Hindawi
Scientific Programming
Volume 2022, Article ID 2302027, 10 pages
https://doi.org/10.1155/2022/2302027

Research Article

An Optimized Systematic Approach to Identify Bugs in Cloud-Based Software

Shanmugasundaram Marappan,[1] **Archana Kollu**,[2] **Ismail Keshta**,[3] **Shehab Mohamed Beram**,[4] **Sahil Bhende**,[5] **and Karthikeyan Kaliyaperumal**[6]

[1]*Department of Computer Science, College of Computer Science & Information Technology, Jazan University, Jizan, Saudi Arabia*
[2]*Department of Computer Engineering, Pimpri Chinchwad College of Engineering and Research Ravet, Pune 412101, India*
[3]*Computer Science and Information Systems Department, College of Applied Sciences, AlMaarefa University, Riyadh, Saudi Arabia*
[4]*Research Centre for Human-Machine Collaboration (HUMAC), Department of Computing and Information Systems, School of Engineering and Technology, Sunway University, Kuala Lumpur, Malaysia*
[5]*UBS Business Solutions (India) Pvt. Ltd., Nanakaramguda, India*
[6]*IT @ IoT-HH Campus, Ambo University, Ambo, Ethiopia*

Correspondence should be addressed to Karthikeyan Kaliyaperumal; karthikeyan@ambou.edu.et

Received 19 July 2022; Revised 10 August 2022; Accepted 22 August 2022; Published 15 September 2022

Academic Editor: Punit Gupta

The resolution of a software bug depends on the severity of the defect report. Open-source software defect tracking solutions have taken over as the principal means of processing enormous amounts of defect information data due to the ongoing increase in software scale. Dealing with software faults requires analyzing the implications of defect report severity in the data warehouse. Thus, the authors have proposed an optimized systematic approach through the research and analysis of Bugzilla defect tracking system data in this study, where it is found that the attribute characteristics of different projects are quite different and the statistical features of the repair rate, resolution time, developers, components, and other attributes are consistent. This technique, therefore, assumes that a rise in the severity of software defect reports will result in a rise in the defect repair rate and that the severity is normally based on the severity distribution of various components and projects. According to the study's findings, developers hold the most defects when the repair rate is low and the defect resolution time is shortest.

1. Introduction

The major approach for ensuring software quality is severity analysis of software defect reports, which is also an important signal for assessing, sorting, and assigning software defect reports. Software errors rise in tandem with the volume and complexity of software development. As a result, software defect correction has become critical in the creation and maintenance of software. Many manufacturers, such as Bugzilla and JIRA, utilize defect tracking systems like Bugzilla and JIRA to monitor and repair software bugs in a fast, accurate, and effective way. At the same time, software users and developers will utilize software defect reports to explain different aspects of software faults [1]. Atlassian, an Australian company, developed the tool JIRA. It is used for project management as well as the bug and issue tracking. The Mozilla Foundation developed a web-based bug tracking program known as Bugzilla. Tracking Mozilla's initiatives, such as the Firefox web browser, is done using the application. Defect tracking systems like Bugzilla and JIRA are utilized by numerous manufacturers to quickly, precisely, and effectively monitor and fix software defects. Software defect findings are employed by software users and developers to describe various parts of software flaws at the

same time. The only difference that may favor Bugzilla over the JIRA is that Bugzilla is preferable for small companies where the start-up budget is low, while JIRA is a large ship that fulfills every useful function necessary for the software tracking system but it is complex in nature and, therefore, used for complex projects.

Errors in the system's setup settings are known as configuration errors. Incorrect system states, such as system breakdowns, performance declines, and other unexpected system behavior, are frequent manifestations of configuration problems.

The following categories can be used to group configuration challenges to tackle the errors according to their characteristics and lack of tool support for troubleshooting and tolerance: systems and configurations are both complicated, making it challenging to guarantee accuracy. Log messages are frequently missing, and users lack troubleshooting assistance, traditional fault tolerance, and recovery methods are of little assistance. The severity of software defect reports may indicate the extent to that faults have an impact on software development and operation [2]. As a result, the severity of the fault will have a direct impact on the priority of defect repair and staff selection [3]. Defect assignment has become a necessary task in software defect repair as a result of long-term data gathering that has resulted in a vast defect report warehouse. The score for flaws, during the dispatching step, is usually completed by the developer. However, while sending the defect to the fixer, the severity of the software defect report should be used to establish the defect's priority, and the degree of impact on software development and operation should be stated [4]. As a result, software defect severity prediction is a crucial problem in software defect correction [5].

At present, artificial methods are used to predict the severity of software defects, and there are the following core problems:

(a) Data volume problem: there are many defect reports submitted through the defect tracking system. For example, Mozilla receives 135 defect reports per day [6]. Therefore, the manual processing method will significantly burden defect dispatchers and processing personnel.

(b) Accuracy problem: due to the differences in the professional level and experience of the defect reporters, the selection of defect severity varies significantly during the submission process of the defect report; so many reporters choose the default severity during the report submission process. Rating, or submitting an incorrect severity rating, affects the severity prediction of defect reports.

