Available online at www.worldscientificnews.com

World Scientific News

An International Scientific Journal

WSN 189 (2024) 1-22 EISSN 2392-2192

Software bugs: detection, analysis and fixing

Rotimi-Williams Bello*, Soncy Justin Tobi

Department of Mathematics and Computer Science, University of Africa, Toru-Orua, 561101 Sagbama, Bayelsa, Nigeria

*E-mail address: sirbrw@yahoo.com

ABSTRACT

The focus of this study is on detecting, analyzing, and fixing of software bugs. The main goal is to identify cost-effective methods for developing and managing software systems by introducing a post-deployment debugging approach. This approach serves two purposes: (1) tracking software stability and (2) acting as a repository for software bug data. To achieve this objective, a web-based application called BugTracker was proposed and developed. BugTracker manages software testing and post-deployment activities while also serving as a repository for bug data. Testers and end-users can report bugs through BugTracker, which developers analyze and resolve by updating the program files with new versions. The BugTracker system was designed using UML and Overview models, and implemented using PHP, HTML, JavaScript, and MySQL database technology. Evaluation and testing of BugTracker demonstrated increased developer productivity, reduced production costs, and improved software stability. This study shows that effectively managing post-deployment activities can enhance software stability and reduce development costs.

Keywords: Software bugs, Bugs detection, Bugs analysis, Bugs fixing, Bugs tracker

1. INTRODUCTION

On June 4th, 1996, the inaugural flight of the Ariane 5 launcher met with failure. Approximately 40 seconds after the launch sequence began, at an altitude of around 3700 meters, the launcher deviated from its intended path, disintegrated, and exploded.

(Received 20 November 2023; Accepted 16 December 2023; Date of Publication 18 December 2023)

This catastrophic failure resulted from a critical software error in the Inertial Reference System of the launcher, leading to a complete loss of guidance and altitude information 37 seconds after the ignition of the main engine, or 30 seconds post-lift-off. Software has evolved into a crucial component across a wide spectrum of products and services, spanning various sectors of the economy.

Examples of software-intensive systems encompass vast and diverse domains, such as embedded systems for automotive use, telecommunications, wireless ad-hoc networks, web-centric business applications, and more. Our daily lives have grown increasingly dependent on complex software-intensive systems, from banking and communication to transportation and healthcare. The innovation and capabilities of industries are significantly influenced by their proficiency in software technology.

Nevertheless, it is widely acknowledged that software developed within many organizations today often lacks quality and stability (Kim et al., 2006; Bello and Otobo, 2018). A pervasive communication gap exists between software developers and end-users, resulting in frequent instances of software abandonment.

This communication gap also hampers the collection of meaningful feedback from system users, and the scant feedback received is often inadequately documented. Moreover, the absence of a comprehensive repository for software bug data makes it challenging to make informed decisions about enhancing the software development process. To address these shortcomings in quality and stability, software development organizations have embraced four common strategies.

The first involves recruiting top-tier talent to craft bug-free software, although defining the criteria for selecting such personnel is seldom straightforward, and it is nearly impossible for an individual to create a bug-free system that satisfies the expectations of diverse users. Another approach is the reuse of existing software rather than starting from scratch. Unfortunately, only a few organizations have managed to create versatile, dependable software that can be reused without substantial modifications. Developing software at higher levels of abstraction constitutes yet another strategy. However, convincing others of the merits of this approach remains challenging, given concerns about potential system performance drawbacks.

Hence, the fourth and final approach, which this study examines, advocates a post-deployment method for analyzing software bugs. This approach aims to reduce the number of software bugs and their variability over time while simultaneously establishing a bug-data repository that can enhance the software development process.

The primary challenge we encountered was the absence of comprehensive software development data. This issue stems from the fact that the Nigerian software development industry is still in its nascent stages, predominantly comprising small-scale enterprises with a limited number of developers. These developers often operate without strict adherence to established software development processes. Instead, their primary focus has been on ensuring that the software functions correctly, often overlooking the documentation of the development steps taken.

2. LITERATURE REVIEW

This section presents a comprehensive review of software bugs. Additionally, the overview of relevant studies and their respective findings within this field are also reviewed.

2. 1. Software defects

A software defect can be described as an imperfection, flaw, issue, error, glitch, inconsistency, or deviation in a computer program or software system (Rodríguez-Pérez et al. 2020). This imperfection leads the software to produce results that are inconsistent or incorrect, causing the system to exhibit unexpected behavior. Any deviation from the specified behavior of a system is considered a defect. The majority of defects arise due to human errors and mistakes made during either the program's source code creation or its design. A few defects may also stem from compilers generating incorrect code.

Understanding the source of a problem is believed to be instrumental in preventing its recurrence in the future. Therefore, the initial step in the process of reducing the number of defects in future software systems is to identify how defects are introduced (Bello et al., 2018; Bader et al., 2019). Defects primarily result from human factors within the software development process. These human factors encompass mistakes, oversights, or misunderstandings during various stages of software development. Hence, we examine the phases of software development to pinpoint how defects can be introduced at each stage.

2. 1. 1. Requirement analysis and specification

The process of requirement analysis and specification addresses the fundamental question of "WHAT" with regard to software development. It involves detailing the expected functions and features of the software under development. The outcome of this stage is the creation of a document known as the Software Requirement Specification (SRS) (Lutz et al., 2014). The SRS document precisely outlines the anticipated functionalities that users of the system should anticipate. Any inconsistencies in requirements or incomplete implementation of these requirements can lead to what Amusuo et al. (2022) referred to as errors of omission, which encompass unintentional omissions, lack of clarity, and ambiguity situations where two individuals may interpret the requirements differently. It is worth noting that bugs introduced during the requirement analysis stage, if left unaddressed, tend to be the costliest and challenging to rectify.

For instance, consider a school's result computation software that fails to correctly handle carryover courses. In this scenario, the software replaces old scores with new ones due to a lack of clear guidelines from the school's management on how to manage and handle carryover scores. This serves as an illustration of how bugs can be introduced during the software requirement specification stage.

2. 1. 2. Design

The design phase serves as the bridge that transforms the "WHAT" established during software requirement analysis into the "HOW." Its outcome furnishes comprehensive details on how the software will be implemented, with minimal need for further engagement with the anticipated system users. This phase encompasses the specification of critical elements such as the software's user interface, architectural framework, and component-level design. In Anthony's paper (Anthony, 1999), architectural design involves modularizing the software, while the interface delineates how these modules will interact. Component or detailed design further delves into creating a meticulous pseudocode outlining the processes to be executed within each module of the software.

According to Amusuo et al. (2022), two prevalent types of software errors may be introduced during the design phase: errors of omission, which involve unintentional omissions from the SRS document, and errors of commission, where erroneous information is provided, only to be corrected later. Notably, numerous performance-related bugs originate during the design phase, leading to fundamental software flaws. An illustration of how bugs can surface during the design stage is evident in a school management portal. In this case, the portal was met with rejection from both staff and students due to its overly complex interface, which rendered its usage exceedingly challenging. Additionally, users complained of overlapping button designs, resulting in unintended operations being triggered.

2. 1. 3. Coding and implementation

The coding and implementation phase revolves around translating the "HOW" derived from the design phase into an executable program. It entails the conversion of all design specifications into lines of code using a designated programming language, as determined or agreed upon during the design phase. Implementation represents the real-world realization of processing functions and information structures (Ricca and Tonella, 2001). The most prevalent error encountered during the coding stage is the error of commission, wherein actions are taken inappropriately (Amusuo et al., 2022). Remarkably, 50% of the critical bugs or errors discovered during implementation do not stem from the source code itself; instead, they primarily arise from the programmer's inability to grasp the design or from design misinterpretations.

For instance, consider a software system designed to generate a list of students with a minimum of four carryovers in a particular semester. The system was only producing a list of students with more than four carryovers, inadvertently omitting those with exactly four carryovers. A meticulous review of the source code revealed that an error had occurred in the conditional statement. Instead of:

```
if ($n_co >= 4)
   status = "you are required to repeat this session";
```

The code mistakenly read:

```
if ($n_co > 4)
   status = "you are required to repeat this session";
```

2. 1. 4. Documentation

Documentation serves as a cornerstone in the realm of successful software engineering, offering invaluable guidance for software support. It elucidates the software's usage, operational procedures, requirements for optimal functionality, and its overall functions and capabilities.

According to the standards established by the IEEE Standards Association (Šmite et al., 2014), documentation constitutes a vital component of software quality assurance. Amusuo et al. (2022) observes that errors of commission and omission are often embedded within documentation, with ambiguity being a common culprit. Bugs within documentation frequently result in performance-related issues.

