A Review Based on Active Research Areas in Mining Software Bug Repositories: Limitations and Possible Future Trends

Fatima Waseem[1*], Farah Haneef[1], Muhammad Nouman Noor[2], Aisha Khalid[3], Hina Rashid[1], Qudsia Yousaf[1]
[1]Software Engineering Dept. CUST Islamabad, Pakistan
[2]AI and Data Science Dept. FAST National University, Pakistan
[3]Computer Science Dept. NUML, Islamabad, Pakistan
[*]**Corresponding author:** fatima.waseem@cust.edu.pk

Citation | Waseem. F, Haneef. F, Noor. M. N, Khalid. A, Rashid. H, Yousaf. Q, "A Review Based on Active Research Areas in Mining Software Bug Repositories: Limitations and Possible Future Trends", IJIST, Vol. 6 Issue. 3 pp 1249-1266, Aug 2024
Received | Aug 2, 2024 **Revised |** Aug 23, 2024 **Accepted |** Aug 24, 2024 **Published |** Aug 25, 2024.

Introduction/ Importance of Study: Bug repository mining is a crucial research area in software engineering, analyzing software change trends, defect prediction, and evolution. It involves developing methods and tools for mining repositories, and providing essential data for bug management.
Objective: The goal of this study is to analyze and synthesize recent trends in mining software bug repositories, providing valuable insights for future research and practical bug management.
Novelty statement: Our research contributes novel insights into mining software repository techniques and approaches employed in specific tasks such as bug localization, triaging, and prediction, along with their limitations and possible future trends.
Material and Method: This study presents a comprehensive survey that categorizes and synthesizes the current research within this field. This categorization is derived from an in-depth review of studies conducted over the past fifteen years, from 2010 to 2024. The survey is organized around three key dimensions: the test systems employed in bug repositories, the methodologies commonly used in this area of research, and the prevailing trends shaping the field.
Results and Discussion: Our results highlighted the significance of artificial intelligence and machine learning integration in bug repository mining; which has revolutionized the software development process by enhancing the classification, prediction, and vulnerability detection of bugs.
Concluding Remarks: This survey aims to provide a clear and detailed understanding of the evolution of bug repository mining, offering valuable insights for the ongoing advancement of software engineering.

Keywords: Mining Software Repositories, Bug Localization, Bug Classification, Bug Estimation, Bug Triaging

Introduction:

Software repository mining involves the extraction of valuable information from the historical records of software development and evolution, which are stored in software repositories. This practice supports various software engineering activities, including code analysis, bug detection, program comprehension, and architecture recovery. Over recent years, software repository mining has emerged as a prominent research field, with substantial efforts directed toward advancing the methodologies and tools used in this domain. Software repositories are extensive databases that store detailed information about source code changes, bug reports, and other specifications related to software projects. Among these, bug repositories are particularly crucial as they house comprehensive details about software bugs. These repositories play an integral role in the bug management process, encompassing activities such as bug localization, prediction, triaging, and analysis.

Researchers in software engineering have employed diverse techniques and approaches to explore and analyze critical information stored in bug repositories. This information is instrumental for software engineers and testers in the development and maintenance of projects. In this paper, we examined the latest research trends in mining software bug repositories, the techniques employed, and the limitations associated with these approaches. Our analysis focuses on studies published over the past fifteen years, from 2010 to 2024.

Objective of Study:

The goal of this survey is to analyze and synthesize recent trends in mining software bug repositories, providing valuable insights for future research and practical bug management.

Novelty Statement:

This paper introduces several novel contributions, including an exploration of recent trends in bug repository mining, an examination of commonly used test systems for analyzing bug repository data, a comprehensive review of MSR techniques and approaches employed in tasks such as bug localization, triaging, and prediction along with their limitations, and a discussion on potential future trends in bug repository mining. The paper is organized as follows: Material and methods used in this study are explained along with the categorization of active research areas in bug repository mining, then a detailed survey of research contributions is provided, followed by discussion on research limitations and future trends. It then presents the results and discussion. Finally, the paper concludes with a summary.

Material And Methods:

This study presents a comprehensive survey of research conducted over the last 15 years, focusing on the test systems employed in bug repositories, the methodologies commonly used in this area of research, and the prevailing trends shaping the field.

Selection Criteria:

This review includes over 100 research papers focused on bug repositories, with the majority being presented at the MSR conference between 2010 and 2024. To ensure a comprehensive review, we have also incorporated papers from other prominent journals and conferences, including ICCDA, ICSE, IACC, RTITC, CIS, IEEE, CSAC, and the IEEE Transactions on Knowledge and Data Engineering as these are reputed and the most influential venues in the field, possessing high impact in this area of research.

Categorization of Active Research Areas:

The field of mining bug repositories has seen significant contributions from researchers. An analysis of the literature reveals that from 2010 to 2017, research predominantly focused on proposing new techniques for bug classification and prediction. During this period, much of the work was centered around bug categorization. In contrast, the years 2018 to 2024 have seen a shift in focus towards bug localization, resolution, and triaging. A year-wise categorization of active research areas in bug repository mining is illustrated in Figure 1.

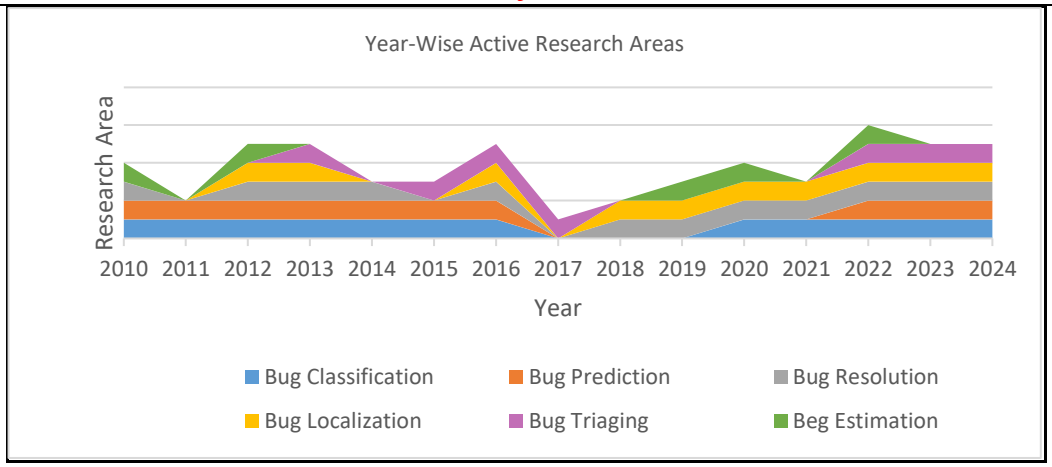

Figure 1. Year-wise Active Research Areas of Mining Bug Repositories

Before presenting a detailed and comprehensive survey based on these categories, a brief definition of a bug, along with its categorization, is provided below: A bug is an error, failure, fault, or flaw in a computer program that causes it to produce unexpected and incorrect results or behave unpredictably.

- **Bug Localization**: The process of identifying and locating the precise position of a bug within the code. This is often a challenging, time-consuming, and costly task.
- **Bug Classification**: The categorization of bugs based on specific properties, such as severity, type, or priority.
- **Bug Estimation**: A technique for assessing the likelihood of a bug's existence, typically by analyzing bug reports. Bugs may sometimes be introduced due to code changes.
- **Bug Resolution**: The process of fixing or eliminating bugs.
- **Bug Triaging**: The process of reviewing and prioritizing bug reports, often handled by a bug tracker or trigger. It involves ensuring the quality of bug reports and assigning them to the appropriate developer for resolution.

