e-ISSN:2582-7219

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH IN SCIENCE, ENGINEERING AND TECHNOLOGY

Volume 7, Issue 7, July 2024

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.521

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| Monthly, Peer Reviewed & Referred Journal|

| Volume 7, Issue 7, July 2024 |

| DOI:10.15680/IJMRSET.2024.0707230 |

A Survey on Digital Bug Surveillance Tool

Vishwanath V Murthy, G. Ananya Datt Revankar, Dhanvika H A

Associate Professor, Department of MCA, RNS Institute of Technology, Bengaluru, Karnataka, India

PG Student, Department of MCA, RNS Institute of Technology, Bengaluru, Karnataka, India

PG Student, Department of MCA, RNS Institute of Technology, Bengaluru, Karnataka, India

ABSTRACT: Software companies allocate over 45 percent of their costs to addressing software bugs. A critical step in fixing these bugs is bug triage, which involves assigning the correct developer to a new bug. To reduce the manual effort and time required, text classification techniques are employed for automatic bug triage. This paper tackles the issue of data reduction for bug triage, focusing on minimizing the scale and enhancing the quality of bug data. We integrate instance selection with feature selection to concurrently reduce the data scale on both the bug dimension and the word dimension. To determine the optimal sequence of applying instance selection and feature selection, we extract attributes from historical bug datasets and build a predictive model for new bug datasets. Our empirical investigation encompasses 600,000 bug reports from two large open-source projects, Eclipse and Mozilla. The results demonstrate that our data reduction approach effectively decreases data scale and enhances the accuracy of bug triage. This work offers a method for utilizing data processing techniques to create reduced, high-quality bug data for software development and maintenance.

KEYWORDS: Mozilla, bug

I. INTRODUCTION

Software companies face substantial costs, with over 45 percent attributed to managing software bugs. Central to addressing these challenges is bug triage, a critical process that ensures efficient allocation of developers to resolve new bugs promptly. Automating bug triage through text classification techniques offers a promising solution to reduce manual effort and enhance productivity. However, the sheer volume and complexity of bug data necessitate effective data reduction strategies to streamline processing and improve the quality of triage outcomes.

In this paper, we tackle the problem of data reduction in bug triage, focusing on scaling down the dataset while enhancing its informational content. Our approach integrates instance selection and feature selection methods tailored to reduce both the volume of bug reports and the dimensionality of textual features. To determine the optimal sequence of these reduction techniques, we leverage attributes extracted from historical bug datasets to build predictive models for new bug reports. We empirically evaluate our methodology using extensive datasets comprising 600,000 bug reports from two prominent open-source projects, Eclipse and Mozilla.

Our findings underscore the effectiveness of our data reduction approach in achieving significant improvements in triage accuracy while managing data scalability. By demonstrating the practical application of advanced data processing techniques in software development and maintenance, our work contributes a valuable framework for generating concise, high-quality bug data. This research aims to advance the state-of-the-art in automated bug triage systems, facilitating more efficient software bug resolution and ultimately enhancing software reliability and development efficiency.

II. LITERATURE REVIEW

This study provides a comparative analysis of prominent online bug tracking systems, including Jira, Bugzilla, Redmine, and MantisBT. The evaluation criteria include feature set, integration capabilities, user interface design, scalability, and community support. Jira is noted for its extensive feature set and robust integration with other development tools, making it popular in large enterprises. Bugzilla, with its open-source nature, offers flexibility and extensive customization options, though it may require more configuration effort. Redmine is recognized for its simplicity and ease of use, with strong project management features. MantisBT is highlighted for its lightweight design and straightforward bug tracking functionalities, suitable for smaller teams. The study also considers the impact of these systems on team productivity and bug resolution times, providing insights into how different systems support Agile and Scrum methodologies. The findings suggest that the choice of a bug tracking system should align with the

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| Monthly, Peer Reviewed & Referred Journal|

| Volume 7, Issue 7, July 2024 |

| DOI:10.15680/IJMRSET.2024.0707230 |

project size, complexity, and specific team needs, balancing features, ease of use, and integration capabilities to enhance development workflows effectively.