For instance, consider a result computation system that experienced a crash mere weeks after deployment, despite exhaustive efforts to rectify the issue. It was subsequently revealed that the documentation specifying the minimum system requirements was inaccurate. The accompanying documentation had indicated that a PC with a configuration of 500MB RAM, a 1GHz Processor, and a 10GB Hard disk was sufficient to run the software. However, this configuration proved suitable only for testing the software with a limited number of records, not for the entire university. To accommodate all university departments, the software necessitated a minimum system configuration of approximately 8GB RAM, a 4GHz Quad-core system processor, and a minimum of 1TB hard disk space.

2. 1. 5. Maintenance

Maintenance refers to any action taken concerning a software application after it has been developed and placed into operational use (Lutz et al., 2014). This maintenance effort can be categorized as either remedial or adaptive in nature. Remedial maintenance involves addressing and rectifying faults or bugs detected within the software, while adaptive maintenance pertains to modifying the software, either due to changes in user requirements or alterations in the software's operating environment. In the course of resolving or fixing a bug, it is possible to inadvertently introduce a secondary bug into the application, even if the original issue has been resolved.

These bugs, generated during the process of eliminating an existing bug, are commonly referred to as "fix defects" or "fix bugs," as described by Amusuo et al. (2022). It is important to note that fix defects often result from errors of commission.

For instance, consider a transcript-generating software initially capable of generating transcripts for all students within a department in a single operation. However, following an upgrade of the software with additional features, it was unable to generate more than 10 transcripts in a single operation. Upon closer examination of the code, it was revealed that the programmer had set a benchmark of 10 while testing a new module, and this setting was inadvertently left unchanged during deployment. This situation exemplifies how a fix defect can arise during the maintenance phase.

2. 2. Software bug detection

The process of identifying bugs within a software system is commonly referred to as software testing. In this section, we delve into the intricacies of the software testing process, as well as the criteria that dictate the conclusion of a software test. Software bugs assume critical importance when software transitions to the production stage. Bugs not only diminish client satisfaction but also pose a significant threat to the reputations of software companies (Uddin et al., 2017). Software bug detection has emerged as a prominent area of focus for researchers, aiming to mitigate the ramifications of these bugs.

Recent research has underscored that as much as half of developers' and software testers' efforts are consumed by avoidable tasks (Hindle and Onuczko, 2019), the majority of which stem from system defects (Kulkarni et al., 2021). The process of defect management, as depicted in Figure 1, comprises defect prevention, establishing deliverable baselines, defect discovery, defect resolution, and process improvement. Consequently, defect discovery stands as a foundational stage, setting the stage for subsequent efficient defect handling processes.

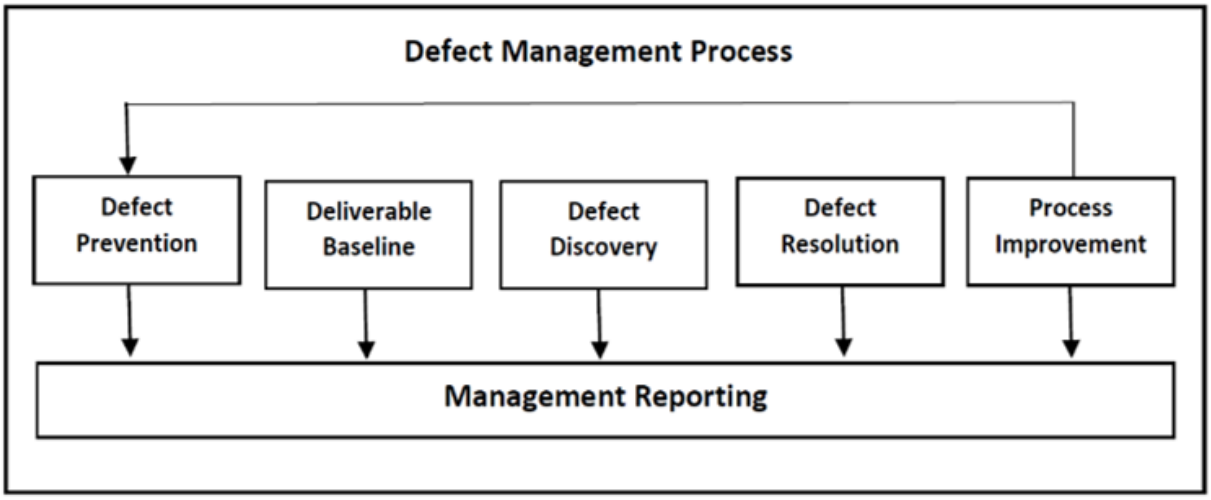


Figure 1. Defect handling proces

2. 2. 1. AI-powered bug detection approaches in software engineering

Li et al. (2017b) proposed a mining strategy for acquiring the attributes of software vulnerabilities using data mining. The proposal introduces a mining algorithm for vulnerabilities that efficiently identifies and analyzes key software vulnerability attributes using data mining techniques. Bian et al. (2018) proposed a NAR-miner, a tool designed to uncover negative association rules from code to aid in bug detection using data mining. The NAR-miner system was developed to automatically extract negative association programming rules from extensive software systems and subsequently identify rule violations as a means of bug detection. Alqahtani and Rilling (2017) proposed an approach based on ontology to automate the tagging of software artifacts using ontologies. They introduced a fully automated classification and tagging method capable of extracting security tags from text without the requirement for training the system using manual training data.

The technology presented by Chen et al. (2018) involves software behavior decision trees for defect detection. This layered detection technology relies on a model of software behavior decision trees. The approach described by Schekotihin et al. (2018) involves test-driven ontology development using Protégé. An ontology was introduced through a combination of logic and test-driven development methodologies. It primarily drew from cues in test-driven development within the field of software engineering. Subsequently, it underwent processing, leading to the identification and presentation of detected bugs. DWEN, an acronym for Deep Word Embedding Network, is a system developed by Budhiraja et al. (2018) using neural network for the purpose of detecting duplicate bug reports within software repositories. The proposed approach primarily focuses on the identification and categorization of duplicate bug reports.

This methodology is crucial in preventing the assignment of the same bug to multiple developers. Detecting and resolving bugs can lead to a substantial reduction in the overall cost of the software development process. MODE, which stands for Automated Neural Network Model Debugging via State Differential Analysis and Input Selection, is a system introduced by Ma et al. (2018). The proposed debugging approach using neural network primarily functions by performing differential analysis on the model's state to identify and examine the

internal model features responsible for bugs. It subsequently initiates a process akin to training input selection, similar to program input selection in regression testing. DeBGUer, denoted as a tool for bug prediction and diagnosis, was introduced by Elmishali et al. (2019).

DeBGUer, which uses fault prediction model was put forth as a web-based application designed to forecast and categorize software bugs. Its implementation was structured around distinct levels, including learning, diagnosing, and planning (LDP) as part of its approach. The LDP paradigm represented a novel approach for integrating artificial intelligence and software engineering to detect software bugs and facilitate the correction process. The work by Elmishali et al. (2018) introduces an artificial intelligence paradigm for addressing and resolving software bugs. The main features and ideas are machine learning, which focused on leveraging knowledge from the structure of the source code and historical revisions of failures to pinpoint software components that are likely to contain bugs.

Automated diagnosis, which encompassed the specifications of software components necessitating modification to rectify identified bugs. Automated planning, which devised additional tests when necessary to improve the precision of bug diagnostics. The primary concept revolves around the significant expenses associated with software testing, compelling practitioners to seek valuable insights for detecting defects prior to the testing phase (Sheneamer and Kalita, 2016). In pursuit of this objective, predictors of software bugs primarily employ data mining approaches, aiding in the prioritization of software module lists for efficient allocation of testing resources. This approach allows for the discovery of most existing defects with minimal effort and cost (Pandey et al., 2018). There is a notable upward trend in the number of studies employing intelligent techniques in agile software development (Perkusich et al., 2020). These techniques find application in various areas, including effort estimation, requirements prioritization, resource allocation, requirements selection, and requirements management (Xie, 2018).

Predicting software bugs is crucial because bugs can also emerge from software configuration changes during runtime. These types of bugs can have severe consequences (Ni et al., 2020). Consequently, researchers have explored the integration of artificial intelligence methods for bug prediction when software configurations change or when inputs are likely to generate bugs. These approaches aim to identify high-risk updates immediately after the code commit process, rather than waiting until the entire software module is completed. Predictive models that focus on changes that induce bugs in specific software applications can be developed based on information describing previous software updates. These models take advantage of the fresh knowledge that developers have when they commit code updates, making it easier to analyze and connect bug discoveries with other findings.