Therefore, the software defect severity prediction is a crucial problem in software defect correction, and the core problems as discussed above mainly include the data volume concern and the accuracy problem. Hence, due to the presence of major concerns, this paper comprehensively analyzes the relationship between the severity of software defect reports and the report submitters, resolution time, repair rate, components, and products by comparing the defect report data of Mozilla products and Eclipse products in the Bugzilla software defect tracking system relationship. As the outcome of this study, it can be seen that by using machine learning to analyze the data from software defect reports and assess their severity, software defects can be classified more accurately, their repair time for serious flaws can be accelerated, and their repair rates can be increased to ensure the effectiveness of software version iteration.

2. Related Works

Machine learning methods are often used in existing research to anticipate the severity of new fault reports. To assist developers in determining the severity of a particular fault, Menzies and Marcus created the SEVERIS (Severity Issue Assessment) technique [7]. To anticipate the severity of problems, the SEVERIS method uses text mining and combines it with the RIPPER rule learning algorithm. The findings reveal that text mining and machine learning methods are used to forecast fault severity. The authors investigated whether problem severity could be predicted by evaluating the text description of defect reports using text mining and verifying it using the NB method [8]. Then, in 2011, they extended the previous work by conducting tests to examine the influence of four machine learning algorithms, including NB, MNB, KNN, and SVM, on predicting the severity (severe or not) of software flaws [9]. The findings reveal that the MNB algorithm outperforms other algorithms in terms of performance and efficiency. To forecast the severity of fresh defect reports, the authors used information retrieval methods, particularly the BM25-based text similarity algorithm and the KNN algorithm. This technique outperforms the SEVERIS algorithm in terms of prediction accuracy the author has created and applied CP, a novel text classification approach, to estimate the severity of defect reports in 2015 [10].

Data mining methods are then used to predict false positives with a lower false-positive rate. Literature has proposed a formal privacy analysis method based on static taint analysis to simplify the design and review process of high-level privacy properties of software architectures [11]. Literature obtained second order database access information flow through string analysis of database, filename, and session variables, which can detect second order SQLIA and multiorder vulnerabilities the proposed PHP (hypertext preprocessor) static analysis framework, which can automatically parse the standard features of dynamic languages, independently define the value and heap analysis of dynamic languages, and automatically combine them to find actual security vulnerabilities [12]. The positive rate is low. Literature developed a highly modular static analysis framework SCALA-AM, which achieves modularity by separating operational semantics, can be extended to support multiple languages, including numerous machine abstractions, and can be implemented in different pollution analysis on-demand programs [13]. Literature has innovatively grown the data flow analysis framework into a sparse form. They used the light optimization method to eliminate the unrelated

taint propagation in static taint analysis to optimize the time and space overhead and did not realize the aliasing between aliases: substantial updates and precise computation of conserved subdomains [14]. Literature has proposed and implemented a taint analysis method based on offline indexing of execution traces. Taking bytes as the granularity, the problem of taint loss caused by instant translation execution was described and solved for the first time [15]. Literature has implemented a novel approach to static taint analysis that is accurate and scalable, treating taint analysis as a demand-driven problem that only computes vulnerable information flows instead of computing complete data flow solutions. Requirement-driven tracking and computation of weak information flow improve the accuracy and efficiency of the analysis while scaling the analysis to large codebases [16].

Dynamic taint analysis detects whether the data comes from an untrusted external input immediately when the program is executed and sees whether the data can be propagated from the taint source to the taint convergence point by monitoring the spread of the taint data of the program in the system program in real-time. Literature developed a dynamic integrity contamination analysis model for Java, which addresses the imperfect sanitization problem in an in-depth approach, combined with anticipatory and retrospective measures, which is a forward-looking recognition of sanitization results [17]. The case of retrospective contamination is wholly sterilized, which well avoids the appearance of false positives, but the model scalability needs to be strengthened and developed. As per the study, the authors have developed a binary dynamic taint analysis tool called Sword DTA to maximize the detection of software vulnerabilities. This tool is suitable for both hardware and software. It finds vulnerabilities through vulnerability modeling and taints checking. It has a good effect on buffer overflow, integer overflow, division by zero, UAF (used after free), and other vulnerabilities. Still, it has a good impact on different types of vulnerabilities [18]. Moreover, there are numerous approaches available for ensuring software quality in software development, such as quality management approach, formal technical analysis, multi testing strategy, effective software engineering technology, measurement, and reporting mechanism. Several examples are CloudQA, Akamai CloudTest AppPerfect, CloudSleuth, Nessus, Wire shark, etc. Along with this, software quality assurance provides several benefits and limitations such as generating standard quality software, saving time and cost, low maintenance for a longer time, and improved process creation, and the limitations include incorporating multiple resources, hiring more employees for quality maintenance, etc.

The exposure is temporarily powerless, and the path coverage is not high. Literature has provided a platform-independent dynamic taint analysis for JavaScript, encoding the taint propagation process as abstract machine instructions and executing these instructions to get a taint flow report about the applicant but without tracking the implicit flow; moreover, the taint propagation strategy is not flexible enough and does not support user-defined operations.

The contributions of this paper mainly include the following aspects:

(1) Introduce and analyze the statistical characteristics of the bug report data warehouse of the Bugzilla bug tracking system in the Mozilla project and Eclipse project.

(2) The probability that developers fix bugs decreases with the increase in the number of bug reports.

(3) Through the analysis of the severity of the defect, reports of the two projects, the repair rate, repair time, repairers, and other factors, the correlation of the data between the projects is obtained.