This paper explores how online bug tracking systems enhance software quality assurance (SQA). By integrating bug tracking into continuous integration and deployment pipelines, systems like Jira, GitHub Issues, and Bugzilla streamline the identification, reporting, and resolution of bugs. The study reviews the role of these systems in fostering better communication among developers, testers, and stakeholders, thereby improving issue resolution times and software quality. Additionally, the paper examines the use of analytics and reporting tools within these systems to track defect trends, assess the severity of bugs, and prioritize fixes effectively. The integration of automated testing frameworks and bug tracking systems is also discussed, highlighting how automated tests feed directly into the bug tracking pipeline, enabling faster detection and resolution of issues. The research underscores the importance of real-time collaboration and feedback loops in enhancing SQA practices, ensuring that bugs are addressed promptly, and software releases meet the desired quality standards. The findings demonstrate that a well-implemented bug tracking system is crucial for maintaining high software quality, reducing development cycles, and increasing team productivity. This review addresses the security challenges associated with online bug tracking systems, focusing on vulnerabilities such as data breaches, unauthorized access, and injection attacks. Systems like Jira, Bugzilla, and MantisBT are scrutinized for their security features, including authentication mechanisms, data encryption, and access controls. The study identifies common security weaknesses, such as insufficient input validation, lack of encryption in data transmission, and inadequate access control policies, which can be exploited by attackers. It also discusses the implications of these vulnerabilities for software development and user data protection. The research highlights best practices for enhancing security in bug tracking systems, such as implementing multi-factor authentication, using secure communication protocols (e.g., HTTPS), and regularly updating the software to patch known vulnerabilities. Additionally, the paper explores the role of security audits and code reviews in identifying and mitigating risks. The findings emphasize the need for ongoing security training for development teams and the implementation of robust security frameworks to safeguard sensitive information. By addressing these challenges, organizations can enhance the security posture of their bug tracking systems, ensuring the confidentiality, integrity, and availability of critical development data.

This research examines the impact of user experience (UX) design on the effectiveness and adoption of online bug tracking systems. Focusing on systems like Jira, Redmine, and GitHub Issues, the study analyzes how interface design influences user productivity, satisfaction, and engagement. Key UX principles, such as simplicity, intuitiveness, and responsiveness, are evaluated in the context of these systems. The paper reviews user feedback and usability studies to identify common pain points and areas for improvement, such as cumbersome navigation, complex workflows, and insufficient customization options. It also explores the integration of user-centric design practices, including user personas, journey mapping, and iterative testing, to create more effective and user-friendly interfaces. The research highlights the benefits of providing contextual help, customizable dashboards, and real-time collaboration features to enhance the user experience. Case studies illustrate how organizations have improved user satisfaction and operational efficiency by redesigning their bug tracking interfaces based on UX best practices. The findings underscore the importance of prioritizing user experience in the development and enhancement of bug tracking systems, ensuring they meet the needs of diverse user groups and promote seamless workflow integration.

This study compares open-source and commercial online bug tracking systems, focusing on factors such as cost, customization, scalability, and support. Open-source systems like Bugzilla and MantisBT are praised for their flexibility, transparency, and zero licensing costs, allowing extensive customization and community-driven improvements. However, they often require significant setup, maintenance, and technical expertise. Commercial systems like Jira, GitHub Issues, and Redmine offer robust features, seamless integration with other tools, and professional support, which can be crucial for larger organizations with complex needs. The paper evaluates the total cost of ownership, including initial setup, ongoing maintenance, and support costs, finding that while open-source systems have lower upfront costs, commercial systems provide greater convenience and reliability. The study also considers the scalability of each type of system, analyzing how well they handle growth in users and issues. User feedback and case studies illustrate the practical advantages and challenges of each approach, guiding organizations in selecting the most suitable bug tracking system based on their specific requirements, resource availability, and long-term goals. The comparison highlights the trade-offs between flexibility and convenience, customization and support, essential for making informed decisions in software development management.