2. 2. 2. Software engineering bug prediction approaches using AI techniques

Otoom et al. (2016) focused on predicting the severity of software bugs. The approach integrated robust classification methods with a newly suggested feature set to improve the prediction of bug severity. Additionally, it employed a boosting algorithm and classification algorithms to further enhance its performance. Li et al. (2017) introduced software defect prediction through the utilization of convolutional neural networks. They put forward a framework named “Defect Prediction” that harnessed the power of Convolutional Neural Networks (DP-CNN). This framework embraced the principles of deep learning to create meaningful features using the Abstract Syntax Trees (ASTs) of programs. Hammouri et al. (2018) presented a machine learning approach for software bug prediction. This represented a

software bug prediction model grounded in machine learning. It harnessed the capabilities of three supervised machine learning algorithms to effectively anticipate software faults based on historical data. The employed classifiers encompassed naive Bayes classifiers, decision trees, and artificial neural networks.

Pandey et al. (2020) introduced BPDET, an efficient software bug prediction model that incorporates deep representation and ensemble learning techniques. This approach was a fundamental classification method established on a framework for bug prediction, leveraging Deep Representation and Ensemble Learning techniques (BPDET) for software bug prediction. It made use of both Ensemble Learning (EL) and Deep Representation (DR) strategies. Traditional software metrics served as the basis for software bug prediction. A Stacked Denoising Auto-Encoder (SDA) was employed to create a deep representation of the software metrics, known for its robust learning capabilities. Pandey et al. (2018) introduced a comprehensive model that prototypes the use of Bayesian network classifiers for software bug prediction. The model employed two distinct types of classifiers for bug prediction: General Bayesian networks and augmented naive Bayes classifiers. Furthermore, it conducted a comparative analysis of various

Bayesian classifiers and networks to evaluate their effectiveness in bug prediction. Manjula and Florence (2019) introduced a hybrid approach for software defect prediction utilizing deep neural networks and software metrics. They presented a hybrid approach that combined Deep Neural Networks (DNNs) for classification with genetic algorithms for feature optimization.

This hybrid approach included an improved version of genetic algorithms, incorporating a novel technique for computing fitness functions and designing chromosomes. Kumar and Gupta (2016) introduced a software bug prediction system based on neural networks. This model for software bug prediction utilized gradient descent adaptive learning with back-propagation techniques.

3. SYSTEM DESIGN

3. 1. Analysis of existing systems

The analysis of existing systems for bug detection and analysis involves a comprehensive examination of the tools, methodologies, and practices currently employed. The majority of existing systems still rely on manual bug detection and reporting, which can be time-consuming and prone to human error. Automation is often limited to basic testing, leaving more complex or exploratory bug detection to manual efforts. Coordination and communication among project management teams can be disjointed, resulting in inefficient bug resolution. The analysis of existing systems uncovers several limitations and challenges that hinder effective bug detection, analysis, and fixing.

Recognizing these challenges is crucial for the design of an improved system. One of the primary challenges is the reliance on manual bug detection and reporting, which not only consumes valuable time but can also lead to inconsistent and incomplete bug data. Ineffective communication channels and practices among various teams involved in software development can result in misunderstandings, delays, and missed bug-related information. Many existing systems lack comprehensive testing automation, leaving critical areas of software unchecked and vulnerable to bugs that may go undetected until late in the development cycle.

Inadequate tracking and reporting mechanisms can make it difficult to monitor the progress of bug resolution, assess the impact of bugs, and prioritize them effectively.

3. 2. Description of the proposed system

In response to the limitations and challenges identified in the existing systems, the proposed system aims to revolutionize the bug detection and analysis process. It is designed to provide a more efficient, automated, and collaborative environment for identifying, understanding, and resolving software bugs. The proposed system incorporates state-of-the-art technologies and methodologies, addressing the shortcomings of existing systems and providing a comprehensive solution to the software bug problem. The journey towards an improved bug detection and analysis system begins here, with the recognition of the shortcomings of the status quo and the commitment to overcoming them through a well-designed and strategically planned solution. Figure 2 shows the system architectural design.

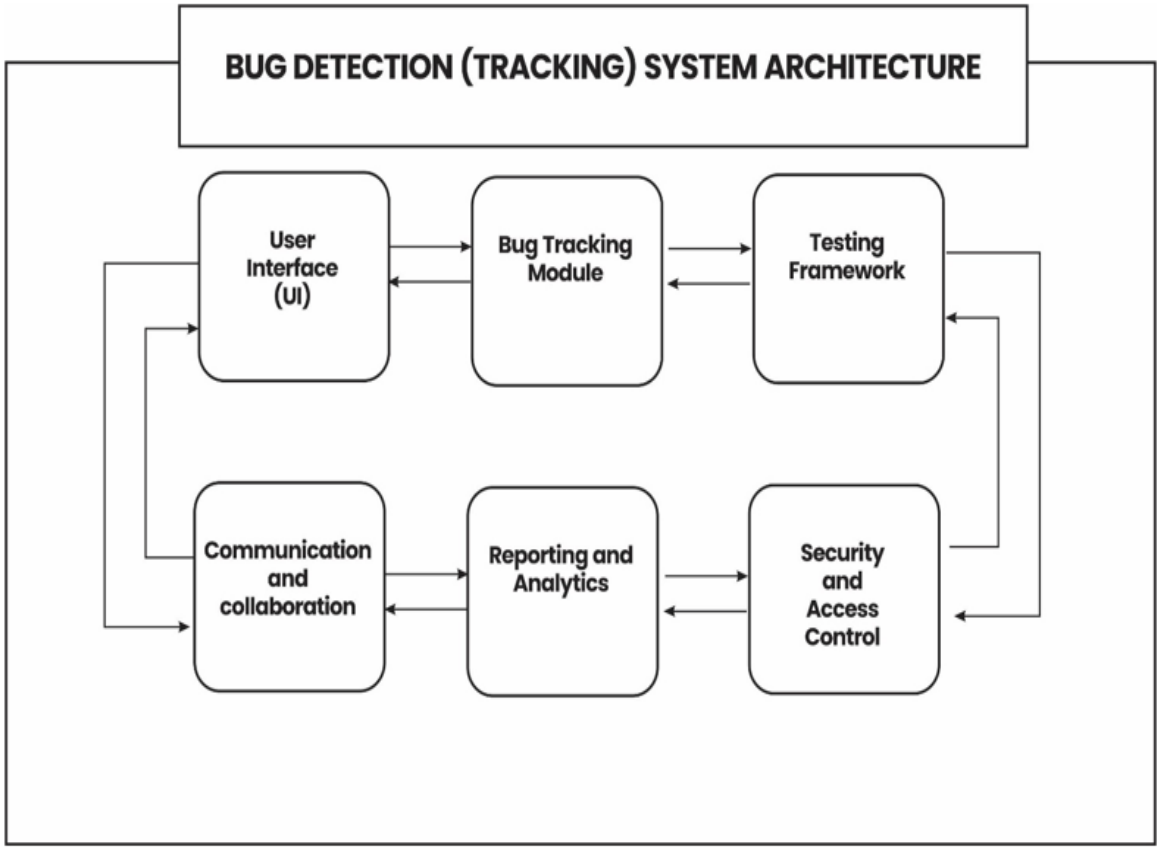


Figure 2. System architectural design

3. 2. 1. Key components and features of the proposed system

The success of a bug detection and analysis system hinges on its components and features. In this section, we outline the core elements that constitute the proposed system and describe the features that empower it to effectively tackle the challenges of software bugs.

3. 2. 1. 1. Bug tracking and management

At the heart of the proposed system is a robust bug tracking and management module. This component enables the systematic recording, categorization, and monitoring of software bugs throughout their lifecycle. Key features of this module include:

a) Issue tracking: A centralized repository for logging and tracking bugs, providing a comprehensive view of the status and history of each issue.
b) Prioritization: Tools for assigning priorities to bugs, ensuring that critical issues are addressed promptly.
c) Assignment: The ability to assign bugs to specific team members, streamlining responsibility and accountability.
d) Real-time updates: Real-time notifications and updates to keep all stakeholders informed about the progress of bug resolution.

3. 2. 1 .2. Testing framework

Automation is a cornerstone of efficient bug detection and analysis. The proposed system includes an advanced automated testing framework, incorporating the following features:

a) Regression testing: Automatic retesting of software to identify new bugs introduced during development.
b) Test case management: A repository for test cases and scripts to ensure comprehensive coverage.
c) Continuous integration: Seamless integration with the development process to enable automated testing with each code commit.
d) Custom test suites: The ability to create tailored test suites for specific projects or modules.

3. 2. 1. 3. Real-time communication and collaboration

Efficient communication and collaboration among development, QA, and project management teams are essential for effective bug resolution. The proposed system offers features such as:

a) Discussion threads: Dedicated discussion threads for each bug, facilitating focused conversations and eliminating communication silos.
b) File sharing: The ability to share files, screenshots, and logs related to bug reports directly within the system.
c) Collaborative workspaces: Shared workspaces where team members can collectively address and resolve bugs.