An explicit view of research methodology is described in Figure 2.

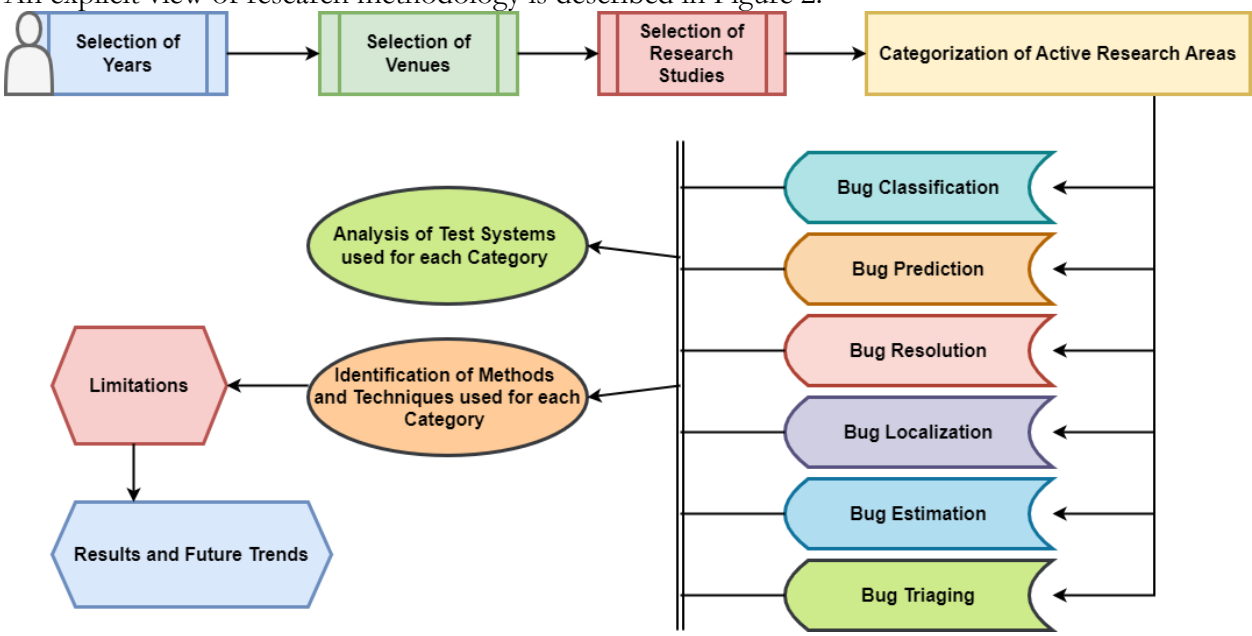

Figure 2. Flow Diagram of Methodology.

Comprehensive Survey:

This section presents an extensive survey that categorizes and summarizes the active research areas in mining software bug repositories over the last fifteen years.

Bug Localization:

Bug localization has been a significant focus in software repository mining, with several methodologies proposed to enhance bug-tracking systems. For instance, some researchers [1] [2] have explored innovative approaches, such as utilizing Stack Overflow's game mechanisms to improve bug tracking. In [3], three distinct methodologies were introduced for constructing statistical regression models aimed at forecasting software flaws and development efforts. The study in [4] [5] delved into the question of whether faults in real software systems are localizable, identifying specific locations of Android bugs within the architectural layers of the system's infrastructure. Researchers [6] [7] have developed algorithms to automatically detect and locate bugs, proposing methods to rank buggy files. Conversely, a search-based methodology for detecting duplicate bugs at BlackBerry was created by the authors in [8]. The correlation between the stability, encapsulation, and popularity of libraries was examined in [7]. Meanwhile, [9] introduced a Query Reformulation (QR) framework for bug localization, offering an innovative approach to this challenge.

Further advancements include the introduction of a novel approach in [10] that scores each file directly against the current bug report, potentially eliminating the need for past code and reports. The study in [11] discussed the application of a Non-Dominant Sorting Genetic Algorithm (NSGA-II) combined with text mining techniques, which outperformed conventional techniques in bug localization. A unique method for LTL specification mining-based bug localization in RTL systems was proposed in [12], where fine-grained assertions are extracted from RTL design to localize faults detected during full-system simulations. In [13], the author examined three generations of bug localization methods, introducing SCOR, a new technique that enhances bug localization in code files by combining semantic word vectors with term-term ordering relationships. Bug localization in web-based systems, specifically through mining crash reports, was addressed in [14], assessing the performance of various standards and methods in classifying crash reports and locating problematic files.

Text mining and genetic algorithms were discussed in [15] for bug localization, with the multi-objective optimization algorithm SPEA II enhancing the localization process. A graph-based neural model called BLoco was proposed in [16] for automated bug localization, targeting buggy source code files for given bug reports. Lastly, [17] introduced a pseudo-Siamese network with semantic guidance for bug localization, leveraging the semantic production of code data to learn and identify bugs. A comprehensive empirical investigation was conducted in [18] to understand the challenges in localizing bugs within deep learning systems, with findings suggesting that the effectiveness of localization techniques varies with bug type.

Bug Classification / Prediction:

Significant work has been done in the area of bug classification and prediction. For instance, the model constructed in [19] distinguishes duplicate bug reports from non-duplicates, facilitating the extraction of similar bug reports. Some researchers [20] have built models to predict bug complexity, while others [21] explored the feasibility of mining by comparing the predictive accuracy of five different classification algorithms. Statistical regression models, including local, global, and hybrid models, were developed in [3] to predict software defects and development efforts. The approach introduced in [22] identifies disguised impact vulnerabilities in bug databases using a text mining classifier. Clone genealogies were examined in [23] to investigate phenomena like mutation and migration, while [24] improved bug deduplication performance by incorporating domain knowledge about software processes and products. A defect model illustrating the relationship between bug-proneness and the strength of relations was proposed in [25], while [26] discussed the impact of misclassification of bugs on previous studies. Models for predicting blocking bugs and performance bugs were introduced in [27] and [28], respectively, with [29] proposing context-aware rank transformations for predictors. A

novel toolset named Exception-Miner was developed in [30] for analyzing Java exception stack traces, and [31] focused on mining Component Repositories for installation issues.

Various probabilistic methods were presented in [32] for categorizing app evaluations, while [33] explored the correlation between pre-release problems and post-release vulnerabilities. Concurrency bugs in complex code bases were characterized in [34], and a multi-stage approach for automating the prediction process was proposed in [35]. The use of a machine learning engine for categorizing issues was discussed in [36], and a bug classification model that filters out unnecessary information from bug reports was suggested in [37]. Advanced machine learning methods [38] were also employed for automated bug classification, as seen in [39], where convolutional neural networks (CNN) with L1 and L2 regularization were proposed. A transfer learning-based classification technique called PIFTNet was introduced in [40], integrating personal data with text features for improved categorization. A multi-view ordinal classification for predicting software bugs was proposed in [41], taking into account the inherent order of class labels. Automatic classification systems based on Word2Vec, LDA, and self-attention mechanisms were explored in [42]. A sentiment analysis and machine learning-based technique for predicting bug priority was proposed in [43], and the use of explainable artificial intelligence (XAI) in bug classification was discussed in [44]. Deep learning techniques for bug bite classification were introduced in [45], with a hybrid method for classifying bugs in cloud computing applications proposed in [46]. A novel deep learning-based method for bug classification was also presented in [47].