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| Monthly, Peer Reviewed & Referred Journal|

| Volume 7, Issue 7, July 2024 |

| DOI:10.15680/IJMRSET.2024.0707230 |

III. CORE FUNCTIONALITIES OF BUG SURVEILLANCE TOOLS

Effective bug surveillance tools typically offer a range of functionalities to support developers in identifying and resolving issues. Some of these core functionalities include:

1. **Automated Scanning and Detection**
    a. **Description**: Automated scanning allows tools to continuously monitor the codebase or running application for potential bugs and vulnerabilities. This feature is crucial for maintaining high code quality in large projects.
    b. **Advantages**: It reduces the manual effort required for bug detection and ensures consistent monitoring.
2. **Reporting and Notification**
    a. **Description**: Once a bug is detected, the tool generates detailed reports and notifications. These reports often include information such as the bug's severity, location in the code, and suggested fixes.
    b. **Advantages**: Clear and detailed reports help developers understand and prioritize bug fixes efficiently.
3. **Integration with Development Environments**
    a. **Description**: Modern bug surveillance tools integrate seamlessly with popular development environments and continuous integration/continuous deployment (CI/CD) pipelines.
    b. **Advantages**: This integration facilitates real-time bug detection and resolution, fitting smoothly into existing workflows without disrupting the development process.
4. **Collaboration and Documentation**
    a. **Description**: These tools often include features for collaboration and documentation, allowing team members to comment on, update, and track the status of bug reports.
    b. **Advantages**: Enhanced collaboration ensures that all team members are aware of current issues and can contribute to their resolution.

IV. EXISTING SYSTEM

To investigate the relationships in bug data, Sandusky et al. form a bug report network to examine the dependency among bug reports.

Besides studying relationships among bug reports, Hong et al. build a developer social network to examine the collaboration among developers based on the bug data in Mozilla project. This developer social network is helpful to understand the developer community and the project evolution.

By mapping bug priorities to developers, Xuan et al. identify the developer prioritization in open source bug repositories. The developer prioritization can distinguish developers and assist tasks in software maintenance.

To investigate the quality of bug data, Zimmermann et al. design questionnaires to developers and users in three open source projects. Based on the analysis of questionnaires, they characterize what makes a good bug report and train a classifier to identify whether the quality of a bug report should be improved.

Duplicate bug reports weaken the quality of bug data by delaying the cost of handling bugs. To detect duplicate bug reports, Wang et al. design a natural language processing approach by matching the execution information.

DISADVANTAGES OF EXISTING SYSTEM:
Traditional software analysis is not completely suitable for the large-scale and complex data in software repositories.
In traditional software development, new bugs are manually triaged by an expert developer, i.e., a human triager. Due to the large number of daily bugs and the lack of expertise of all the bugs, manual bug triage is expensive in time cost and low in accuracy.

V. PROPOSED SYSTEM

In this paper, we address the problem of data reduction for bug triage, i.e., how to reduce the bug data to save the labor cost of developers and improve the quality to facilitate the process of bug triage.

Data reduction for bug triage aims to build a small-scale and high-quality set of bug data by removing bug reports and words, which are redundant or non-informative.

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| Monthly, Peer Reviewed & Referred Journal|

| Volume 7, Issue 7, July 2024 |

| DOI:10.15680/IJMRSET.2024.0707230 |

In our work, we combine existing techniques of instance selection and feature selection to simultaneously reduce the bug dimension and the word dimension. The reduced bug data contain fewer bug reports and fewer words than the original bug data and provide similar information over the original bug data. We evaluate the reduced bug data according to two criteria: the scale of a data set and the accuracy of bug triage.

In this paper, we propose a predictive model to determine the order of applying instance selection and feature selection. We refer to such determination as prediction for reduction orders.

Drawn on the experiences in software metrics,1 we extract the attributes from historical bug data sets. Then, we train a binary classifier on bug data sets with extracted attributes and predict the order of applying instance selection and feature selection for a new bug data set.

ADVANTAGES OF PROPOSED SYSTEM:

Experimental results show that applying the instance selection technique to the data set can reduce bug reports but the accuracy of bug triage may be decreased. Applying the feature selection technique can reduce words in the bug data and the accuracy can be increased. Meanwhile, combining both techniques can increase the accuracy, as well as reduce bug reports and words.

Based on the attributes from historical bug data sets, our predictive model can provide the accuracy of 71.8 percent for predicting the reduction order.

We present the problem of data reduction for bug triage. This problem aims to augment the data set of bug triage in two aspects, namely a) to simultaneously reduce the scales of the bug dimension and the word dimension and b) to improve the accuracy of bug triage.We propose a combination approach to addressing the problem of data reduction. This can be viewed as an application of instance selection and feature selection in bug repositories.We build a binary classifier to predict the order of applying instance selection and feature selection. To our knowledge, the order of applying instance selection and feature selection has not been investigated in related domains.