3. 2. 1. 4. Reporting and analytics

Comprehensive reporting and analytics are integral to understanding the bug landscape and making data-driven decisions. Key components and features include:

a) Custom dashboards: Customizable dashboards that allow users to create visual representations of bug data, highlighting trends and patterns.

b) Historical data analysis: Tools for analyzing historical bug data to identify recurring issues and areas for improvement.
c) Export and sharing: Options to export reports and share insights with stakeholders.

3. 2. 1. 5. Integration capabilities

The proposed system is designed with flexibility and adaptability in mind. Integration capabilities include:

a) Version control integration: Integration with version control systems to link bug reports directly to code changes.
b) IDE plugins: Plugins for popular integrated development environments (IDEs) that enable developers to report and track bugs without leaving their coding environment.

3. 2. 1. 6. Security and access control

Ensuring the security and privacy of bug data is paramount. The system provides features for:

a) Access control: Role-based access control to restrict access to sensitive bug data to authorized personnel.
b) Data encryption: Encryption of data in transit and at rest to safeguard against data breaches.
c) Audit trails: Comprehensive audit trails to track all actions and changes made within the system for accountability and compliance purposes.

3. 2. 1. 7. Mobile accessibility

In an increasingly mobile world, accessibility on the go is essential. The system offers responsive web interfaces, allowing users to report, monitor, and manage bugs from their smartphones and tablets. The combination of these key components and features creates a comprehensive bug detection and analysis system that empowers development teams to streamline their bug resolution processes, improve communication, and enhance software quality. In the subsequent sections of this study, we explore the significance and potential impact of this system on software development practices.

3. 3. Database design

This section establishes the data structure that will store the software's data. The data types employed for software data storage can encompass integers, characters, strings, etc., contingent upon the specific application software in use. Various data structures can be employed, such as arrays, tables, lists, etc. The designated database for storing this software's data is named "Bug-Tracking," comprising nine (9) tables, they are (1) Contacts (2) Employees (3) Priorities (4) Projects (5) Roles (6) Statuses (7) Tickets (8) Types (9) Users. Figure 3 shows the bug-tracking database (Figure 3).

3. 4. Deployment and system specification

The client-side software requirements for the BugTracker are as follows: (1) A web browser (compatible with IE 9.0 and above, Chrome 25 and above, Firefox 27 and above), (2)

An operating system (compatible with any) and (3) IDE – Visual Studio Code. To ensure the efficient operation of BugTracker on the client's side, the hardware configuration should include (1) Internet access, (2) Motherboard model: any motherboard, (3) RAM size: a minimum of 256MB, (4) CPU speed: a minimum of 200MHz. The development environment for BugTracker involves the use of the following software tools (1) Visual Studio Code, (2) Adobe Photoshop CS6. BugTracker is deployed on a web server that supports the XAMP stack, which includes Windows, Apache, MySQL, and PHP.

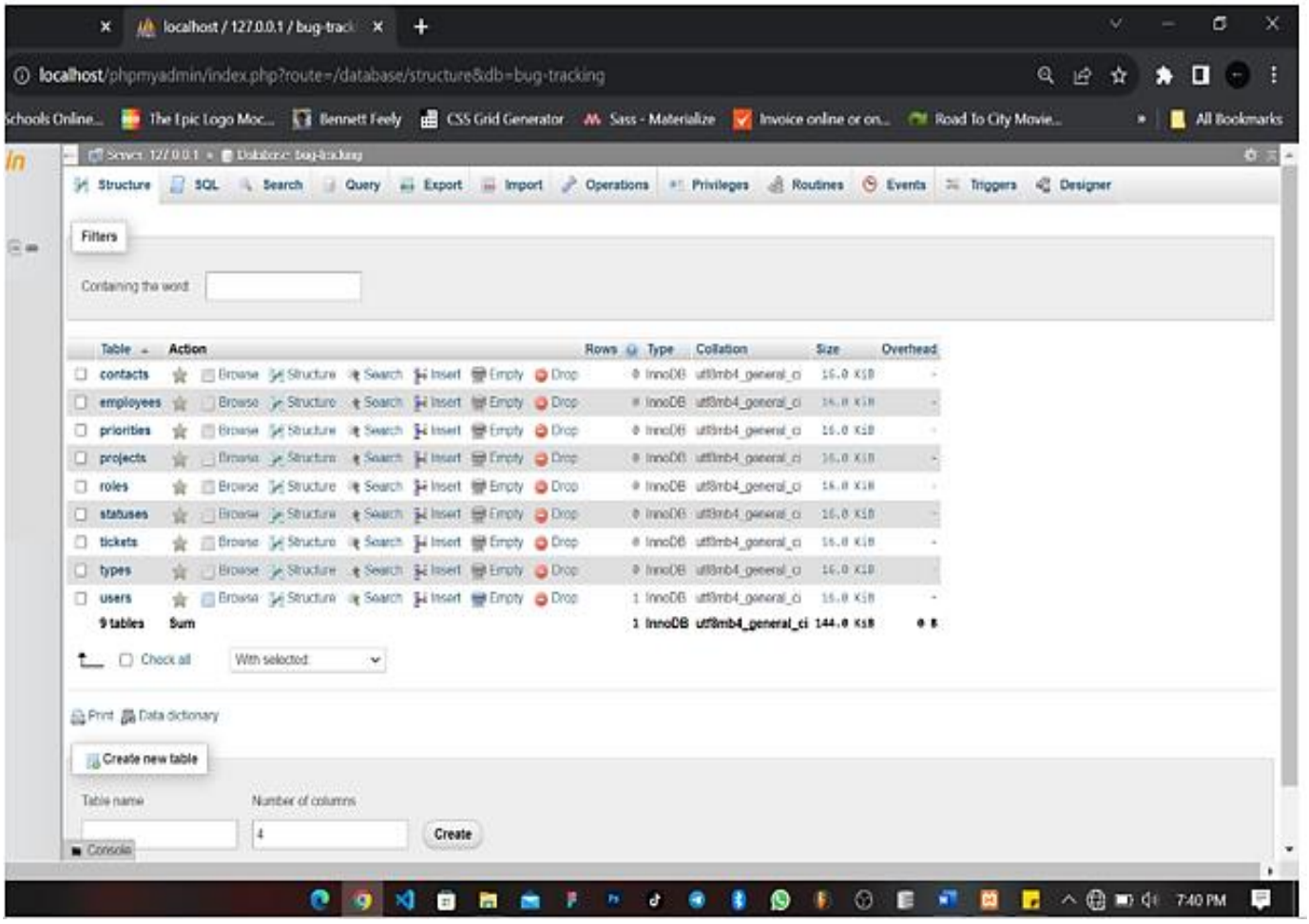


Figure 3. Bug-tracking database

4. SYSTEM IMPLEMENTATION

4. 1. System implementation

System implementation is the phase where the previously created designs, developed during the analysis stage, are transformed into a fully functional software system. This process entails selecting a programming language that aligns with the specific requirements of the application. It is crucial to perform testing with sample or test data before finalizing the implementation to verify that the design can effectively fulfill its intended purpose and

objectives. While system analysis and design are not bound to any particular programming language, system implementation necessitates the use of a suitable high or low-level programming language. The successful compilation of the source code results in a functional application that aligns with our design specifications.

4. 2. Choice of programming language

The selection of the programming language for this work was guided by the specific features required. Among the features are server-side scripting capabilities and a robust database management system to handle the storage needs, as numerous modules and sections of this system relied on these functionalities. Given that this is a web-based application, it was essential to choose a scripting language capable of implementing all the required functionalities and working on the server side to meet the storage requirements.

PHP was the chosen scripting language for the implementation of this project. PHP, which stands for "PHP: Hypertext Pre-processor," is a widely used open-source scripting language known for its ability to handle web-based applications and data processing. When a PHP script is requested from the server, it is parsed, and the instructions within it are executed, resulting in the generation of HTML code that is then sent to the web browser. PHP is open source and can be executed on various servers, with Apache being one of the compatible options.

There are several compelling reasons for selecting PHP over alternative scripting languages, the reasons are as follows

1) Freedom from licensing restrictions: PHP is open source and not bound by the licensing constraints typically associated with commercial software products. Open-source software users enjoy greater freedom to modify, redistribute, and integrate the product into various applications. While there may be some variations in licensing terms among open-source variants, users generally have the liberty to adapt the software to their needs.
2) Inclusive development: PHP development teams are not restricted to a specific organization or entity. Individuals with an interest and aptitude for programming are encouraged to contribute to PHP projects. This openness fosters a diverse talent pool, ultimately enhancing the quality of the final product.