Bug Estimation:

Several tools and methodologies have been proposed for software bug estimation in recent years. For example, Rebug Detector, a tool for detecting related bugs using bug code features, was implemented in [48]. The estimation of software bugs from a repository was predicted in [49], while the relationship between defect density, download numbers, software size, and developer was explored in [50]. Bug-inducing modifications in the Android platform were extracted by associating bug reports with specific changes [51]. Trends in bug discussions and feature histories were analyzed in [52] [53], with the history of commits used for repository mining and estimation. A novel framework for automatically discovering genuine specifications from execution traces was presented in [54], while [55] examined the evolution of data races by analyzing committed code samples.

The effects of conceptual concerns on code quality were studied in [56] [57], proposing a new approach to predict the time required to resolve bugs using machine learning. Entropy-based techniques, including Shane entropy and Kapur entropy, were utilized in [58] to forecast the number of software flaws, while a bug risk estimation approach based on duplicate bug reports, software component priority levels, and bug-fix time forecasts was proposed in [59]. The severity of bugs was estimated using entropy-based machine learning algorithms in [60], and the assessment of bug severity through static analysis and source code metrics was discussed in [61]. The application of statistical analysis for comparing the performance of bug prediction and tracing models was presented in [62].

Bug Resolution:

The quality of process data and the characteristics that affect the bug-fixing process were investigated in [63] [64], with an algorithm proposed for transferring bug reports to software fault cases. The role of bug repositories in uncovering details of the software verification process was analyzed in [65] [66], along with an examination of the necessity for additional patches and the common reasons behind incomplete fixes. A correlation between higher software maintainability and faster defect resolution was found in [67]. Essential non-committers were formally defined in [68], identifying bug resolution catalysts through a proposed approach. The process of finding, reporting, and fixing performance defects was examined in [69], while dormant and non-dormant bugs were compared in [70] based on fixing time, size, and the

identity of the fixer. A new approach for automatically extracting bug-fix patterns was proposed in [71], and the application of process mining for effective process management was discussed in [72].

A method for identifying modifications related to bug fixes was introduced in [73], while a large-scale study on bug fixes, focusing on patterns, replacements, deletions, and additions, was detailed in [74]. CLEVER, a method for detecting risky commits before they are merged into the central repository, was introduced in [75], while the correlation between programming languages and bug resolution attributes was analyzed in [76]. Workarounds in bug reports were empirically investigated in [77], with the majority of bug reports fixed as workarounds due to issues related to libraries, settings, and clients. In [78], processes for handling bugs to achieve efficient defect resolution were proposed, with a focus on how bug report vectorization and preprocessing influence assignment accuracy. A CNN-based method for categorizing bug reports as accepted or rejected was introduced in [79], demonstrating notable performance improvements compared to existing methods. Issue resolution times were predicted in [80], with a novel method for determining a bug reporter's reputation based on interactions with bug tracking systems proposed. The identification and analysis of factors influencing issue resolution times in open-source repositories were discussed in [81], with two primary groups of categorical variables i.e., Author Factors and Time Factors, found to impact a bug's lifecycle. A novel approach to enhance bug tracking and resolution by fusing Just-In-Time (JIT) tactics with machine learning was presented in [82]. For automation testing and bug resolution, some advanced machine learning algorithms were also proposed in [83] [84].

Bug Triaging:

In [85], a two-phased location-based method was introduced, recommending bug report assignments based on the anticipated bug location. A novel method for suggesting assignees based on user behavior in bug tracking repositories was proposed in [86] [87], aiming to forecast developers with the appropriate expertise to address new bug reports. The Eclipse and Mozilla Defect Tracking Dataset, specifically curated to include only authentic problems and cover the entire bug-triage lifecycle, was introduced in [88]. The problem of developer prioritization was addressed in [89], where a model for prioritization was developed. Data and discussion related to bug life cycles were explored in [90], focusing on the likelihood of being reopened. The authors of [35] proposed a supervised learning approach to enhance bug triage in open-source projects, with the method presented in [91] significantly improving bug-triaging processes by using a cascade approach. Automated bug triaging, using deep learning techniques like RNNs and LSTMs, was discussed in [92], suggesting improvements in assignment accuracy. The integration of sentiment analysis with bug triage was explored in [93], proposing a sentiment-aware approach to the assignment process. A machine-learning-based model for bug triage was presented in [76], focusing on categorizing bug reports to streamline the triage process. Automated systems for bug triage using unsupervised and semi-supervised learning techniques were discussed in [94], with findings suggesting improvements in accuracy and efficiency.

The effect of assigning bug reports to experienced developers was analyzed in [95], with an automated system for optimizing bug triage processes introduced in [96]. Collaborative filtering techniques were employed in [97] for bug triage, and [98] proposed a method for reducing the time required to resolve bugs by using a hybrid model that incorporates bug report similarity and developer expertise. Finally, an empirical study on bug triage and assignment issues was presented in [99], exploring challenges and solutions in this area. In [51], the relationship between bug fixes and bugs that introduce new problems was explored, and the methodology in [87] proposed an evaluation framework for detecting related bugs. A machine-learning-based bug triage system was discussed in [100], while the proposed approach in [27] classifies app evaluations based on specific bugs, such as performance issues, offering a novel approach to bug report analysis. Studies in [101] proposed a technique for automating bug triage, while [37]

discussed a framework for efficiently identifying and resolving bugs by focusing on relevant information from bug reports. A novel method for categorizing app evaluations based on bug severity was proposed in [39], offering a fresh perspective on bug report analysis. The impact of user-generated bug reports on the software lifecycle was explored in [61], with findings suggesting that user-generated reports significantly impact bug resolution times. A machine-learning-based approach for analyzing bug reports was proposed in [80], while the study in [96] focused on the impact of bug report vectorization and preprocessing on assignment accuracy.

An empirical study on bug report analysis, focusing on the impact of report structure and content on resolution times, was presented in [98]. Findings suggest that well-structured and informative bug reports significantly reduce resolution times. A deep learning-based approach for analyzing bug reports was proposed in [45], focusing on categorizing bugs based on severity and priority levels. The study in [47] introduced a novel framework for analyzing bug reports in real-time, offering a new perspective on bug report analysis. In [65] [66], the support provided by bug repositories in discovering software verification process details was examined, with the study also exploring the need for supplementary patches and common causes of incomplete fixes. A novel approach for managing bug repositories was proposed in [67], focusing on improving the efficiency of bug tracking and resolution processes. The impact of bug repository management on software maintenance was explored in [68], with findings suggesting that effective management significantly reduces bug resolution times. A machine-learning-based approach for managing bug repositories was proposed in [72], focusing on automating the bug triage process. A novel framework for managing bug repositories was introduced in [91], offering a new perspective on bug repository management. The study in [75] focused on the impact of bug repository management on software quality, with findings suggesting that well-managed repositories significantly improve software quality. A machine-learning-based approach for managing bug repositories was proposed in [77], focusing on automating the bug triage process.

The study in [94], introduced a novel framework for managing bug repositories, focusing on improving the efficiency of bug tracking and resolution processes. The impact of bug repository management on software maintenance was explored in [81], with findings suggesting that effective management significantly reduces bug resolution times. A machine-learning-based approach for managing bug repositories was proposed in [99], focusing on automating the bug triage process. For better bug repository management, Table 1 describes the summary of test systems based on each research area, along with their specifications.