3. Overview of Bugzilla, an Open-Source Bug Tracking System

Mozilla Corporation created and maintains Bugzilla, an open-source bug tracking system. The plan creates a comprehensive defect tracking system for users by recording and tracking software defects, as well as providing services such as defect submission, assignment, repair, and closure across the entire software. The bug tracking tool Bugzilla enables developers to monitor still open issues on their products. It uses MySQL and is coded in perl. Although Bugzilla is a tool for recording defects, it can also be used to manage tests and may thus be readily integrated with other test case management platforms like quality center, etc. It is simple to use and get to Bugzilla. Criteria for finding bugs are extremely effective. It is helpful to rename and save each bug search thread. The user interface for listing issues, particularly the ability to modify columns and edit searches, is quite helpful. The navigation to various screens in Bugzilla can be sluggish at times, with less customized options. Additionally, the UI and UX of Bugzilla appear to be out of date and in need of an update. Furthermore, there are not many plugins available, which is a problem.

Users' software defect reports have a complete life cycle, as shown in Figure 1. When the user confirms and submits the defect report, the report status is set to the uncertified state (unconfirmed); when the developer accepts the defect, it becomes the certified state (confirmed); if the developer can directly solve the fault, the defect report will be changed to the resolved state (resolved); if it cannot be decided, it will enter the dispatching process, and the defect state will be adjusted to be repairing (in progress). Repair; after the repair is completed and confirmed, it reaches the verified state (verified); when the developer approves the solution of the defect report to be correct, the report status becomes resolved, and the information is closed at the same time; when the message needs to be repeated when opened, the reporting status is adjusted to reopened [19].

In the dispatching process, the severity attribute of the defect (severity) is mainly used to determine the defect repair sequence and repair personnel. Suppose the data flow analysis is performed directly and simply with each pair of source and sink. In that case, it will waste computing resources because there is no executable path between many sources and sinks.

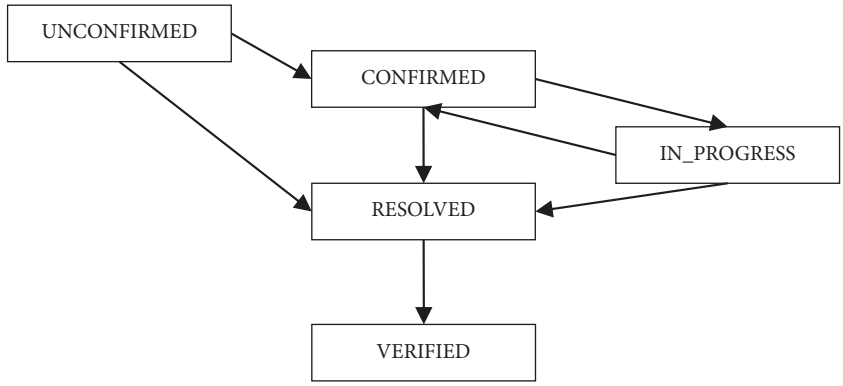

Figure 1: Software defect tracking process diagram [9].

According to the location of the source and sink in Web applications and the characteristics of cross-file propagation, this paper proposes a multimatching preprocessing method, which first excludes source and sink pairs that cannot have executable paths before performing complex data flow analysis. This reduces unnecessary resource consumption.

The multiple matching preprocessing methods include four submethods, which are given as follows:

(a) Exact method matching

(b) Exact Request information matching

(c) Session information matches

(d) Method parameters match

All these methods are classified independently as follows: (1) exact method matching, the same method of judging whether source and sink are located in the same file; (2) exact request information matching, judging whether source and sink can return values according to the get method of the exact request information matches with set method parameters; (3) session information matches, judges whether source and sink can be matched through the attributes of the session object; and (4) method parameters match, judges whether source is passed to the method of sink as a parameter of the method call statement. Currently, match methods can support the constant propagation of request and session object properties.

In addition, for a successfully matched source and sink pair, if the source is not native, the native source mapped by the source needs to be checked with the sink.

3.1. Mozilla Project Defect Tracking System. When a defect report is submitted, the necessary attributes are generally marked. For example, the defect report numbered 35 includes the status, author, time, defect description, severity level after the account is submitted, and detailed attribute information contained in each part. The report ID is 35, and the title is "navigator does not free preference hash table when exiting." The report was written and last updated on March 1, 2018. The product targeted by the information is MozillaClassic, the component is XFE, and the severity level is "minor." According to Bugzilla's fix log, the report was assigned only once between 1998 and 2018 to Chris McAfee, who revised the account on December 12, 1998, with "verified" status [20].

3.2. Eclipse Project Defect Tracking System. When a defect report is submitted, the necessary attributes are generally marked, for example, the defect report information of Eclipse with ID 30. The report is titled "[CVS Core] server.CVS ignore files should be considered by the client." A report authored by Jean–Michel Lemieux and last updated May 6, 2009. The product targeted by the information is a platform, the component is a team, and the severity level is average. According to Bugzilla's fix record, the report was fixed by nine people between 2001 and 2009, the report authors assigned the object Platform VCm-Inbox, and the fix was completed on May 6, 2009 [21].