System Architecture

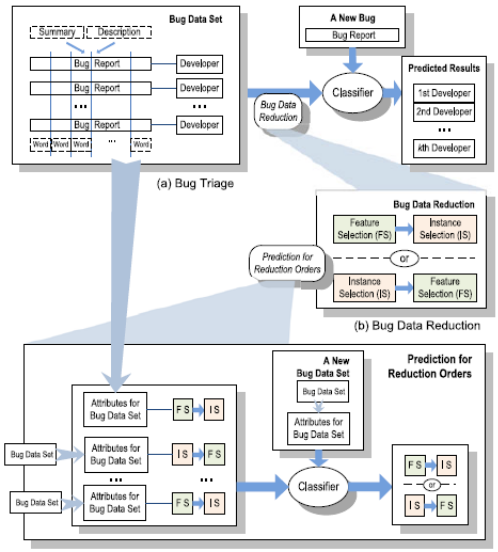

Fig 1. System Architecture

VI. MODULE DESCRIPTION

Dataset Collection:

To collect and/or retrieve data about activities, results, context and other factors. It is important to consider the type of information it want to gather from your participants and the ways you will analyze that information. The data set corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| Monthly, Peer Reviewed & Referred Journal|

| Volume 7, Issue 7, July 2024 |

| DOI:10.15680/IJMRSET.2024.0707230 |

table represents a particular variable. after collecting the data to store the Database.

Preprocessing Method:
Data Preprocessing or Data cleaning, Data is cleansed through processes such as filling in missing values, smoothing the noisy data, or resolving the inconsistencies in the data. And also used to removing the unwanted data. Commonly used as a preliminary data mining practice, data preprocessing transforms the data into a format that will be more easily and effectively processed for the purpose of the user.

Feature Selection/ Instance Selection:
The combination of instance selection and feature selection to generate a reduced bug data set. We replace the original data set with the reduced data set for bug triage. Instance selection is a technique to reduce the number of instances by removing noisy and redundant instances. By removing uninformative words, feature selection improves the accuracy of bug triage. It recover the accuracy loss by instance selection.

Bug Data Reduction:
The data set can reduce bug reports but the accuracy of bug triage may be decreased. It improves the accuracy of bug triage. It tends to remove these words to reduce the computation for bug triage. The bug data reduction to reduce the scale and to improve the quality of data in bug repositories. It reducing duplicate and noisy bug reports to decrease the number of historical bugs.

Performance Evaluation:
In this Performance evaluation, algorithm can provide a reduced data set by removing non-representative instances. The quality of bug triage can be measured with the accuracy of bug triage. to reduce noise and redundancy in bug data sets.

Core Functionalities of Bug Surveillance Tools

Effective bug surveillance tools typically offer a range of functionalities to support developers in identifying and resolving issues. Some of these core functionalities include:

1. **Automated Scanning and Detection**
    a. **Description**: Automated scanning allows tools to continuously monitor the codebase or running application for potential bugs and vulnerabilities. This feature is crucial for maintaining high code quality in large projects.
    b. **Advantages**: It reduces the manual effort required for bug detection and ensures consistent monitoring.
2. **Reporting and Notification**
    a. **Description**: Once a bug is detected, the tool generates detailed reports and notifications. These reports often include information such as the bug's severity, location in the code, and suggested fixes.
    b. **Advantages**: Clear and detailed reports help developers understand and prioritize bug fixes efficiently.
3. **Integration with Development Environments**
    a. **Description**: Modern bug surveillance tools integrate seamlessly with popular development environments and continuous integration/continuous deployment (CI/CD) pipelines.
    b. **Advantages**: This integration facilitates real-time bug detection and resolution, fitting smoothly into existing workflows without disrupting the development process.
4. **Collaboration and Documentation**
    a. **Description**: These tools often include features for collaboration and documentation, allowing team members to comment on, update, and track the status of bug reports.
    b. **Advantages**: Enhanced collaboration ensures that all team members are aware of current issues and can contribute to their resolution.