Table 1. Test Systems based on Research-Area

Test System	Research Area	Open Source	Language
ASPECTJ	Bug localization	Yes	Java
Chrome	Bug localization	No	Android
Mozilla Firefox	Bug localization, Bug classification/prediction, Bug estimation, Bug triaging	Yes	Android
Launchpad	Bug localization	Yes	Android
Eclipse	Bug localization, Bug classification/prediction, Bug estimation, Bug resolution, Bug triaging	Yes	Java
SWT	Bug localization	Yes	Java
Zxing	Bug localization	Yes	Java
Tomcat	Bug localization	Yes	Java
Argo UML	Bug localization	Yes	UML

OpenOffice	Bug classification/prediction, Bug estimation	Yes	C++, Java
FreeBSD	Bug classification/prediction	Yes	C, Python, and Perl
MySQL	Bug classification/prediction	Yes	JavaFX
CHINA Lucene 2.4	Bug classification/prediction	Yes	Java
NasaCoc Xalan 2	Bug classification/prediction	Yes	Java
XML	Bug classification/prediction, Bug estimation	Yes	C++, Python, and Perl
Linux	Bug classification/prediction	Yes	C, Python, and Perl
NetBeans	Bug classification/prediction	Yes	Java
Chromium	Bug classification/prediction, Bug resolution	Yes	Python
Free Desktop	Bug classification/prediction	Yes	Java, C#
Source-Forge	Bug classification/prediction	Yes	C++
Google-Code	Bug classification/prediction	Yes	C++, Python
Google Play	Bug classification/prediction, Bug resolution	No	Java
iOS (Apple)	Bug classification/prediction	No	Swift
Bugzilla	Bug classification/prediction, Bug resolution	Yes	Perl
Jboss	Bug classification/prediction	Yes	Java
Ant	Bug classification/prediction	Yes	Java
Lenya	Bug classification/prediction	Yes	Python, Java
TomCat5	Bug classification/prediction	Yes	Java
Redhat	Bug classification/prediction	Yes	Python
OpenFOAM	Bug classification/prediction	Yes	C++
Apache	Bug classification/prediction, Bug estimation	Yes	C
Lucene-Java	Bug estimation	Yes	Java
GNOME	Bug estimation	Yes	C, C++, C#, XML, HTML, CSS, JavaScript, Vala, Python.
JEDIT	Bug estimation	Yes	Java
COLUMBA	Bug estimation	Yes	Java
Groovy	Bug resolution	Yes	Java
CherryPy	Bug resolution	Yes	Python
FogBugz	Bug resolution	Yes	Java, C#
Jazz ALM	Bug resolution	Yes	C, C++
RTC	Bug resolution	Yes	C, C++
ATLAS Reconstruction	Bug triaging	Yes	C++, Python
UNICASE	Bug triaging	Yes	Python
NetBeans	Bug triaging	Yes	Java
Maemo	Bug triaging	Yes	C, Python

According to the above-discussed literature, Table 2 offers a thorough analysis of the strategies and tactics used in the designated field of study. The research areas are represented by each row, while the various respective techniques or methodologies are represented by the columns.

Table 2. Methods / Techniques used in specific Research-Area

Research Area	Techniques / Methodologies			
	Machine Learning	**Statistical**	**Information Retrieval / Social Network Analysis**	**Natural Language Processing**
Bug localization	Regression test suite, clustering, bug layer classification, bayesian belief networks	Descriptive statistics, similarity metric, probabilistic score, lexical similarity score	IR, bug reporters social network analysis, bug staring analysis	Manual inspection
Bug classification / prediction	SVM, 0-R, decision tree, naive bayes, logistic regression, K-NN, random forest	Cosine similarity, Jaccord similarity, TF-IDF similarity	IR, social network properties	NLP techniques; text mining, sentiment analysis techniques
Bug estimation	Regression analysis, case base reasoning,	Patches filters, source code filters, stack traces filters, enumeration filters	Context selection, co-evolutionary graphs	Text similarity,
Bug resolution	Maintainability model, classification, pattern finding	Correlation matrix, hill climbing method	Minimal essential graph, social network analysis techniques,	Text similarity
Bug triaging	Classification, instance selection and feature selection, expectation maximization	Statistical methods		User topics associations, the noun extraction process, the term weighting scheme

Table 2 demonstrates which Information Retrieval (IR), Machine Learning (ML), Social Network Analysis (SNA), and Natural Language Processing (NLP) techniques greatly improve bug mining and deal with the complexity of contemporary software systems. These technologies speed up the software development process, enhance bug localization, and enable early defect identification. Although these approaches have many advantages, there are still difficulties in adjusting them to different software environments and making sure developers can understand them.

Limitations And Future Trends:

Limitations in Bug Localization:

Bug localization continues to face significant challenges, primarily due to the scarcity of labeled training data essential for deep learning models. Traditional methods, although widely used, struggle with accuracy and time efficiency. Training robust bug localization models is

hindered by the arduous task of gathering substantial amounts of accurately labeled bug data, which is both time-consuming and difficult. Furthermore, the complexity of extensive codebases, limited tooling, and inadequate automated methods contribute to inaccurate results and context sensitivity issues. Human factors, such as developer experience and familiarity with the code, also play a role. These limitations can lead to prolonged debugging periods, lower software quality, resource exhaustion, and maintenance challenges, ultimately impacting the overall efficiency and effectiveness of software development projects.

Limitations in Bug Classification and Prediction:

Bug classification and prediction methodologies face limitations in enhancing text encoding vectors due to constraints in dataset size and quality. Additionally, existing techniques often overlook the diverse characteristics of bug reports, leading to inefficiencies in their classification. The manual construction of action vocabularies for bug report validation is further complicated by the variability in user-defined activities, affecting the overall validation process. Relying solely on the textual content of bug reports can result in inaccuracies, as the underlying intent of the report is not always fully considered. These limitations can lead to decreased effectiveness in problem detection, missed or overlooked bugs, misallocation of resources, and ultimately, compromised software quality and user experience.

Limitations in Bug Estimation:

In the realm of bug estimation, current techniques struggle with accurately predicting the severity of bugs. While code metrics such as Lines of Code (LOC) and FanOut are valuable for identifying faulty code, they fall short of accurately assessing the seriousness of errors. Static analysis tools like Spot Bugs and Infer, though commonly used, also exhibit limitations in predicting bugs and assigning appropriate severity labels. The impact of severe bugs, particularly those related to security, underscores the need for improved bug assessment approaches that account for varying severity levels. The study highlights the potential benefits of integrating code metrics with static analysis techniques to enhance the accuracy of severity estimation. These findings emphasize the ongoing challenges in accurately determining bug severity in software engineering, pointing to the necessity for more reliable and comprehensive prediction models.

Limitations in Bug Resolution:

Bug resolution processes are hampered by several challenges, including the influence of programming languages on the time and effort required to resolve issues, the uncertainty surrounding workarounds, and the overwhelming amount of search results and resources developers must sift through. Additionally, the increasing volume of daily bug reports adds to the workload of developers, making the resolution process more complex and time-consuming. These challenges highlight the need for more efficient and effective bug resolution strategies, which are critical for maintaining software quality and meeting project deadlines.

Limitations in Bug Triaging:

Bug triaging faces significant limitations, primarily due to the manual and time-consuming nature of assigning bugs. This process is prone to inefficiencies, especially when handling a large volume of defect reports. Traditional bug assignment methods often overlook critical factors such as developer expertise, bug type, and resolution time, leading to higher costs and resource allocation challenges. To address these limitations, researchers have proposed innovative solutions such as machine learning-based recommender systems and explainable AI models for bug assignment. Tools like Bugs by have also been developed to automate bug analysis and improve triaging accuracy. These advancements aim to enhance the efficiency, accuracy, and transparency of bug triaging in software development projects.

