some may have concerns regarding the imperfection of innovative technologies after application. For example, if triage results are inaccurate, some may advocate reverting to the previous method. In such cases, persuading stakeholders that human triagers can also make similar errors is crucial. Identifying the cause of the inaccuracies and improving the model is also essential. Moreover, considering explainable artificial intelligence, which has become an important topic, is also worthwhile.

Resolving these concerns in a tight period is challenging, and the benefits of automatic bug triage can be maximized once it is fully integrated into the company. Establishing a dedicated organization with a long-term mission for these activities is essential, and this organization should operate with a forward-looking vision.

A. THREATS TO VALIDITY

Several threats to establishing the validity of the proposed approach exist:

- 1) This study examined the proposed approach in detail from the perspective of industrial projects. However, this study is based on a single company; thus, it is difficult to generalize the results to different industries, organizational structures, and development methodologies. Companies commonly operate with a consistent team structure and development methodologies, making it challenging to validate results using data from other companies.
- 2) This work employed a dataset collected from a global company in the electronics industry. As summarized in Table 3, industrial project-based research targeting developer triage has used datasets with fewer than 7000 bug reports, except in two cases. However, Table 4 reveals that this study employed a dataset of over 10,000 bug reports in four cases. Validation with a more extensive and more varied industrial dataset is still needed for the scalability of this approach.
- 3) The source for the dataset is software-intensive projects from an electronics company. In general, the dataset from an open-source project is accessible to other researchers to examine for further research. However, most industrial data contain confidential information that could risk a company's business, so the data are not publicly accessible.
- 4) Automatic bug triage requires model training. However, training with domain-specific data produces outcomes biased toward that domain, making it

challenging to apply the model to different industrial projects.

B. FUTURE RESEARCH

Future research should address several aspects, including ensuring the generalizability of the approach across domains, enhancing the scalability to accommodate more extensive and complex projects, refining the method through iterative improvements and real-time feedback, and developing robust, standardized datasets explicitly tailored to industrial projects.

More testing is necessary to achieve generalizability and scalability. Testing with various industrial data can help determine whether the proposed approach is suitable across domains. Testing across multiple organizations could also help validate whether the component-based organizational approach is scalable. This result could facilitate determining whether a customized testing framework is necessary.

This work evaluated the proposed approach to assess its effectiveness and applicability. A real-time feedback loop for further refinement would enhance the accuracy and reliability of this approach, helping to validate its suitability for future projects and ensuring it adapts effectively to varying conditions and requirements.

A robust dataset is crucial for a deep learning-based approach. Despite the inherent limitations of industrial projects, creating a benchmark dataset remains essential for a standardized approach. The volume of open-source data is far greater than that for industrial projects. Continuous research to increase the number of data is still needed. This research demonstrates the effectiveness of incorporating image data; however, domain dependencies exist, such as the frequency of image usage and text in images. Therefore, a more in-depth consideration and future research are required.

VIII. CONCLUSION

Most bug triage research focuses on open-source projects due to challenges accessing industrial projects and convincing their teams to adopt new triage technology. Based on this background, this research explored methods suitable for industrial projects. We categorized three characteristics of industrial projects. The first is the necessity for automatic bug triage due to tight development cycles compared with open-source projects. The second is the superior bug report quality compared with open-source projects. The third is the challenge of assigning bugs to developers due to frequent organizational and developer changes.

Based on these insights, we proposed a new triage approach at the team level that focuses on components. This approach yielded a 6%p improvement over human triage for top-1 results. However, human triagers demonstrated better performance in the top-3 and later results, highlighting the effectiveness of a combined model-human triage approach. Specifically, model triage effectively initiates bug analysis, whereas human triage ensures precise assignment following an in-depth bug report analysis. Moreover, considering industrial projects typically have fewer data than open-source

projects, we explored the potential to enhance bug triage performance by incorporating additional information, such as text extracted from images. This research presents the potential to integrate and advance the unique characteristics of industrial projects into traditional bug triage research, which has historically been centered on open-source projects.