VII. BENEFITS OF USING DIGITAL BUG SURVEILLANCE TOOLS

Implementing digital bug surveillance tools offers numerous benefits to development teams and organizations:

1. **Improved Software Quality**
    a. **Description**: By continuously monitoring and detecting bugs, these tools help maintain high standards of code quality and reliability.
    b. **Impact**: Fewer bugs in production lead to a better user experience and reduced maintenance costs.
2. **Cost and Time Efficiency**

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| Monthly, Peer Reviewed & Referred Journal|

| Volume 7, Issue 7, July 2024 |

| DOI:10.15680/IJMRSET.2024.0707230 |

a. **Description**: Early detection and resolution of bugs can significantly reduce the time and cost associated with fixing issues later in the development cycle.
b. **Impact**: Development teams can allocate resources more effectively and meet project deadlines more consistently.

3. **Enhanced Security**
   a. **Description**: Many bug surveillance tools also focus on identifying security vulnerabilities, ensuring that applications are robust against potential attacks.
   b. **Impact**: Protecting sensitive data and maintaining user trust by preventing security breaches.
4. **Streamlined Development Processes**
   a. **Description**: With automated detection and detailed reporting, developers can focus more on writing quality code rather than spending time on manual bug detection.
   b. **Impact**: Increased productivity and a more streamlined workflow.

VIII. CHALLENGES IN IMPLEMENTING BUG SURVEILLANCE TOOLS

Despite their benefits, digital bug surveillance tools also present certain challenges:

1. **False Positives and Negatives**
   a. **Description**: Tools may sometimes report non-existent bugs (false positives) or miss actual bugs (false negatives), leading to potential confusion and wasted effort.
   b. **Solution**: Continuous calibration and refinement of tool settings can help minimize these issues.
2. **Integration and Compatibility**
   a. **Description**: Integrating new tools with existing development environments and workflows can be complex and time-consuming.
   b. **Solution**: Choosing tools with robust integration capabilities and seeking vendor support during the implementation phase can ease this process.
3. **Performance Overhead**
   a. **Description**: Some tools may introduce performance overhead, slowing down the development and testing processes.
   b. **Solution**: Balancing the depth of analysis with performance impact and running intensive scans during off-peak hours can mitigate this challenge.

IX. FUTURE TRENDS IN DIGITAL BUG SURVEILLANCE

The future of digital bug surveillance is likely to be shaped by advancements in artificial intelligence and machine learning. These technologies can enhance the accuracy and efficiency of bug detection by learning from vast datasets of code and bug reports. Additionally, the increasing adoption of DevOps and CI/CD practices will drive the development of more integrated and automated bug surveillance solutions, further streamlining the development lifecycle.

X. TYPES OF DIGITAL BUG SURVEILLANCE TOOLS

Digital bug surveillance tools can be broadly categorized into several types based on their functionality and the stage of development they target:

1. **Static Analysis Tools**
   a. **Description**: These tools analyze the source code without executing it. They look for potential bugs, vulnerabilities, and code smells by examining the code structure and syntax.
   b. **Examples**: SonarQube, ESLint, Pylint.
   c. **Benefits**: They can identify issues early in the development process, reducing the cost and effort required to fix bugs later.
2. **Dynamic Analysis Tools**
   a. **Description**: Unlike static analysis tools, dynamic analysis tools examine the program during its execution. They monitor the behavior of the software in real-time to detect bugs that might not be evident from the code alone.
   b. **Examples**: Valgrind, Dynatrace.
   c. **Benefits**: These tools can detect runtime errors, memory leaks, and performance issues that static analysis might miss.

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| Monthly, Peer Reviewed & Referred Journal|

| Volume 7, Issue 7, July 2024 |

| DOI:10.15680/IJMRSET.2024.0707230 |

3. **Interactive Application Security Testing (IAST) Tools**
    a. **Description**: IAST tools combine static and dynamic analysis by monitoring applications in real-time while they are being tested manually or automatically.
    b. **Examples**: Contrast Security, Seeker.
    c. **Benefits**: They provide a comprehensive view of potential vulnerabilities and bugs by combining multiple analysis techniques.

4. **Bug Tracking Systems**
    a. **Description**: These systems are designed to record, manage, and track bug reports from detection to resolution. They often integrate with other development tools to streamline the bug management process.
    b. **Examples**: JIRA, Bugzilla, Trello.
    c. **Benefits**: They ensure that bugs are documented, assigned, and resolved systematically, improving communication and collaboration among development teams.