Future Trends:

The future of bug repository mining is poised to leverage Mining Unstructured Data (MUD) techniques, which focus on the growing volume of unstructured data in issue trackers, versioning systems, and other repositories. Machine learning approaches [102] for pattern

recognition in software repositories are also expected to advance, offering new opportunities for knowledge discovery and decision-making systems within organizations. Additionally, the collaborative development of a heuristic repository, where researchers can share various heuristics for classifying software engineering artifacts, holds promise for improving classification accuracy. Finally, the application of social network analysis techniques [103] and heterogeneous graph-based models to automatically identify communities of software engineers with shared interests could lead to more effective collaboration and knowledge sharing in software projects.

Results And Discussion:

The findings of this study highlighted that the challenges identified in bug repository mining are closely aligned with broader trends in software engineering, particularly the shift towards DevOps and CI/CD pipelines. Nowadays number of organizations are adopting these methodologies, thus the need for effective bug tracking and resolution has become paramount. CI/CD pipelines automate the software delivery process, which necessitates efficient bug tracking to ensure quality. Bug repository mining can enhance this by identifying severe bugs early in the development cycle, thus improving the overall workflow. The study of embedded IoT systems shows that communities are actively discussing and solving CI/CD challenges, indicating a collaborative approach to bug resolution that aligns with DevOps principles [104]. Moreover, some tools like Garimpeiro help facilitate the analysis of CI/CD practices in repositories, helping teams understand the impact of their bug-fixing efforts on deployment efficiency [105]. Another finding is that the AI and ML integration in bug repository mining has revolutionized the software development process by enhancing the classification, prediction, and vulnerability detection of bugs. This is done by using advanced algorithms [106] and automated techniques [38], that analyze vast datasets, uncover hidden patterns, and overall improve software quality.

Significant improvements were seen by using machine learning techniques for classifying bugs, for instance, a study [107] classified over 126,000 bugs into nine categories therefore achieving around a 29.73% increase in classifier performance compared to traditional methods. Also study at [108] also shows that deep learning models used for bug prediction have outperformed conventional classifiers in small datasets and have shown higher performance in predicting buggy classes, which majorly reduces the cost associated with error fixing later in the software lifecycle. As AI and ML continue to evolve, their application in software repository mining will likely expand, through offering new insights and improving decision-making processes in software development.

Concluding Remarks:

In this survey paper, we examined over 100 papers focused on mining bug repositories, identifying research trends, current limitations, and potential future directions in this field. The majority of the papers were selected from the MSR conference. Analyzing the contributions of these studies, we observed that the first eight years primarily focused on developing tools and techniques for bug prediction and fault detection. In contrast, the past seven years have seen increased attention to bug triaging, resolution, and classification. Over the entire 15-year period, bug classification and prediction have remained the most active and prominent areas in bug repository mining. This survey provides a comprehensive overview of the evolution of this field, serving as a valuable resource for researchers interested in mining bug repositories. It offers insights into recent research trends and highlights key techniques and methodologies across various areas of bug repository mining. This work is expected to aid future researchers in gaining a deeper understanding of the field and assist project managers in optimizing software development processes for improved efficiency and effectiveness.

References:

[1] R. Lotufo, L. Passos, and K. Czarnecki, "Towards improving bug tracking systems with game mechanisms," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 2–11, 2012, doi: 10.1109/MSR.2012.6224293.

[2] B. Sisman and A. C. Kak, "Incorporating version histories in Information Retrieval based bug localization," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 50–59, 2012, doi: 10.1109/MSR.2012.6224299.

[3] N. Bettenburg, M. Nagappan, and A. E. Hassan, "Think locally, act globally: Improving defect and effort prediction models," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 60–69, 2012, doi: 10.1109/MSR.2012.6224300.

[4] Lucia, F. Thung, D. Lo, and L. Jiang, "Are faults localizable?," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 74–77, 2012, doi: 10.1109/MSR.2012.6224302.

[5] V. Guana, F. Rocha, A. Hindle, and E. Stroulia, "Do the stars align? Multidimensional analysis of Android's layered architecture," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 124–127, 2012, doi: 10.1109/MSR.2012.6224269.

[6] A. Breckel, "Error mining: Bug detection through comparison with large code databases," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 175–178, 2012, doi: 10.1109/MSR.2012.6224278.

[7] S. Wang, F. Khomh, and Y. Zou, "Improving bug localization using correlations in crash reports," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 247–256, 2013, doi: 10.1109/MSR.2013.6624036.

[8] M. Amoui, N. Kaushik, A. Al-Dabbagh, L. Tahvildari, S. Li, and W. Liu, "Search-based duplicate defect detection: An industrial experience," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 173–182, 2013, doi: 10.1109/MSR.2013.6624025.

[9] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 309–318, 2013, doi: 10.1109/MSR.2013.6624044.

[10] "Locating Bugs without Looking Back | IEEE Conference Publication | IEEE Xplore." Accessed: Aug. 28, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7832908

[11] R. Malhotra, S. Aggarwal, R. Girdhar, and R. Chugh, "Bug localization in software using NSGA-II," ISCAIE 2018 - 2018 IEEE Symp. Comput. Appl. Ind. Electron., pp. 428–433, Jul. 2018, doi: 10.1109/ISCAIE.2018.8405511.

[12] V. Iyer, D. Kim, B. Nikolic, and S. A. Seshia, "RTL bug localization through LTL specification mining (WIP)," MEMOCODE 2019 - 17th ACM-IEEE Int. Conf. Form. Methods Model. Syst. Des., Oct. 2019, doi: 10.1145/3359986.3361202.

[13] S. A. Akbar, "Source Code Search for Automatic Bug Localization," Theses Diss. Available from ProQuest, Jan. 2020, Accessed: Aug. 28, 2024. [Online]. Available: https://docs.lib.purdue.edu/dissertations/AAI30504782

[14] M. Medeiros, U. Kulesza, R. Bonifacio, E. Adachi, and R. Coelho, "Improving Bug Localization by Mining Crash Reports: An Industrial Study," Proc. - 2020 IEEE Int. Conf. Softw. Maint. Evol. ICSME 2020, pp. 766–775, Sep. 2020, doi: 10.1109/ICSME46990.2020.00086.

[15] A. Sood et al., "Bug Localization Using Multi-objective Approach and Information Retrieval," Adv. Intell. Syst. Comput., vol. 1165, pp. 709–723, 2021, doi: 10.1007/978-981-15-5113-0_58.

[16] Z. Zhu, H. Tong, Y. Wang, and Y. Li, "Enhancing bug localization with bug report decomposition and code hierarchical network," Knowledge-Based Syst., vol. 248, p. 108741, Jul. 2022, doi: 10.1016/J.KNOSYS.2022.108741.

[17] Y. Zhao, X. Li, Y. Li, Y. Zhang, W. Qi, and J. Song, "Bug localization with semantic

guidance using pseudo-Siamese network," SPIE, vol. 12721, p. 127210A, Jun. 2023, doi: 10.1117/12.2683291.

[18] S. Jahan, M. B. Shah, and M. M. Rahman, "Towards Understanding the Challenges of Bug Localization in Deep Learning Systems," Feb. 2024, Accessed: Aug. 28, 2024. [Online]. Available: https://arxiv.org/abs/2402.01021v1

[19] C. Sun, D. Lo, X. Wang, J. Jiang, and S. C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," Proc. - Int. Conf. Softw. Eng., vol. 1, pp. 45–54, 2010, doi: 10.1145/1806799.1806811.