4. Impact of Project Attributes on the Severity of Defect Reports

In collecting user reports, the Bugzilla defect report processing system will mark the content of the information with a severity rating, which determines the processing priority of the word and affects the defect processing time and repair rate, and other characteristics. The severity of the defect report is divided into seven levels, from high to low, blocked, critical, major, standard, minor, trivial, and enhancement, where every day is the default level when the defect report is submitted. It is also the most selected by the user in presenting the information. Mozilla and Eclipse projects' attributes accounted for 73.92% and 67.02%, respectively. The life cycle of developing software inevitably includes defects. No code is ever written effectively on the first try. It is important to find, note, and fix bugs, irregularities, and errors. Therefore, extensive testing and optimization are needed to provide a strong software product. A variety of bugs include which can tamper with the functioning of the program called functional bugs, or which can disrupt the normal workflow as workflow bugs, similarly, they may include logical bugs, bugs in unit level and system level integration, etc. Depending upon their severity levels, as can be seen, defect reports with higher severity have a bigger impact on software functionality. Developers will, therefore, pay more attention to these software problems, and as a

Table 1: Bug reports [7].

Severity	Quantity	% of all defect reports	Number of resolved	Proportion of reports resolved at this level/%
Blocker	15576	2.14	14458	99.12
Critical	86194	7.9	79239	95.18
Major	85594	6.84	78874	95.45
Minor	46231	4.79	44136	88.84
Normal	895328	84.27	796 968	88.99
Trivial	19615	2.56	19316	93.12
Enhancement	66537	6.5	52100	77.87

result, their rate of rectification will increase. On the other hand, less serious software faults are less likely to be fixed and have little impact on performance.

4.1. Mozilla Project Bug Report Properties and Severity Levels

4.1.1. Relationship Analysis

(1) The impact of defect severity on the pace of defect repair: the total number of defect reports with an average severity level recorded in this study in the Mozilla project is 885,328, accounting for more than 73% of all defect reports. The blocker-level defect reported with the greatest severity, on the other hand, is just 1.14%. Table 1 shows the percentage of bug reports with severity levels.

Defect reports of higher severity have a greater influence on software functionality, as can be observed. As a result, developers will pay greater attention to such software flaws, and their rate of correction will rise as a result. Software flaws of lesser severity, on the other hand, have minimal influence on software performance and have a low repair rate.

(2) The impact of the severity of the defect report on the defect resolution time because the resolution time is one of the essential characteristics of the defect report, which states the total time taken by the software to resolve the errors/bugs. Therefore, this experiment compares and analyzes the average resolution time of bug reports of different severity in the Mozilla project, and the results are shown in Figure 2 with Table 2. The average resolution time of all defect reports collected in the experiment was 772 days; the resolution time of the blocker-level defect report with the highest severity was 1,083 days; the resolution time of the enhancement-level defect with the lowest severity was 1,240 days; the resolution time for defects with an ordinary degree is shorter than the average resolution time because the defect with the ordinary degree can be resolved quickly compared to the defects with the higher severity degree; and for defects with a non-normal severity, the resolution time is longer than the average resolution time.

It can be seen that software defects with an average severity level will be solved by developers first; defects with higher severity levels are more challenging to solve, resulting in longer resolution time; defects

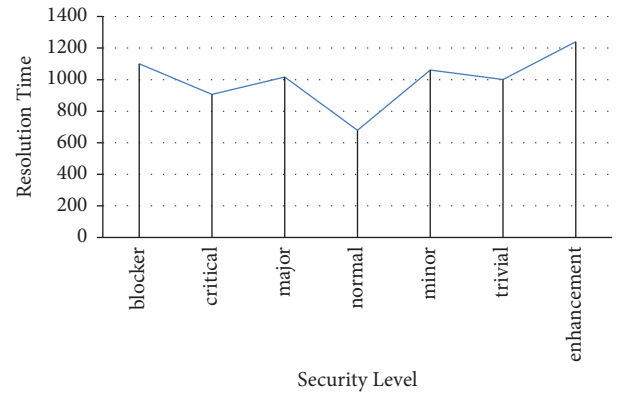

Figure 2: The resolution time chart of defect reports of different severity levels.

Table 2: The resolution time chart of defect reports of different severity levels [8].

Blocker	Critical	Major	Normal	Minor	Trivial	Enhancement
1100	907	1016	679	1060	1001	1240

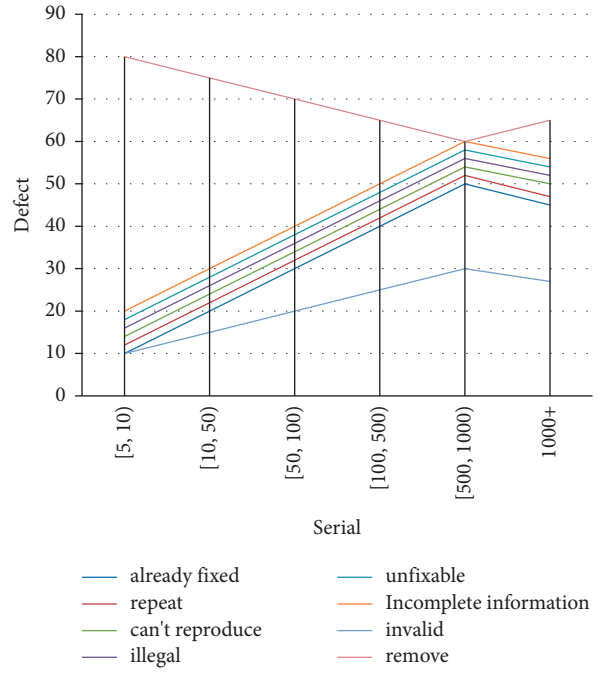

Figure 3: Proportion of the developer's solution to the defect.