To effectively manage the data exchanged within the application, a database management system was essential. MySQL was selected as the database management system, primarily due to its strong compatibility with Apache servers and PHP. Since PHP operates as a server-side technology, the script code is processed by a web server. Apache 2.2.22 was chosen for this purpose. Apache's selection was driven by its widespread popularity, cross-platform compatibility (ability to run on various operating systems), and its tailored support for PHP and MySQL databases.

HTML (HyperText Markup Language) played a crucial role in crafting the user interface of the application, particularly in defining the page layout. HTML provides instructions for formatting and presenting text-based content on the World Wide Web. These instructions are seamlessly integrated into the content, as HTML commands serve as the foundational building blocks of web pages.

HTML adheres to platform-independence, making the choice of the machine largely inconsequential in web page rendering.

5. RESULTS

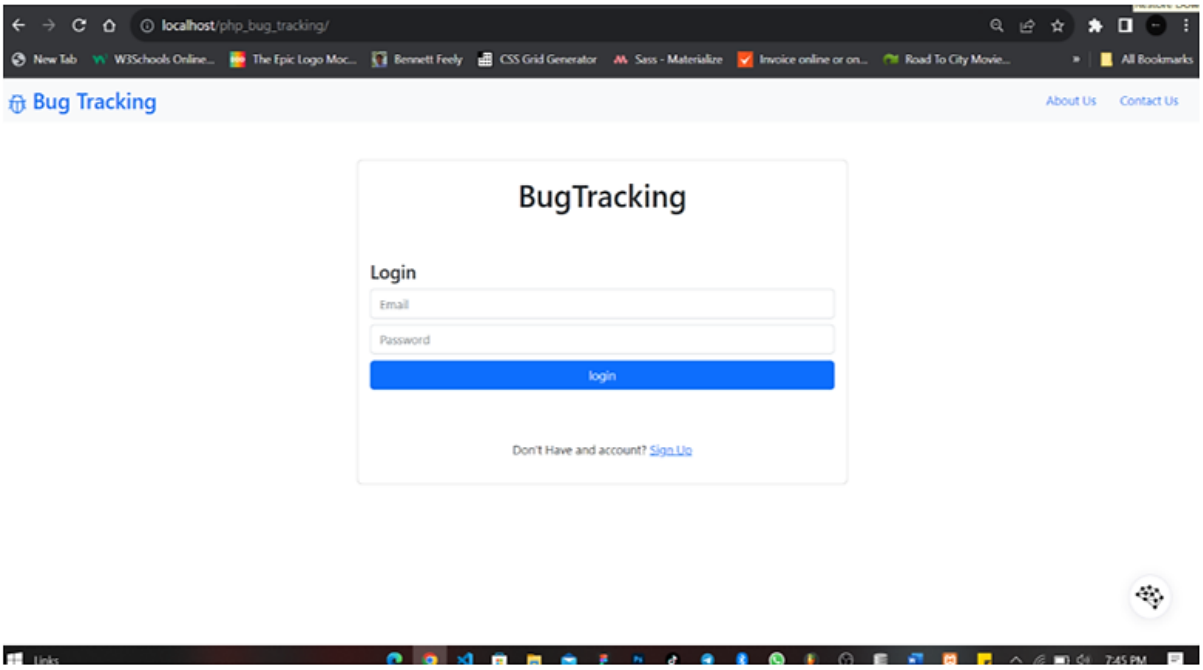


Figure 4. Graphics user interface for user login

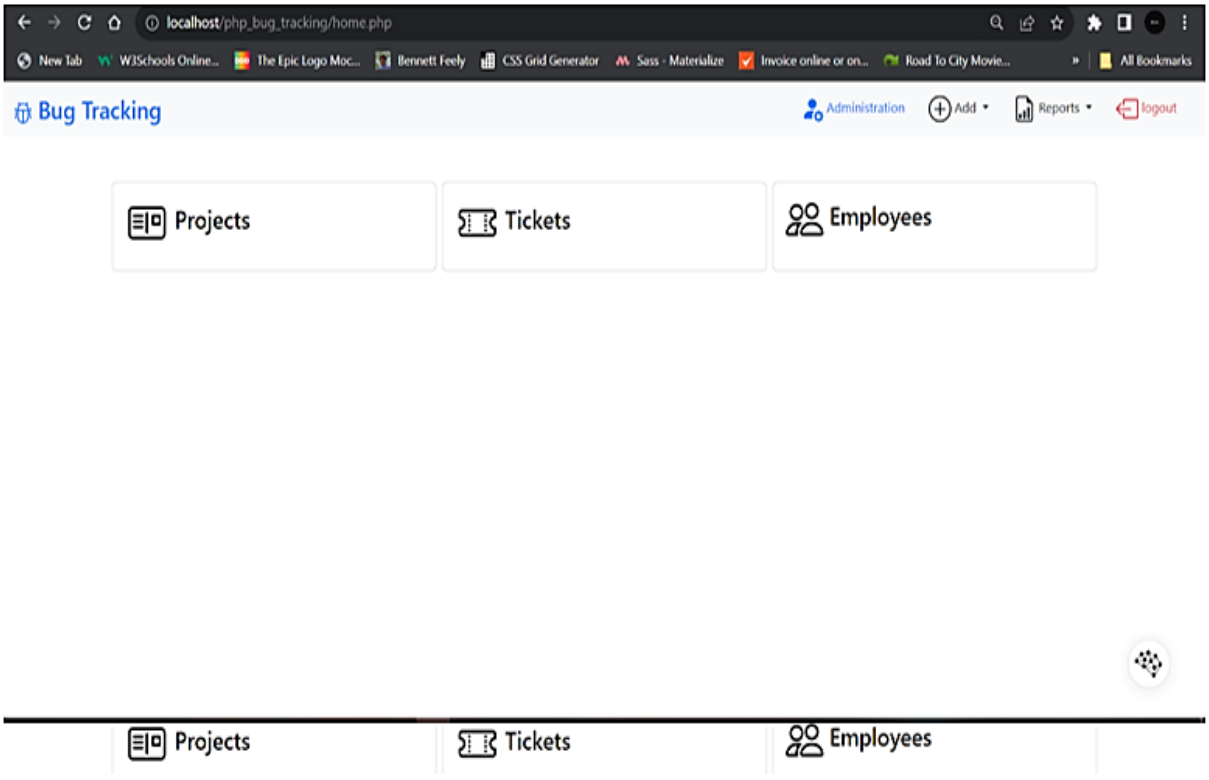


Figure 5. System home page

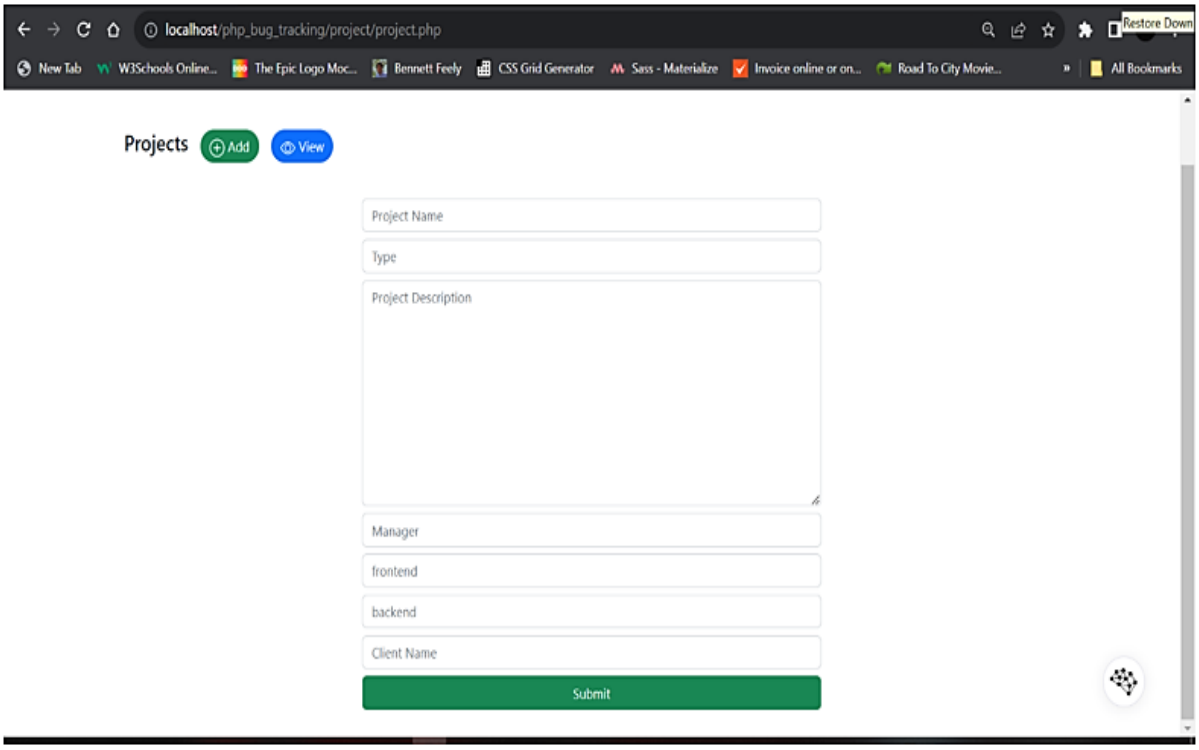


Figure 6. Inferface for adding projects for issue fixing

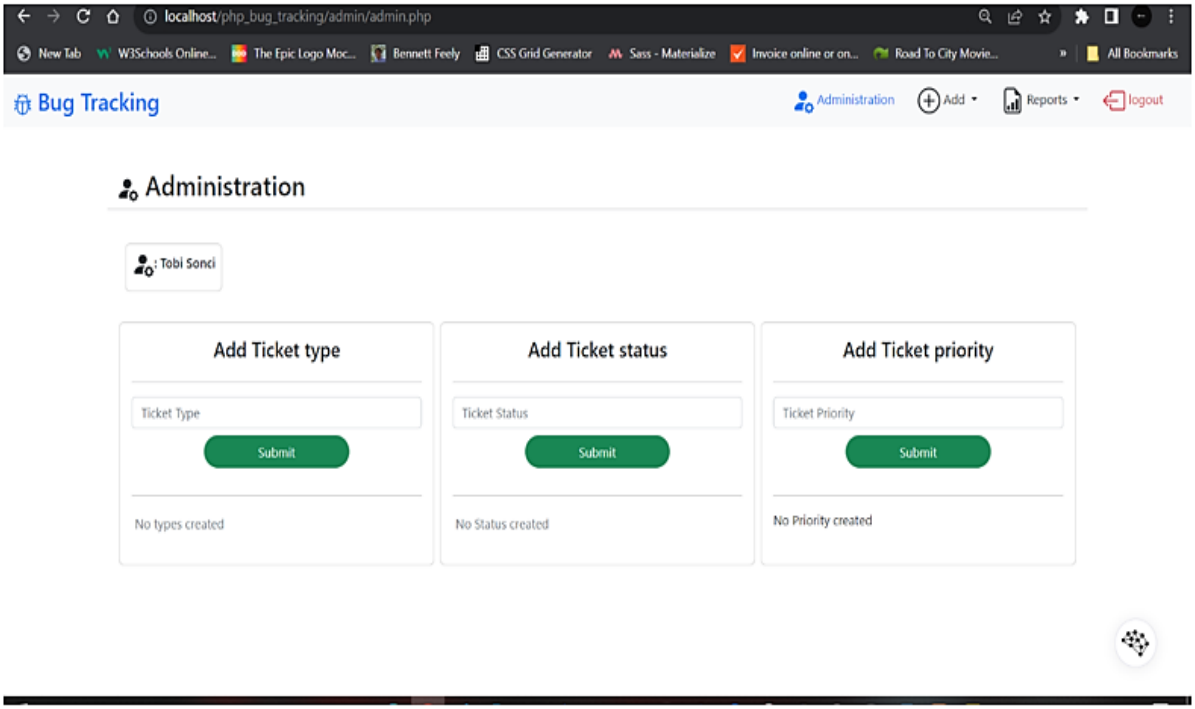


Figure 7. Interface for adding ticket type, status and priority

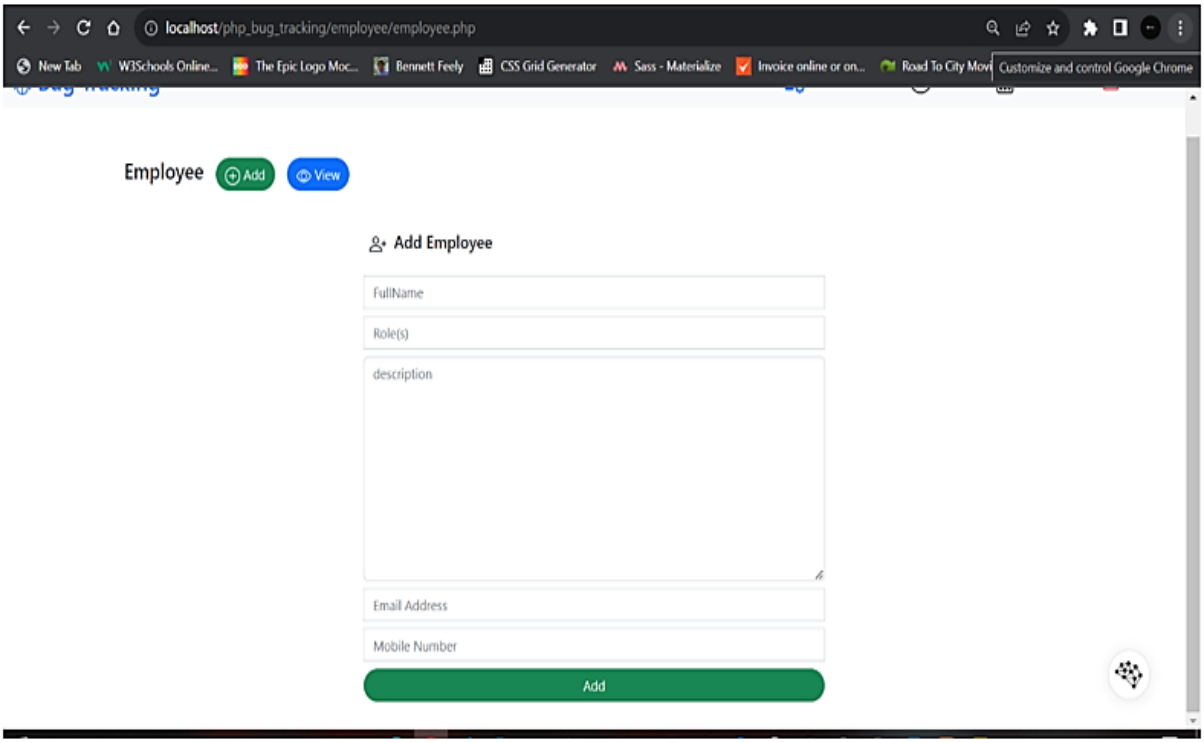
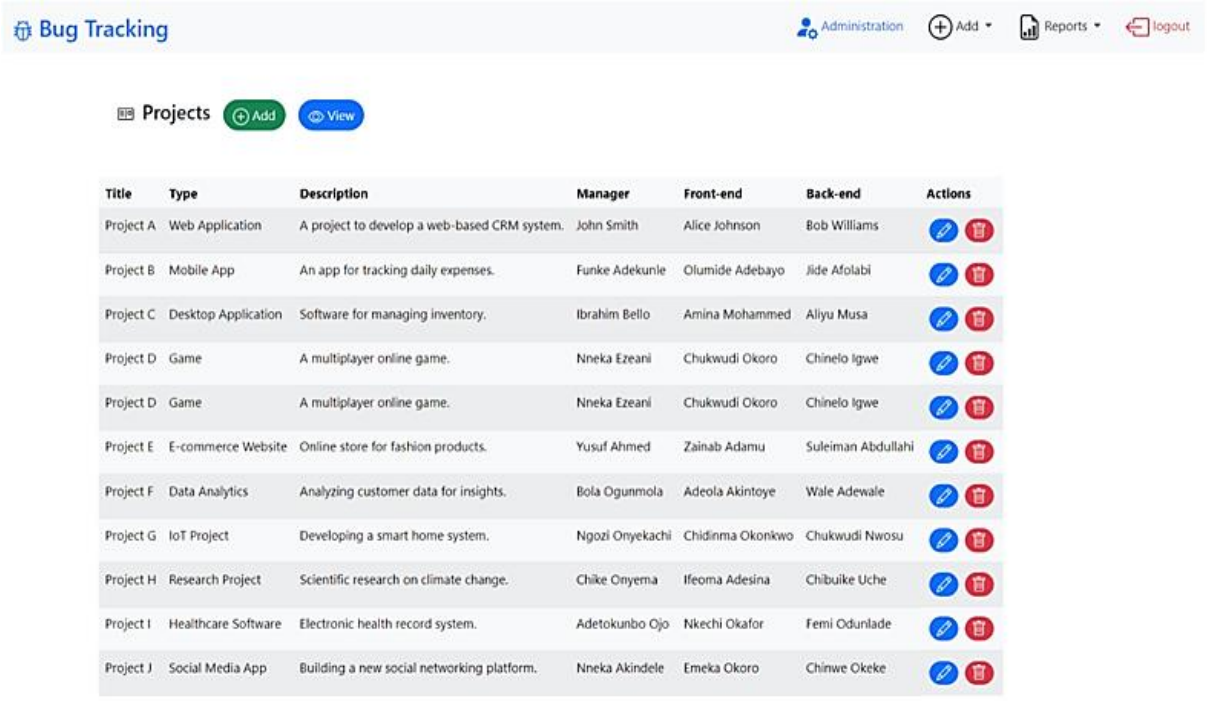
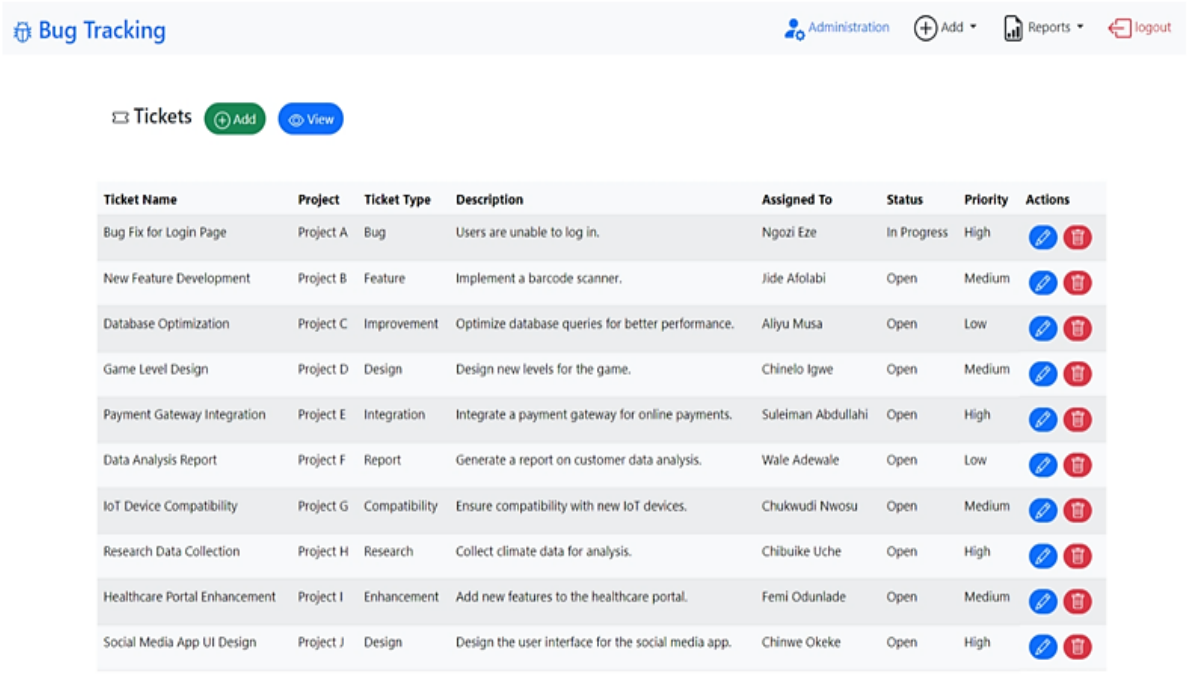