XI. CORE FUNCTIONALITIES OF BUG SURVEILLANCE TOOLS

Effective bug surveillance tools typically offer a range of functionalities to support developers in identifying and resolving issues. Some of these core functionalities include:

1. **Automated Scanning and Detection**
    a. **Description**: Automated scanning allows tools to continuously monitor the codebase or running application for potential bugs and vulnerabilities. This feature is crucial for maintaining high code quality in large projects.
    b. **Advantages**: It reduces the manual effort required for bug detection and ensures consistent monitoring.

2. **Reporting and Notification**
    a. **Description**: Once a bug is detected, the tool generates detailed reports and notifications. These reports often include information such as the bug's severity, location in the code, and suggested fixes.
    b. **Advantages**: Clear and detailed reports help developers understand and prioritize bug fixes efficiently.

3. **Integration with Development Environments**
    a. **Description**: Modern bug surveillance tools integrate seamlessly with popular development environments and continuous integration/continuous deployment (CI/CD) pipelines.
    b. **Advantages**: This integration facilitates real-time bug detection and resolution, fitting smoothly into existing workflows without disrupting the development process.

4. **Collaboration and Documentation**
    a. **Description**: These tools often include features for collaboration and documentation, allowing team members to comment on, update, and track the status of bug reports.
    b. **Advantages**: Enhanced collaboration ensures that all team members are aware of current issues and can contribute to their resolution.

XII. RESULT

Bug triage is an expensive step of software maintenance in both labor cost and time cost. In this paper, we combine feature selection with instance selection to reduce the scale of bug data sets as well as improve the data quality. To determine the order of applying instance selection and feature selection for a new bug data set, we extract attributes of each bug data set and train a predictive model based on historical data sets. We empirically investigate the data reduction for bug triage in bug repositories of two large open-source projects, namely Eclipse and Mozilla.

XIII. CONCLUSION

In conclusion, our study presents a novel approach to optimizing bug triage through a combination of feature selection and instance selection techniques. By applying these methods, we effectively reduce the scale of bug data sets while enhancing data quality, thereby mitigating the substantial labor and time costs associated with bug triage in software maintenance. Our empirical investigation on bug repositories from Eclipse and Mozilla demonstrates the feasibility and benefits of our approach in real-world scenarios. Moving forward, our future work aims to further refine our data

| ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 7.521| Monthly, Peer Reviewed & Referred Journal|

| Volume 7, Issue 7, July 2024 |

| DOI:10.15680/IJMRSET.2024.0707230 |

reduction techniques and explore strategies for improving the preparation of high-quality bug data sets tailored to specific software domains. Additionally, we plan to delve deeper into understanding the relationships between bug data set attributes and optimal reduction orders, offering insights that could significantly advance the efficiency and effectiveness of bug triage processes in software development and maintenance contexts.

REFERENCES

[1] 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.
[2] S. Artzi, A. Kie_zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, "Finding bugs in web applications using dynamic test generation and explicit-state model checking," IEEE Softw., vol. 36, no. 4, pp. 474–494, Jul./Aug. 2010.
[3] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," ACM Trans. Soft. Eng. Methodol., vol. 20, no. 3, article 10, Aug. 2011.
[4] C. C. Aggarwal and P. Zhao, "Towards graphical models for text processing," Knowl. Inform. Syst., vol. 36, no. 1, pp. 1–21, 2013.
[5] Bugzilla, (2014). [Online]. Avaialble: http://bugzilla.org/
[6] K. Balog, L. Azzopardi, and M. de Rijke, "Formal models for expert finding in enterprise corpora," in Proc. 29th Annu. Int. ACM SIGIR Conf. Res. Develop. Inform. Retrieval, Aug. 2006, pp. 43–50.
[7] P. S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," IEEE Trans. Knowl. Data Eng., vol. 24, no. 6, pp. 1146–1150, Jun. 2012.
[8] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," Data Mining Knowl. Discovery, vol. 6, no. 2, pp. 153–172, Apr. 2002.
[9] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: Improving cooperation between developers and users," in Proc. ACM Conf. Comput. Supported Cooperative Work, Feb. 2010, pp. 301–310.
[10] V. Bol_on-Canedo, N. S_anchez-Maro~no, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," Knowl. Inform. Syst., vol. 34, no. 3, pp. 483–519, 2013.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING AND TECHNOLOGY

| Mobile No: +91-6381907438 | Whatsapp: +91-6381907438 | ijmrset@gmail.com |

www.ijmrset.com