Table 3: Proportion of the developer's solution to the defect.

Serial	Already fixed	Repeat	Can't reproduce	Illegal	Unfixable	Incomplete information	Invalid	Remove
[5, 10)	10	12	14	16	18	20	10	80
[10, 50)	20	22	24	26	28	30	15	75
[50, 100)	30	32	34	36	38	40	20	70
[100, 500)	40	42	44	46	48	50	25	65
[500, 1000)	50	52	54	56	58	60	30	60
1000+	45	47	50	52	54	56	27	65

with lower severity levels are not given much attention high, resulting in a longer resolution time.

(3) The impact of developer support on the defect repair rate

Through the analysis of software defect reports of Mozilla products, it is found that there is a close relationship between the holder of the report (developer) and the repair rate of the report. There are 8,497 developers in this project, of which 3,505 hold more than 5 bug fixes. This paper divides these developers into 6 categories, 0~5, 5~10, 10~50, 50–100, 100–500, 500–1,000, and more than 1,000 defect reports. Since the repair rate of developers with less than 5 defect repair jobs is not high, this part of the data are not compared in this experiment.

This experiment analyzes the fixed rate of bug reports held by Mozilla project developers. As shown in Figure 3 of Table 3, with the increase in the number of defect reports held by developers, the reported repair rate dropped significantly from 77.22% to 52.40%; the solution duplication rate increased from 5.05% to 18.31%, and the irreproducible rate increased from 5.81% It increased to 18.02%, and the changes in attributes such as illegal, irreparable, and incomplete information were all within 10%; from the perspective of the severity of reports, the reporting ratio of critical and significant levels held by developers increased by 1.16 times (from 11.51% rose to 24.91%), the repair rate is constantly decreasing, the solution repetition rate and the irreproducible rate increase slightly, and the attributes of illegality, irreparability, and incomplete information also increase somewhat with the overall curve.

In the report distribution process, the higher the repair rate, the higher the probability of obtaining the report, and the higher the probability of getting the report with greater difficulty. As a result, developers with more words have lower report repair rates.

(4) The impact of defect report severity on the repair rate of software components by analyzing the severity distribution of 16 members with more than 10,000 defect reports in Mozilla. It is found that among the defect reports for each element, the number of defect reports with a severity level of standard is the largest by counting the distribution of bug reports for different components of Mozilla. Mozilla has 1,936

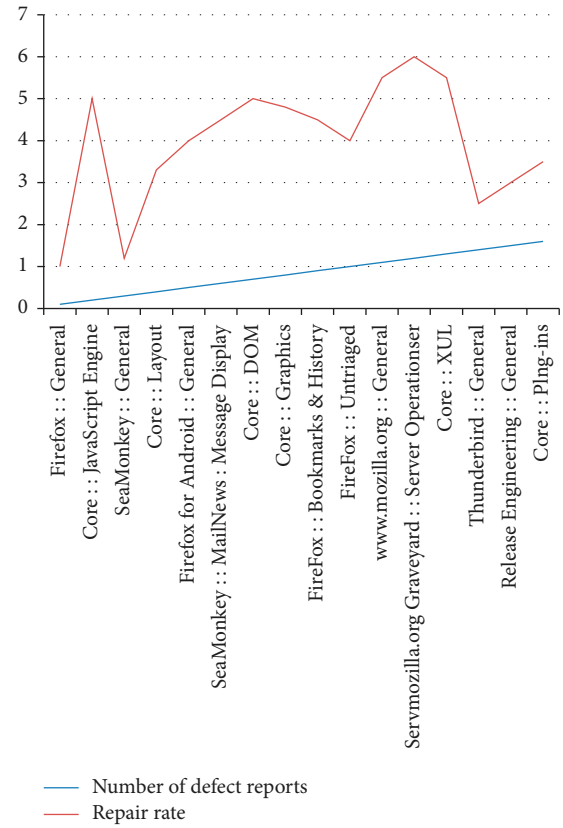

Figure 4: Distribution of the number of defect reports and repair rates for different components.

parts covered in bug reports. As shown in Figure 4 to Table 4, the repair rate of defect reports of various features varies greatly. For example, the repair rate of the server operations component is the highest, reaching 85.94%; the repair rate of the untrained element is the lowest, only 0.84%.

According to the severity distribution of defect reports corresponding to other members, there are significant differences in the severity levels of defect reports for various parts. The main reason is that the proportion of words with different severity levels fluctuates wildly in each element. For example, in the server operations component, the blocker level reports account for the most significant proportion, resulting in the highest repair rate; in the untrained part, the average level reports account for the essential proportion, resulting in the lowest repair rate. In addition, it shows that the repair rate will increase with the increase in severity.

Table 4: Distribution of the number of defect reports and repair rates for different components.