Figure 8. Interface for adding employee



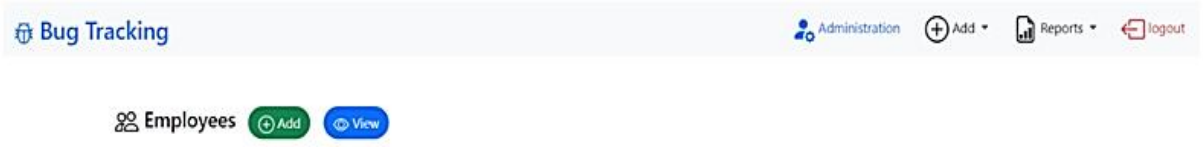
Title	Type	Description	Manager	Front-end	Back-end	Actions
Project A	Web Application	A project to develop a web-based CRM system.	John Smith	Alice Johnson	Bob Williams	
Project B	Mobile App	An app for tracking daily expenses.	Funke Adekunle	Olumide Adebayo	Jide Afolabi	
Project C	Desktop Application	Software for managing inventory.	Ibrahim Bello	Amina Mohammed	Aliyu Musa	
Project D	Game	A multiplayer online game.	Nneka Ezeani	Chukwudi Okoro	Chinelo Igwe	
Project D	Game	A multiplayer online game.	Nneka Ezeani	Chukwudi Okoro	Chinelo Igwe	
Project E	E-commerce Website	Online store for fashion products.	Yusuf Ahmed	Zainab Adamu	Suleiman Abdullahi	
Project F	Data Analytics	Analyzing customer data for insights.	Bola Ogunmola	Adeola Akintoye	Wale Adewale	
Project G	IoT Project	Developing a smart home system.	Ngozi Onyekachi	Chidinma Okonkwo	Chukwudi Nwosu	
Project H	Research Project	Scientific research on climate change.	Chike Onyema	Ifeoma Adesina	Chibuike Uche	
Project I	Healthcare Software	Electronic health record system.	Adetokunbo Ojo	Nkechi Okafor	Femi Odunlade	
Project J	Social Media App	Building a new social networking platform.	Nneka Akindele	Emeka Okoro	Chinwe Okeke	

Figure 9. Report interface on the projects under study



Ticket Name	Project	Ticket Type	Description	Assigned To	Status	Priority	Actions
Bug Fix for Login Page	Project A	Bug	Users are unable to log in.	Ngozi Eze	In Progress	High	
New Feature Development	Project B	Feature	Implement a barcode scanner.	Jide Afolabi	Open	Medium	
Database Optimization	Project C	Improvement	Optimize database queries for better performance.	Aliyu Musa	Open	Low	
Game Level Design	Project D	Design	Design new levels for the game.	Chinelo Igwe	Open	Medium	
Payment Gateway Integration	Project E	Integration	Integrate a payment gateway for online payments.	Suleiman Abdullahi	Open	High	
Data Analysis Report	Project F	Report	Generate a report on customer data analysis.	Wale Adewale	Open	Low	
IoT Device Compatibility	Project G	Compatibility	Ensure compatibility with new IoT devices.	Chukwudi Nwosu	Open	Medium	
Research Data Collection	Project H	Research	Collect climate data for analysis.	Chibuike Uche	Open	High	
Healthcare Portal Enhancement	Project I	Enhancement	Add new features to the healthcare portal.	Femi Odunlade	Open	Medium	
Social Media App UI Design	Project J	Design	Design the user interface for the social media app.	Chinwe Okeke	Open	High	

Figure 10. Report interface on bugs



Name	Role	Description	Email Address	Phone	Actions
Chinedu Okoye	Project Manager	Experienced project manager with a proven track record.	chinedu.okoye@example.com	0801-234-5678	
Ngozi Eze	Frontend Developer	Skilled in HTML, CSS, and JavaScript.	ngozi.eze@example.com	0802-345-6789	
Emeka Okafor	Backend Developer	Proficient in Python, Java, and database management.	emeka.okafor@example.com	0803-456-7890	
Funke Adekunle	Project Manager	Experienced project manager with a focus on mobile app development.	funke.adekunle@example.com	0804-567-8901	
Olumide Adebayo	Frontend Developer	Skilled in React and responsive design.	olumide.adebayo@example.com	0805-678-9012	
Jide Afolabi	Backend Developer	Proficient in Node.js, Ruby on Rails, and MongoDB.	jide.afolabi@example.com	0806-789-0123	
Ibrahim Bello	Project Manager	Experienced project manager in desktop application development.	ibrahim.bello@example.com	0807-890-1234	
Amina Mohammed	Frontend Developer	Skilled in Angular and user interface design.	amina.mohammed@example.com	0808-901-2345	
Aliyu Musa	Backend Developer	Proficient in C#, SQL, and server management.	aliyu.musa@example.com	0809-012-3456	
Nneka Ezeani	Project Manager	Experienced project manager in game development.	nneka.ezeani@example.com	0810-123-4567	

Figure 11. Report interface for info of experts who track bugs

The following Figures (4, 5, 6, 7, 8) show the sample implementation input snapshot of the proposed BugTracker. Figure 4 shows the graphics user interface for user login, Figure 5 shows the system home page, Figure 6 shows the inferface for adding projects for issue fixing, Figure 7 shows the interface for adding ticket type, status and priority, and Figure 8 shows the interface for adding employee. The following Figures (9, 10, 11) show the sample implementation output snapshot of the proposed BugTracker. Figure 9 shows the report interface on the projects under study, Figure 10 shows the report interface on bugs, and Figure 11 shows the report interface for info of experts who track bugs.