REFERENCES

- [1] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proc. 4th Int. Workshop Mining Softw. Repositories*, May 2007, p. 1.
- [2] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proc. 2nd Int. Workshop Recommendation Syst. Softw. Eng.*, May 2010, pp. 52–56.
- [3] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. 28th Int. Conf. Softw. Eng.*, May 2006, pp. 361–370.
- [4] H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage based on historical bug-fix information," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng.*, Nov. 2014, pp. 122–132.
- [5] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, Aug. 2009, pp. 111–120.
- [6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
- [7] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.
- [8] S. Mani, A. Sankaran, and R. Aralikkatte, "DeepTriage: Exploring the effectiveness of deep learning for bug triaging," in *Proc. ACM India Joint Int. Conf. Data Sci. Manage. Data*, Jan. 2019, pp. 171–179.
- [9] D.-S. Wang and C.-G. Lee, "Automatic component prediction for issue reports using fine-tuned pretrained language models," *IEEE Access*, vol. 10, pp. 131456–131468, 2022.
- [10] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, Aug. 2017, pp. 926–931.
- [11] N. Pandey, D. K. Sanyal, A. Hudait, and A. Sen, "Automated classification of software issue reports using machine learning techniques: An empirical study," *Innov. Syst. Softw. Eng.*, vol. 13, no. 4, pp. 279–297, Dec. 2017.
- [12] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: An empirical study of commercial software projects," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 1042–1051.
- [13] M. Cohn, *Succeeding With Agile: Software Development Using Scrum*. London, U.K.: Pearson, 2010.
- [14] K. Schwaber and M. Beedle, *Agile Software Development With Scrum*. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.
- [15] D. Wang, H. Zhang, R. Liu, M. Lin, and W. Wu, "Predicting bugs' components via mining bug reports," *J. Softw.*, vol. 7, no. 5, pp. 1149–1154, Apr. 2012.
- [16] K. K. Sabor, A. Hamou-Lhadj, A. Trabelsi, and J. Hassine, "Predicting bug report fields using stack traces and categorical attributes," in *Proc. 29th Annu. Int. Conf. Comput. Sci. Softw. Eng.*, Jan. 2019, pp. 224–233.
- [17] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, Nov. 2008, pp. 308–318.
- [18] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful ... Really?" in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2008, pp. 337–345.
- [19] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Proc. 7th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, May 2010, pp. 1–10.
- [20] J. Huang and Y. Ma, "Predicting the fixer of software bugs via a collaborative multiplex network: Two case studies," in *Proc. 15th Int. Conf. Collaborative Comput., Netw., Appl. Worksharing*, London, U.K. Cham, Switzerland: Springer, Jan. 2019, pp. 469–488.
- [21] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine," in *Proc. 4th Int. Conf. Softw. Eng. Adv.*, Sep. 2009, pp. 216–221.
- [22] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–35, Aug. 2011.
- [23] S. F. A. Zaidi, H. Woo, and C.-G. Lee, "Toward an effective bug triage system using transformers to add new developers," *J. Sensors*, vol. 2022, no. 1, Apr. 2022, Art. no. 4347004.
- [24] S. Mori, C. Y. Suen, and K. Yamamoto, "Historical review of OCR research and development," *Proc. IEEE*, vol. 80, no. 7, pp. 1029–1058, Jul. 1992.
- [25] T. T. H. Nguyen, A. Jatowt, M. Coustaty, and A. Doucet, "Survey of post-OCR processing approaches," *ACM Comput. Surv.*, vol. 54, no. 6, pp. 1–37, Jul. 2022.
- [26] J. K. Manipatruni, R. G. Sree, R. Padakanti, S. Naroju, and B. K. Depuru, "Leveraging artificial intelligence for simplified invoice automation: Paddle OCR-based text extraction from invoices," *Int. J. Innov. Sci. Res. Technol.*, vol. 8, no. 9, pp. 1–6, Sep. 2023.
- [27] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Softw. Eng.*, vol. 21, no. 4, pp. 1533–1578, Aug. 2016.
- [28] V. Dedík and B. Rossi, "Automated bug triaging in an industrial context," in *Proc. 42th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2016, pp. 363–367.
- [29] A. Sarkar, P. C. Rigby, and B. Bartalos, "Improving bug triaging with high confidence predictions at Ericsson," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2019, pp. 81–91.
- [30] W. Zhang, "Efficient bug triage for industrial environments," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2020, pp. 727–735.
- [31] E. U. Aktas and C. Yilmaz, "Automated issue assignment: Results and insights from an industrial case," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 3544–3589, Sep. 2020.
- [32] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "'Not my bug!' and other reasons for software bug report reassignments," in *Proc. ACM 2011 Conf. Comput. Supported Cooperat. work*, 2011, pp. 395–404.
- [33] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017, *arXiv:1711.05101*.
- [34] D.-S. Wang, H. Seong, and C.-G. Lee, "Recommendation technique for bug fixers by fine-tuning language models," *J. KIISE*, vol. 49, no. 11, pp. 987–998, Nov. 2022.