Serial	Number of defect reports	Repair rate
Firefox::General	0.1	1
Core::JavaScript engine	0.2	5
SeaMonkey::General	0.3	1.2
Core::Layout	0.4	3.3
Firefox for Android::General	0.5	4
SeaMonkey::MailNews:Message display	0.6	4.5
Core::DOM	0.7	5
Core::Graphics	0.8	4.8
FireFox::Bookmarks & history	0.9	4.5
FireFox::Untriaged	1	4
www.mozilla.org::General	1.1	5.5
Servmozilla.org Graveyard::Server operationser	1.2	6
Core::XUL	1.3	5.5
Thunderbird::General	1.4	2.5
Release Engineering::General	1.5	3
Core::Plng-ins	1.6	3.5

Table 5: Distribution of defect reports of different severity levels in eclipse projects.

Severity	Quantity	% of all defect reports	Number of resolved	Proportion of reports resolved at this level/%
Blocker	14559	2.14	14440	99.22
Critical	83030	7.88	79136	95.35
Major	82512	7.84	78808	95.56
Minor	46158	4.79	41086	88.87
Normal	891 298	83.93	75155	88.88
Trivial	19603	2.56	17408	93.84
Enhancement	66439	6.49	51152	78.21

4.2. Eclipse Project Defect Report Attribute and Severity Level Relationship Analysis

(1) Influence of defect report severity on repair rate In Eclipse projects, the number of normal-level reports with default severity of defect reports accounted for 67.02% of the total number of reports. Only 1.54% of the blocker-level bug reports with the highest severity were reported. Table 5 shows the resolution of defect reports with different severity levels. The blocker level and critical level defects with the highest severity level are resolved by more than 95%.

The number of cases in which software defects of varying severity are resolved, it can be seen from the quantitative distribution that the severity level is an enhancement level of the bug reports that were fixed, the highest percentage was fixed low, about 60%; the highest proportion of defect reports is invalid, reaching 18%. It can be seen that the repair rate of defect reports with higher severity is more minor than the quality of software products and has a more significant impact, and the repair rate is higher than more serious low but reports have a lower fix rate.

(2) Influence of defect report severity on defect resolution time since defect resolution time is one of the main attributes of defect reports; this paper analyzes the relationship between resolution time and severity level according to the specific situation of the Eclipse project. Figure 5 to Table 6 shows the distribution of average resolution time corresponding to defect reports of different severity in the Eclipse project. On the overall trend, as the severity level decreases, the average resolution time for defect reports gradually increases. For example, the blocker level with the highest severity reported an average resolution time of 45 days. The enhancement level with the lowest severity said an average resolution time of 441 days.

It can be seen that defects with an average severity level will be resolved first and thus have a shorter resolution time. On the other hand, defects with higher or lower severity are affected by the difficulty and importance of resolution, resulting in increased resolution time.

(3) The influence of developers on the defect repair rate

Two thousand six hundred twenty-four developers participate in Eclipse defect repair on the Bugzilla platform. Users with less than five defect reports are the most, accounting for about 35% of the total users. According to developers' number of bug reports, developers are divided into 6 categories.

(4) Influence of defect report severity on software component repair rate because project component is one of the essential attributes of a defect report, this experiment analyzes the distribution of defect reports for different parts of Eclipse products. The total number of components for an Eclipse project is 782. According to the distribution of defect reports, the

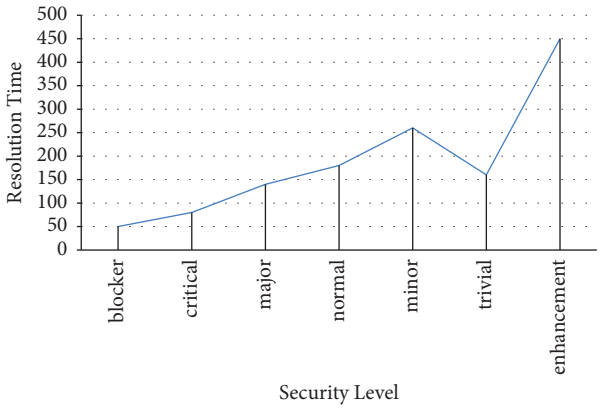

Figure 5: Distribution of the resolution time of each severity level defect report.

Table 6: Distribution of the resolution time of each severity level defect report.

Blocker	Critical	Major	Normal	Minor	Trivial	Enhancement
50	80	140	180	260	160	450

data of the top 10 members with the most significant number of defect reports were selected for analysis. As shown in Figure 6, the repair rate of each component defect reported is higher than 80%, and the repair rate of the TPTP component is the highest, reaching 99.82%.

Furthermore, the various means to optimize the cloud server are detailed here as planning and strategy, evaluation of cost optimization, optimization of spot instances and containers, Cloud Auto-scaling, concentrating on managing cases, make the switch to new cloud deployment models. Do not forget to use cloud optimization tools and services, forecasting, modeling, and analytics, as well as optimize cloud storage.

By giving more resources to a workload than necessary, many enterprises frequently experience overspending in the cloud. For your cloud infrastructure and your corporation, integrating cloud optimization strategies can have a number of significance, including the following.

Efficiency in the cloud: it is attained when workload, performance, compliance, and cost are continuously weighed against the best-fit infrastructure in real-time. Utilizing cloud optimization techniques will improve the performance of your cloud environment by reducing resource waste as much as feasible. Cost savings: although there are many other types of cloud optimization, for many firms, cost optimization is the most crucial one. Costs are decreased as a benefit of lowering cloud waste.

Greater visibility: analytics is used in cloud optimization strategies to increase visibility into your cloud environment. Enhanced productivity: after implementing a cloud optimization plan, personnel will spend less time attempting to resolve issues because an optimized environment stops the issues before they start.