[20] N. K. Nagwani and A. Bhansali, "A data mining model to predict software bug complexity using bug estimation and clustering," ITC 2010 - 2010 Int. Conf. Recent Trends Information, Telecommun. Comput., pp. 13–17, 2010, doi: 10.1109/ITC.2010.56.

[21] G. Bougie, C. Treude, D. M. German, and M. A. Storey, "A comparative exploration of FreeBSD bug lifetimes," Proc. - Int. Conf. Softw. Eng., pp. 106–109, 2010, doi: 10.1109/MSR.2010.5463291.

[22] D. Wijayasekara, M. Manic, J. L. Wright, and M. McQueen, "Mining bug databases for unidentified software vulnerabilities," Int. Conf. Hum. Syst. Interact. HSI, pp. 89–96, 2012, doi: 10.1109/HSI.2012.22.

[23] S. Xie, F. Khomh, and Y. Zou, "An empirical study of the fault-proneness of clone mutation and clone migration," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 149–158, 2013, doi: 10.1109/MSR.2013.6624022.

[24] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 183–192, 2013, doi: 10.1109/MSR.2013.6624026.

[25] W. Hu and K. Wong, "Using Citation Influence to Predict Software Defects".

[26] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," Proc. - Int. Conf. Softw. Eng., pp. 392–401, 2013, doi: 10.1109/ICSE.2013.6606585.

[27] H. Valdivia-Garcia, E. Shihab, and M. Nagappan, "Characterizing and predicting blocking bugs in open source projects," J. Syst. Softw., vol. 143, pp. 44–58, Sep. 2018, doi: 10.1016/J.JSS.2018.03.053.

[28] Y. Liu, C. Xu, and S. C. Cheung, "Characterizing and detecting performance bugs for smartphone applications," Proc. - Int. Conf. Softw. Eng., no. 1, pp. 1013–1024, May 2014, doi: 10.1145/2568225.2568229.

[29] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model," 11th Work. Conf. Min. Softw. Repos. MSR 2014 - Proc., pp. 182–191, May 2014, doi: 10.1145/2597073.2597078.

[30] R. Coelho, L. Almeida, G. Gousios, and A. Van Deursen, "Unveiling exception handling bug hazards in android based on GitHub and Google code issues," IEEE Int. Work. Conf. Min. Softw. Repos., vol. 2015-August, pp. 134–145, Aug. 2015, doi: 10.1109/MSR.2015.20.

[31] P. Abate, R. Di Cosmo, L. Gesbert, F. Le Fessant, R. Treinen, and S. Zacchiroli, "Mining component repositories for installability issues," IEEE Int. Work. Conf. Min. Softw. Repos., vol. 2015-August, pp. 24–33, Aug. 2015, doi: 10.1109/MSR.2015.10.

[32] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," 2015 IEEE 23rd Int. Requir. Eng. Conf. RE 2015 - Proc., pp. 116–125, Nov. 2015, doi: 10.1109/RE.2015.7320414.

[33] F. Camilo, A. Meneely, and M. Nagappan, "Do bugs foreshadow vulnerabilities? A study of the chromium project," IEEE Int. Work. Conf. Min. Softw. Repos., vol. 2015-August, pp. 269–279, Aug. 2015, doi: 10.1109/MSR.2015.32.

[34] P. Ciancarini, F. Poggi, D. Rossi, and A. Sillitti, "Mining Concurrency Bugs", Accessed: Aug. 27, 2024. [Online]. Available: https://bz.apache.org/bugzilla/

[35] J. Xuan et al., "Towards effective bug triage with software data reduction techniques," IEEE Trans. Knowl. Data Eng., vol. 27, no. 1, pp. 264–280, Jan. 2015, doi: 10.1109/TKDE.2014.2324590.

[36] "U.S. Patent and Trademark Office | U.S. Department of Commerce." Accessed: Aug. 28, 2024. [Online]. Available: https://www.commerce.gov/bureaus-and-offices/uspto

[37] F. Fang, J. Wu, Y. Li, X. Ye, W. Aljedaani, and M. W. Mkaouer, "On the classification of bug reports to improve bug localization," Soft Comput., vol. 25, no. 11, pp. 7307–7323, Jun. 2021, doi: 10.1007/S00500-021-05689-2/METRICS.

[38] M. N. Noor, T. A. Khan, F. Haneef, and M. I. Ramay, "Machine Learning Model to Predict Automated Testing Adoption," https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/IJSI.293268, vol. 10, no. 1, pp. 1–15, Jan. 1AD, doi: 10.4018/IJSI.293268.

[39] O. Koksal and C. E. Ozturk, "A Survey on Machine Learning-based Automated Software Bug Report Classification," ISMSIT 2022 - 6th Int. Symp. Multidiscip. Stud. Innov. Technol. Proc., pp. 635–640, 2022, doi: 10.1109/ISMSIT56059.2022.9932822.

[40] L. Zhifang, W. Kun, Z. Qi, L. Shengzong, Z. Yan, and H. Jianbiao, "Classification of open source software bug report based on transfer learning," Expert Syst., vol. 41, no. 5, p. e13184, May 2024, doi: 10.1111/EXSY.13184.

[41] P. Yildirim Taser, "A novel multi-view ordinal classification approach for software bug prediction," Expert Syst., vol. 39, no. 7, p. e13044, Aug. 2022, doi: 10.1111/EXSY.13044.

[42] Y. Tang, H. Zhou, and H. Su, "Automatic Classification of Software Bug Reports Based on LDA and Word2Vec," 2022 2nd Int. Conf. Comput. Sci. Electron. Inf. Eng. Intell. Control Technol. CEI 2022, pp. 491–495, 2022, doi: 10.1109/CEI57409.2022.9950207.

[43] A. Singh, P. K. Kapur, and V. B. Singh, "Developing classifiers by considering sentiment analysis of reported bugs for priority prediction," Int. J. Syst. Assur. Eng. Manag., vol. 15, no. 5, pp. 1888–1899, May 2024, doi: 10.1007/S13198-023-02199-2/METRICS.

[44] L. Chmielowski and M. Kucharzak, "Impact of Software Bug Report Preprocessing and Vectorization on Bug Assignment Accuracy," Lect. Notes Networks Syst., vol. 255, pp. 153–162, 2022, doi: 10.1007/978-3-030-81523-3_15.

[45] B. Ilijoski et al., "Deep Learning Methods for Bug Bite Classification: An End-to-End System," Appl. Sci. 2023, Vol. 13, Page 5187, vol. 13, no. 8, p. 5187, Apr. 2023, doi: 10.3390/APP13085187.

[46] N. Tabassum, A. Namoun, T. Alyas, A. Tufail, M. Taqi, and K. H. Kim, "Classification of Bugs in Cloud Computing Applications Using Machine Learning Techniques," Appl. Sci. 2023, Vol. 13, Page 2880, vol. 13, no. 5, p. 2880, Feb. 2023, doi: 10.3390/APP13052880.

[47] J. P. Meher, S. Biswas, and R. Mall, "Deep learning-based software bug classification," Inf. Softw. Technol., vol. 166, p. 107350, Feb. 2024, doi: 10.1016/J.INFSOF.2023.107350.

[48] D. Wang, M. Lin, H. Zhang, and H. Hu, "Detect related bugs from source code using bug information," Proc. - Int. Comput. Softw. Appl. Conf., pp. 228–237, 2010, doi: 10.1109/COMPSAC.2010.27.

[49] N. K. Nagwani and S. Verma, "Predictive data mining model for software bug estimation using average weighted similarity," 2010 IEEE 2nd Int. Adv. Comput. Conf. IACC 2010, pp. 373–378, 2010, doi: 10.1109/IADCC.2010.5422923.