HYUN-TAEK HONG received the M.S. degree in information engineering. He is currently pursuing the Ph.D. degree in computer engineering with Chung-Ang University. He is a certified LG software architect, having completed the software architect course with Carnegie Mellon University. He works as the Lead Architect at LG Electronics. He also works as a Board Member of the Vehicle Semiconductor and System Software Society, The Korea Society of Automotive Engineers (KSAE).

His current work at LG Electronics focuses on software defined vehicle and software platform to enhance vehicle flexibility through evolving software.



DAE-SUNG WANG was born in Jeonju, South Korea. He received the B.S. degree in software engineering from Jeonbuk National University, Jeonju, and the M.S. degree in computer science and engineering from Chung-Ang university, Seoul, South Korea, where he is currently pursuing the Ph.D. degree in computer science and engineering. His research interests include software engineering, requirement engineering, and natural language processing.



SE-JIN KIM was born in Gyeonggi-do, South Korea. He received the B.S. degree in computer science and engineering from Chung-Ang University, Seoul, South Korea, in 2023, where he is currently pursuing the M.S. degree in computer science and engineering. His research interests include software engineering, log generation, and natural language processing.



HO-HYUN PARK received the B.S. degree from Seoul National University, in 1987, and the M.S. and Ph.D. degrees in computer science and engineering from KAIST, in 1995 and 2001, respectively. From 1987 to 2003, he was a Principal Engineer at Samsung Electronics. He is currently working as a Professor of electrical and electronics engineering with Chung-Ang University. His research interests include big data, deep learning, machine vision, information security, and real-time and embedded systems.



HOON SUNG received the B.S. and M.S. degrees in computer engineering from Chung-Ang University, in 2019 and 2022, respectively, where he is currently pursuing Ph.D. degree with the Department of Computer Engineering. His research interests include software engineering, software maintenance, natural language processing, and machine learning.



CHANG-WON PARK received the Ph.D. degree in computer science. He is currently working as the Director of Software Governance Organization and a Research Fellow with LG Electronics. He also leads Software Defined Vehicle task including Software Platform at Vehicle Solution Company in LG Electronics. His research interests include software product line and governing to achieve sustainable software development and maintenance. He also works as a Board Member of Vehicle Semiconductor and System Software Society, The Korea Society of Automotive Engineers (KSAE).



CHAN-GUN LEE was born in Seoul, South Korea, in 1972. He received the B.S. degree in computer engineering from Chung-Ang University, Seoul, in 1996, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 1998, and the Ph.D. degree in computer science from The University of Texas at Austin, Austin, TX, USA, in 2005. From 2005 to 2007, he was a Senior Software Engineer at Intel, Hillsboro, OR, USA. Since 2007, he has been working as a Professor with the Department of Computer Science and Engineering, Chung-Ang University. He is the author of more than 30 articles and conference papers. His research interests include software engineering and real-time systems. He was a recipient of the Korea Foundation of Advanced Studies (KFAS) Fellowship, from 1999 to 2005.

...