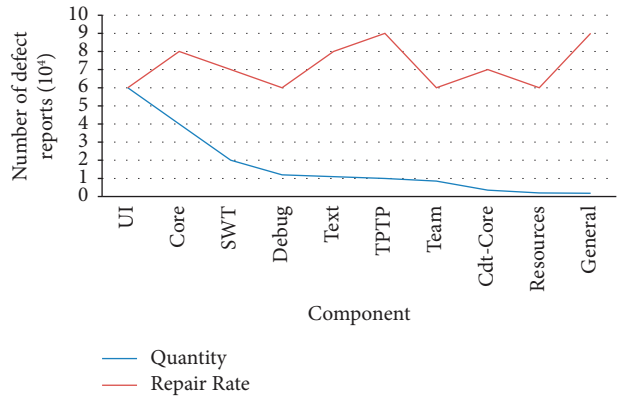

Figure 6: Distribution of defect reports and repair rates for different components.

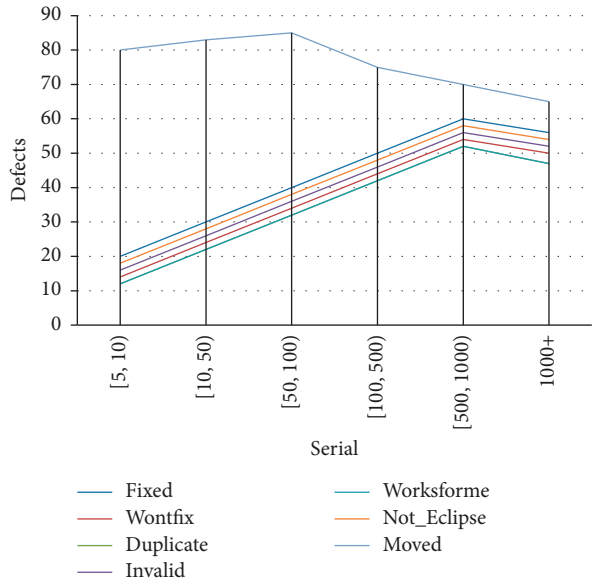

Figure 7: Proportion of solutions for developers to solve defects.

Organizational innovation and efficiency: implementing cloud optimization frequently results in a culture transformation within firms, leading to better teamwork and decision-making.

5. Discussion and Findings

The defect repair rates of developers with various numbers of defects are counted in this study, as illustrated in Figure 7 and Table 7. Defect repair rates range from 80.11% when there are fewer than five defects to 49.75% when there are more than 1,000 defects. Additionally, the scheme's repetition rate increased from 3.87% to 18.69%, its non-reproducibility rate increased from 2.43% to 10.18%, and the amount of illegal, irreparable, and incomplete information increased by 10%. The aforementioned phenomenon demonstrates how a spike in developers' problem reports will lower their rate of correction.

According to the severity of defect reports corresponding to different components, it can be seen that among

Table 7: Proportion of solutions for developers to solve defects.

Serial	Fixed	Wontfix	Duplicate	Invalid	Worksforme	Not_Eclipse	Moved
[5, 10)	20	14	12	16	12	18	80
[10, 50)	30	24	22	26	22	28	83
[50, 100)	40	34	32	36	32	38	85
[100, 500)	50	44	42	46	42	48	75
[500, 1000)	60	54	52	56	52	58	70
1000+	56	50	47	52	47	54	65

the members, the defect reports with the severity level average level account for the most significant proportion; in the test and performance tools platform (TPTP) component such as Bugzilla introduced in this study is a software bug tracking tool. TPTP components used for the blocker level defect reports with the highest severity level account for the most critical balance of TPTP components as a result of the experimental analysis, which has been proven in Figure 6. Therefore, make it the highest fix rate; the lowest severity enhancement level defect reports have the most important proportion in the text component, resulting in the lowest fix rate. It can be seen that the severity levels of defect reports of different components are consistent. The defect repair rate is higher for the higher severity level, and the defect repair rate for the element with more high-severity defects is also higher.

6. Conclusion

The maintenance of expansive open-source projects now requires an accurate, thorough, and effective assessment of the severity of defect reports. The software defect report warehouse may be swiftly and uniformly managed using the defect report management solution. The authors in this study systematically analyzed the defect report repositories for Mozilla products and Eclipse products using the Bugzilla defect report management tool. It is expected that the release of a new version of the product will result in an increase in the number of software defect reports; the severity of these reports will draw developers' attention to software defects, increasing the defect repair rate and cutting down the amount of time it takes to fix software defects development. A user's fix rate is independent of the number of bug reports they have. The abovementioned rules are supported by various product components. According to the aforementioned statistical findings, it is clear that using the machine learning method such as random forest approach, bagging approach, SVM approach, Bayesian classification techniques, and neural network techniques. To assess the severity of software defects, reports and classifying their data can increase the classification of software defects' accuracy, and its seen among all ML approaches, the most efficient one is random forest and bagging approach, therefore, these machine learning methods significantly reduce the amount of time needed to fix serious flaws and increase the rate at which defects are fixed, all of which will ensure the effectiveness of software version iteration. In the future, more in-depth research will be carried out to predict the severity of defect reports across projects.