[50] C. Rahmani and D. Khazanchi, "A study on defect density of open source software," Proc. - 9th IEEE/ACIS Int. Conf. Comput. Inf. Sci. ICIS 2010, pp. 679–683, 2010, doi:

10.1109/ICIS.2010.11.

[51] M. Asaduzzaman, M. C. Bullock, C. K. Roy, and K. A. Schneider, "Bug introducing changes: A case study with Android," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 116–119, 2012, doi: 10.1109/MSR.2012.6224267.

[52] L. Martie, V. K. Palepu, H. Sajnani, and C. Lopes, "Trendy bugs: Topic trends in the Android bug reports," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 120–123, 2012, doi: 10.1109/MSR.2012.6224268.

[53] M. Steff and B. Russo, "Co-evolution of logical couplings and commits for defect estimation," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 213–216, 2012, doi: 10.1109/MSR.2012.6224283.

[54] A. C. Nguyen and S. C. Khoo, "Discovering complete API rules with mutation testing," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 151–160, 2012, doi: 10.1109/MSR.2012.6224275.

[55] C. Sadowski, J. Yi, and S. Kim, "The evolution of data races," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 171–174, 2012, doi: 10.1109/MSR.2012.6224277.

[56] T. H. Chen, S. W. Thomas, M. Nagappan, and A. E. Hassan, "Explaining software defects using topic models," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 189–198, 2012, doi: 10.1109/MSR.2012.6224280.

[57] R. Sawarkar, N. K. Nagwani, and S. Kumar, "Predicting Bug Estimation Time for Newly Reported Bug Using Machine Learning Algorithms," 2019 IEEE 5th Int. Conf. Converg. Technol. I2CT 2019, Mar. 2019, doi: 10.1109/I2CT45611.2019.9033749.

[58] A. Munde, "Envisaging Bugs by Means of Entropy Measures," Smart Innov. Syst. Technol., vol. 196, pp. 149–156, 2021, doi: 10.1007/978-981-15-7062-9_15.

[59] H. Mahfoodh and Q. Obediat, "Software Risk Estimation through Bug Reports Analysis and Bug-fix Time Predictions," 2020 Int. Conf. Innov. Intell. Informatics, Comput. Technol. 3ICT 2020, Dec. 2020, doi: 10.1109/3ICT51146.2020.9312003.

[60] M. Kumari, U. K. Singh, and M. Sharma, "Entropy Based Machine Learning Models for Software Bug Severity Assessment in Cross Project Context," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 12254 LNCS, pp. 939–953, 2020, doi: 10.1007/978-3-030-58817-5_66.

[61] E. Mashhadi, S. Chowdhury, S. Modaberi, H. Hemmati, and G. Uddin, "An Empirical Study on Bug Severity Estimation using Source Code Metrics and Static Analysis," Jun. 2022, doi: 10.1016/j.jss.2024.112179.

[62] D. Tambe and L. Ragha, "Analysis of Software Bug Prediction and Tracing Models from a Statistical Perspective Using Machine Learning," 2022 2nd Int. Conf. Intell. Technol. CONIT 2022, 2022, doi: 10.1109/CONIT55038.2022.9848385.

[63] A. Bachmann and A. Bernstein, "When process data quality affects the number of bugs: Correlations in software engineering datasets," Proc. - Int. Conf. Softw. Eng., pp. 62–71, 2010, doi: 10.1109/MSR.2010.5463286.

[64] Y. Liu and K. Ben, "Knowledge representation of software faults based on open bug repository," 2010 Int. Conf. Comput. Des. Appl. ICCDA 2010, vol. 2, 2010, doi: 10.1109/ICCDA.2010.5541110.

[65] R. Souza and C. Chavez, "Characterizing verification of bug fixes in two open source IDEs," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 70–73, 2012, doi: 10.1109/MSR.2012.6224301.

[66] J. Park, M. Kim, B. Ray, and D.-H. Bae, "An Empirical Study of Supplementary Bug Fixes", Accessed: Aug. 27, 2024. [Online]. Available: http://cvs2svn.tigris.org/

[67] A. Issabayeva, A. Nugroho, and J. Visser, "Issue handling performance in proprietary software projects," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 209–212, 2012, doi: 10.1109/MSR.2012.6224282.

[68] S. Mani, S. Nagar, D. Mukherjee, R. Narayanam, V. S. Sinha, and A. A. Nanavati, "Bug resolution catalysts: Identifying essential non-committers from bug repositories," 2013 10th IEEE Work. Conf. Min. Softw. Repos. (MSR 2013), pp. 193–202, May 2013, doi: 10.1109/MSR.2013.6624027.

[69] A. Nistor, T. Jiang, and L. Tan, "Discovering, reporting, and fixing performance bugs," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 237–246, 2013, doi: 10.1109/MSR.2013.6624035.

[70] T. H. Chen, M. Nagappan, E. Shihab, and A. E. Hassan, "An empirical study of dormant bugs," 11th Work. Conf. Min. Softw. Repos. MSR 2014 - Proc., pp. 82–91, May 2014, doi: 10.1145/2597073.2597108.

[71] H. Osman, M. Lungu, and O. Nierstrasz, "Mining frequent bug-fix code changes," 2014 Softw. Evol. Week - IEEE Conf. Softw. Maintenance, Reengineering, Reverse Eng. CSMR-WCRE 2014 - Proc., pp. 343–347, 2014, doi: 10.1109/CSMR-WCRE.2014.6747191.

[72] M. Gupta, A. Sureka, and S. Padmanabhuni, "Process mining multiple repositories for software defect resolution from control and organizational perspective," 11th Work. Conf. Min. Softw. Repos. MSR 2014 - Proc., pp. 122–131, May 2014, doi: 10.1145/2597073.2597081.

[73] H. Oumarou, N. Anquetil, A. Etien, S. Ducasse, and K. D. Taiwe, "Identifying the Exact Bug Fixing Actions," Proc. - 7th Int. Work. Empir. Softw. Eng. Pract. IWESEP 2016, pp. 51–56, May 2016, doi: 10.1109/IWESEP.2016.13.

[74] "A Deeper Look into Bug Fixes: Patterns, Replacements, Deletions, and Additions | IEEE Conference Publication | IEEE Xplore." Accessed: Aug. 28, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/7832938

[75] "CLEVER: Combining Code Metrics with Clone Detection for Just-in-Time Fault Prevention and Resolution in Large Industrial Projects | IEEE Conference Publication | IEEE Xplore." Accessed: Aug. 28, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/8595198

[76] J. M. Zhang et al., "A Study of Bug Resolution Characteristics in Popular Programming Languages," IEEE Trans. Softw. Eng., vol. 47, no. 12, pp. 2684–2697, Dec. 2021, doi: 10.1109/TSE.2019.2961897.

[77] "The Symptom, Cause and Repair of Workaround | IEEE Conference Publication | IEEE Xplore." Accessed: Aug. 28, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/9286099

[78] L. Chmielowski, M. Kucharzak, and R. Burduk, "APPLICATION OF EXPLAINABLE ARTIFICIAL INTELLIGENCE IN SOFTWARE BUG CLASSIFICATION," Inform. Autom. Pomiary w Gospod. i Ochr. Środowiska, vol. 13, no. 1, pp. 14–17, Mar. 2023, doi: 10.35784/IAPGOS.3396.

[79] M. A. Arshad and H. Zhiqiu, "Using CNN to Predict the Resolution Status of Bug Reports," J. Phys. Conf. Ser., vol. 1828, no. 1, p. 012106, Feb. 2021, doi: 10.1088/1742-6596/1828/1/012106.