Data Availability

The data shall can be obtained from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] B. Zhou, I. Neamtiu, and R. Gupta, "Experience report: how do bug characteristics differ across severity classes: a multi-platform study," in *Proceedings of the 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, Gaithersbury, MD, USA, November 2015.

[2] Q. Liu, H. Washizaki, and Y. Fukazawa, "Adversarial multi-task learning-based bug fixing time and severity prediction," in *Proceedings of the 2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*, pp. 185-186, Kyoto, Japan, October 2021.

[3] Y. Jia, X. Chen, S. Xu, G. Yang, and J. Cao, "EKD-BSP: bug report severity prediction by extracting keywords from description," in *Proceedings of the 2021 8th International Conference on Dependable Systems and Their Applications (DSA)*, IEEE, Yinchuan, China, August 2021.

[4] K. Vijayakumar and V. Bhuvaneswari, "How much effort needed to fix the bug? A data mining approach for effort estimation and analysing of bug report attributes in Firefox," in *Proceedings of the 2014 International Conference on Intelligent Computing Applications*, IEEE, Coimbatore, India, March 2014.

[5] A. F. Otoom, D. Al-Shdaifat, M. Hammad, and E. E. Abdallah, "Severity prediction of software bugs," in *Proceedings of the 2016 7th International Conference on Information and Communication Systems (ICICS)*, IEEE, Irbid, Jordan, April 2016.

[6] A. Kaur and S. Goyal Jindal, "Severity prediction of bug reports using text mining: a systematic review," in *Proceedings of the 2018 International Conference on Advances in Computing, Communication Control and Networking*, pp. 774–780, ICACCCN), Greater Noida, India, October 2018.

[7] A. Mehbodniya, J. L. Webber, M. Shabaz, H. Mohafez, and K. Yadav, "Machine learning technique to detect sybil attack on IoT based sensor network," in *IETE Journal of Research*, pp. 1–9, Informa UK Limited, 2021.

[8] A. Kaur and S. G. Jindal, "Bug report collection system (BRCS)," in *Proceedings of the2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence*, IEEE, Noida, India, January 2017.

[9] L. Kumar, M. Kumar, L. B. Murthy, S. Misra, V. Kocher, and S. Padmanabhuni, "An empirical study on application of word embedding techniques for prediction of software defect severity level," in *Proceedings of the Annals of Computer Science*

and Information Systems. 16th Conference on Computer Science and Intelligence Systems, IEEE, Sofia, Bulgaria, September 2021.

[10] Y. Tian, D. Lo, and C. Sun, "DRONE: predicting priority of reported bugs by multi-factor Analysis," in *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, IEEE, Eindhoven, Netherlands, September 2013.

[11] T. Gera, J. Singh, A. Mehbodniya, J. L. Webber, M. Shabaz, and D. Thakur, "Dominant feature selection and machine learning-based hybrid approach to analyze android ransomware," in *Security and Communication Networks*, J. Cui, Ed., vol. 2021, pp. 1–22, Hindawi Limited, Article ID 7035233, 2021.

[12] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," in *IEEE Access*, vol. 6, pp. 35743–35752, Institute of Electrical and Electronics Engineers (IEEE), 2018.

[13] S. Ognawala, M. Ochoa, A. Pretschner, and T. Limmer, "MACKE: compositional analysis of low-level vulnerabilities with symbolic execution," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ASE'16: ACM/IEEE International Conference on Automated Software Engineering*, ACM, Singapore, September 2016.

[14] Y. Lei, S. Vyas, S. Gupta, and M. Shabaz, "AI based study on product development and process design," in *International Journal of System Assurance Engineering and Management*, vol. 13, no. Issue S1, pp. 305–311, Springer Science and Business Media LLC, 2021.

[15] T. Liu, R. Neware, M. W. Bhatt, and M. Shabaz, "A study on detection and defence of malicious code under network security over biomedical devices," in *The Journal of Engineering*, Institution of Engineering and Technology (IET), 2022.

[16] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference*, IEEE, Vasteras, Sweden, July 2014.

[17] P. Rattan, M. Arora, M. Rakhra et al., "A neoteric approach of prioritizing regression test suites using hybrid ESDG models," *Annals of the Romanian Society for Cell Biology*, vol. 2965, 2021, https://www.annalsofrscb.ro/index.php/journal/article/view/2838.

[18] X. Xia, D. Lo, E. Shihab, and X. Wang, "Automated bug report field reassignment and refinement prediction," in *IEEE Transactions on Reliability*, vol. 65, no. Issue 3, pp. 1094–1113, Institute of Electrical and Electronics Engineers (IEEE), 2016.

[19] K. K. Chaturvedi and V. B. Singh, "Determining Bug severity using machine learning techniques," in *Proceedings of the 2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, IEEE, Indore, India, September 2012.

[20] L. Kumar, P. Gupta, L. B. Murthy et al., "Predicting software defect severity level using sentence embedding and ensemble learning," in *Proceedings of the 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, Palermo, Italy, September 2021.

[21] I. Alazzam, A. Aleroud, Z. Al Latifah, and G. Karabatis, "Automatic bug triage in software systems using graph neighborhood relations for feature augmentation," in *IEEE Transactions on Computational Social Systems*, vol. 7, no. Issue 5, pp. 1288–1303, Institute of Electrical and Electronics Engineers (IEEE), 2020.