[80] M. K. Yucel and A. Tosun, "Measuring Bug Reporter's Reputation and Its Effect on Bug Resolution Time Prediction," Proc. - 7th Int. Conf. Comput. Sci. Eng. UBMK 2022, pp. 110–115, 2022, doi: 10.1109/UBMK55850.2022.9919454.

[81] E. Eiroa-Lledo, R. H. Ali, G. Pinto, J. Anderson, and E. Linstead, "Large-Scale Identification and Analysis of Factors Impacting Simple Bug Resolution Times in Open Source Software Repositories," Appl. Sci. 2023, Vol. 13, Page 3150, vol. 13, no. 5, p. 3150, Feb. 2023, doi: 10.3390/APP13053150.

[82] "Machine Learning and Just-in-Time Strategies for Effective Bug Tracking in Software Development | International Journal of Intelligent Systems and Applications in

Engineering." Accessed: Aug. 28, 2024. [Online]. Available: https://ijisae.org/index.php/IJISAE/article/view/4013

[83] F. Haneef and M. A. Sindhu, "DLIQ: A Deterministic Finite Automaton Learning Algorithm through Inverse Queries," Inf. Technol. Control, vol. 51, no. 4, pp. 611–624, Dec. 2022, doi: 10.5755/J01.ITC.51.4.31394.

[84] F. Haneef and M. A. Sindhu, "IDLIQ: An Incremental Deterministic Finite Automaton Learning Algorithm Through Inverse Queries for Regular Grammar Inference," https://home.liebertpub.com/big, May 2023, doi: 10.1089/BIG.2022.0158.

[85] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 2–11, 2013, doi: 10.1109/MSR.2013.6623997.

[86] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 22–30, 2013, doi: 10.1109/MSR.2013.6623999.

[87] "(PDF) Efficient Bug Triaging Using Text Mining." Accessed: Aug. 27, 2024. [Online]. Available: https://www.researchgate.net/publication/235911086_Efficient_Bug_Triaging_Using_Text_Mining

[88] A. Lamkanfi, J. Pérez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: A genuine dataset for mining bug information," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 203–206, 2013, doi: 10.1109/MSR.2013.6624028.

[89] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," Proc. - Int. Conf. Softw. Eng., pp. 25–35, 2012, doi: 10.1109/ICSE.2012.6227209.

[90] M. E. Joorabchi, M. Mirzaaghaei, and A. Mesbah, "Works for me! characterizing non-reproducible bug reports," 11th Work. Conf. Min. Softw. Repos. MSR 2014 - Proc., pp. 62–71, May 2014, doi: 10.1145/2597073.2597098.

[91] C. A. Thompson, G. C. Murphy, M. Palyart, and M. Gašparič, "How Software developers use work breakdown relationships in issue repositories," Proc. - 13th Work. Conf. Min. Softw. Repos. MSR 2016, pp. 281–285, May 2016, doi: 10.1145/2901739.2901779.

[92] Tamanna and O. P. Sangwan, "Review of text mining techniques for software bug localization," Proc. 9th Int. Conf. Cloud Comput. Data Sci. Eng. Conflu. 2019, pp. 208–211, Jan. 2019, doi: 10.1109/CONFLUENCE.2019.8776959.

[93] A. Chauhan and R. Kumar, "Bug Severity Classification Using Semantic Feature with Convolution Neural Network," Adv. Intell. Syst. Comput., vol. 1025, pp. 327–335, 2019, doi: 10.1007/978-981-32-9515-5_31.

[94] S. Panthaplackel, J. J. Li, M. Gligoric, and R. J. Mooney, "Learning to Describe Solutions for Bug Reports Based on Developer Discussions," Proc. Annu. Meet. Assoc. Comput. Linguist., pp. 2935–2952, 2022, doi: 10.18653/V1/2022.FINDINGS-ACL.231.

[95] "GitHub - HadiJahanshahi/ADPTriage: ADPTriage: Approximate Dynamic Programming for Bug Triage." Accessed: Aug. 28, 2024. [Online]. Available: https://github.com/HadiJahanshahi/ADPTriage

[96] R. R. Panda and N. K. Nagwani, "An Improved Software Bug Triaging Approach Based on Topic Modeling and Fuzzy Logic," Lect. Notes Networks Syst., vol. 479, pp. 337–346, 2023, doi: 10.1007/978-981-19-3148-2_29.

[97] Y. Liu, X. Qi, J. Zhang, H. Li, X. Ge, and J. Ai, "Automatic Bug Triaging via Deep Reinforcement Learning," Appl. Sci. 2022, Vol. 12, Page 3565, vol. 12, no. 7, p. 3565, Mar. 2022, doi: 10.3390/APP12073565.

[98] J. Jang and G. Yang, "A Bug Triage Technique Using Developer-Based Feature

Selection and CNN-LSTM Algorithm," Appl. Sci. 2022, Vol. 12, Page 9358, vol. 12, no. 18, p. 9358, Sep. 2022, doi: 10.3390/APP12189358.

[99] H. Dong, H. Ren, J. Shi, Y. Xie, and X. Hu, "Neighborhood contrastive learning-based graph neural network for bug triaging," Sci. Comput. Program., vol. 235, p. 103093, Jul. 2024, doi: 10.1016/J.SCICO.2024.103093.

[100] S. Raemaekers, G. F. Nane, A. Van Deursen, and J. Visser, "Testing principles, current practices, and effects of change localization," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 257–266, 2013, doi: 10.1109/MSR.2013.6624037.

[101] & S. Hart, G. W., Kern Jr, E. C., "F. C. (1989). U.S. Patent No. 4,858,141. Washington, DC: U.S. Patent and Trademark Office.".

[102] F. Haneef, "Review of Automaton Learning Algorithms with Polynomial Complexity -- Completely Solved Examples," Apr. 2024, Accessed: Aug. 28, 2024. [Online]. Available: https://arxiv.org/abs/2404.11096v1

[103] "(PDF) A Review on Big Data and Social Network Analytics Techniques." Accessed: Aug. 28, 2024. [Online]. Available: https://www.researchgate.net/publication/342144889_A_Review_on_Big_Data_and_Social_Network_Analytics_Techniques

[104] I. M. Pereira, T. G. de Senna Carneiro, and E. Figueiredo, "Exploring the Ci/Cd Pipeline in Floss Repositories of Embedded Iot Systems", doi: 10.2139/SSRN.4529908.

[105] G. A. Destro and B. B. N. De França, "Mining Software Repositories for the Characterization of Continuous Integration and Delivery," ACM Int. Conf. Proceeding Ser., pp. 664–669, Oct. 2020, doi: 10.1145/3422392.3422503.

[106] F. Haneef and M. A. Sindhu, "A Reinforcement Learning Based Grammatical Inference Algorithm Using Block-Based Delta Inverse Strategy," IEEE Access, vol. 11, pp. 12525–12535, 2023, doi: 10.1109/ACCESS.2023.3242124.

[107] J. P. Meher and R. Mall, "Machine Learning-based Software Bug Classification through Mining Open Source Repositories," OCIT 2023 - 21st Int. Conf. Inf. Technol. Proc., pp. 17–22, 2023, doi: 10.1109/OCIT59427.2023.10431348.

[108] S. M. Abozeed, M. Y. Elnainay, S. A. Fouad, and M. S. Abougabal, "Software bug prediction employing feature selection and deep learning," 2019 Int. Conf. Adv. Emerg. Comput. Technol. AECT 2019, Feb. 2020, doi: 10.1109/AECT47998.2020.9194215.