6. DISCUSSIONS

Evaluation was carried out on the system to assess the effectiveness and performance of the system in improving software quality through efficient bug detection, analysis, and resolution. To evaluate the system, a real-world software development environment was used. The system was implemented to facilitate the process of bug management and resolution. The evaluation process involved creating projects, tickets (representing bugs), and assigning them to employees based on their areas of expertise. Developers and managers, acting as employees, tested the assigned bugs and submitted their reports, which consisted of ticket name, project details, ticket type (indicating the type of bug), description, assigned personnel, status (such as “in progress” or “open”), and priority (such as “high,” “medium,” or “low”).

The evaluation results revealed several key insights. Firstly, the system effectively facilitated the creation and assignment of projects and tickets. The admin was able to create projects and assign specific bugs to employees who possessed the relevant expertise. This streamlined the bug management process, ensuring that bugs were appropriately allocated and addressed by the assigned personnel. Secondly, the system successfully captured relevant information in the bug reports submitted by developers and managers. The reports provided comprehensive details about each bug, including a clear description of the issue, the project it belonged to, and its assigned personnel. Additionally, the ticket type, status, and priority allowed for efficient tracking and prioritization of bug resolution activities.

Furthermore, the evaluation results indicated that the system supported effective bug testing and analysis. Developers and managers were able to thoroughly examine the assigned bugs, identify their root causes, and provide detailed analysis reports. This enabled a deeper understanding of the bugs and facilitated the development of appropriate bug-fixing strategies. The evaluation also demonstrated that the system improved communication and collaboration among team members. By providing a centralized platform for bug management, the system enabled efficient coordination and ensured that all stakeholders had access to relevant bug information. This enhanced collaboration and facilitated timely bug resolution.

In order to analyze and interpret the findings obtained from the implementation of the bug management system, and provide a deeper understanding of the implications and significance of the results, as well as explore their impact on the bug resolution process and overall software quality improvement, a comparative discussion was conducted. One of the key insights from the discussion is the effectiveness of the system in streamlining the bug management process as evaluated (Liu et al., 2018). By allowing the admin to create projects and assign bugs to employees based on their expertise, the system ensured that bugs were addressed by the most

suitable personnel. This allocation of bugs based on specialization optimized bug resolution efforts and contributed to improved efficiency in addressing software issues.

The comprehensive bug reports submitted by developers and managers also proved to be valuable. The detailed information provided, such as the ticket type, status, and priority, allowed for effective bug tracking and prioritization. This enhanced visibility and enabled project managers to allocate appropriate resources and prioritize bug fixes based on their impact and urgency. Moreover, the system facilitated effective bug testing and analysis. By assigning bugs to developers and managers, the system ensured that bugs underwent thorough examination and analysis. This resulted in a deeper understanding of the bugs' root causes, enabling the development of targeted bug-fixing strategies (Ni et al., 2020). The system thus supported more efficient bug resolution and contributed to the overall improvement of software quality.

Additionally, the centralized nature of the system improved communication and collaboration among team members. By providing a shared platform for bug management, the system promoted transparency and facilitated effective information sharing. This enabled team members to stay updated on bug statuses, collaborate on bug resolution efforts, and ensure timely bug fixes. The improved collaboration and communication fostered by the system contributed to a more streamlined and efficient bug resolution process. The evaluation results validate the effectiveness of the implemented bug management system in improving the bug resolution process and enhancing software quality. The system's ability to streamline bug allocation, capture comprehensive bug reports, facilitate bug testing and analysis, and promote collaboration among team members demonstrates its potential in addressing the challenges associated with bug management. These findings highlight the significance of the system in improving software development practices and contribute to the ongoing research and development in the field of bug management and software quality assurance.

7. CONCLUSIONS

The work reported in this paper has successfully identified a post-deployment bug management method for software systems, aimed at enhancing software stability and reliability, while simultaneously reducing development and management costs. The culmination of this effort is the creation of BugTracker, a web-based application designed to track and manage bugs in developed software systems. BugTracker serves as a comprehensive software bug repository that facilitates efficient bug tracking, removal, and optimization. It offers a valuable resource for software quality researchers, enabling them to refine software development processes, proactively address potential future bugs, and enhance overall software quality. Moreover, the work demonstrates the feasibility of enhancing a software system's stability, reducing development costs, and increasing user confidence through the implementation of a well-planned post-deployment strategy.

BugTracker played a pivotal role in managing software development and maintenance for organizations like BrainBench Technologies and DivaSoft. Additionally, it provided a rich bug-data repository that informed decisions aimed at improving the software development process. The iterative interaction and information logged in BugTracker benefited both developers and software testers by enhancing their understanding of software operations, including control and data flow responsible for failures. This, in turn, increased software

stability and reliability, ultimately leading to improved customer satisfaction and reduced operational costs for software development organizations.

References

[1] S. S. Alqahtani, J. Rilling, An ontology-based approach to automate tagging of software artifacts. *In 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (2017) (pp. 169-174).

[2] M. J. Lutz, J. F. Naveda, J. R. Vallino, Undergraduate software engineering. *Communications of the ACM* 57(8) (2014) 52-58

[3] P. Bian, B. Liang, W. Shi, J. Huang, Y. Cai, Nar-miner: discovering negative association rules from code for bug detection. *In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2018 October) (pp. 411-422).

[4] Budhiraja, K. Dutta, R. Reddy, M. Shrivastava, DWEN: deep word embedding network for duplicate bug report detection in software repositories. *In Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings* (2018 May) (pp. 193-194).

[5] X. Z. Chen, H. X. Ding, J. Zhang, W. A. N. G. Yang, G. Zhang, Y. N. Wang, An artificial intelligence (AI) defect detection technology based on software behavior decision tree. *2018 International Conference on Computer, Communication and Network Technology (CCNT 2018)* (pp. 113-126).

[6] A. Elmishali, R. Stern, M. Kalech, An artificial intelligence paradigm for troubleshooting software bugs. *Engineering Applications of Artificial Intelligence* 69 (2018) 147-156

[7] Elmishali, R. Stern, M. Kalech, Debguer: A tool for bug prediction and diagnosis. *In Proceedings of the AAAI Conference on Artificial Intelligence* 33 (01) 9446-9451

[8] P. C. Amusuo, A. Sharma, S. R. Rao, A. Vincent, J. C. Davis, Reflections on software failure analysis. *In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2022 November) (pp. 1615-1620).

[9] A. Hammouri, M. Hammad, M. Alnabhan, F. Alsarayrah, Software bug prediction using machine learning approach. *International Journal of Advanced Computer Science and Applications* 9(2) (2018).

[10] Hindle, C. Onuczko, Preventing duplicate bug reports by continuously querying bug reports. *Empirical Software Engineering* 24(2) (2019) 902–936

[11] D. Šmite, C. Wohlin, Z. Galviņa, R. Prikladnicki, An empirically based terminology and taxonomy for global software engineering. *Empirical Software Engineering* 19 (2014) 105-153

[12] R. Kumar, D. L. Gupta, Software bug prediction system using neural network. *European Journal of Advances in Engineering and Technology* 3(7) (2016) 78-84.

[13] J. Li, P. He, J. Zhu, M. R. Lyu, Software defect prediction via convolutional neural network. *In 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)* (2017 July) (pp. 318-328).

[14] X. Li, J. Chen, Z. Lin, L. Zhang, Z. Wang, M. Zhou, W. Xie, A mining approach to obtain the software vulnerability characteristics. *In 2017 fifth international conference on advanced cloud and big data (CBD)* (2017 August) (pp. 296-301).

[15] S. Ma, Y. Liu, W. C. Lee, X. Zhang, A. Grama, MODE: automated neural network model debugging via state differential analysis and input selection. *In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (2018 October) (pp. 175-186).

[16] Manjula, L. Florence, Deep neural network based hybrid approach for software defect prediction using software metrics. *Cluster Computing* 22(4) (2019) 9847-9863.

[17] A. F. Otoom, D. Al-Shdaifat, M. Hammad, E. E. Abdallah, Severity prediction of software bugs. *In 2016 7th International Conference on Information and Communication Systems (ICICS)* (2016 April) (pp. 92-95).

[18] S. K. Pandey, R. B. Mishra, A. K. Tripathi, BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems with Applications* 144 (2020) 113085

[19] S. K. Pandey, R. B. Mishra, A. K. Triphathi, Software bug prediction prototype using Bayesian network classifier: A comprehensive model. *Procedia Computer Science* 132 (2018) 1412-1421

[20] F. Ricca, P. Tonella, Analysis and testing of web applications. *In IEEE Proceedings of the 23rd International Conference on Software Engineering ICSE 2001* (2001 May) (pp. 25-34).

[21] K. Schekotihin, P. Rodler, W. Schmid, M. Horridge, T. Tudorache, Test-driven ontology development in Protégé. *In Proceedings of the 9th International Conference on Biological Ontology (ICBO 2018)* Corvallis Oregon USA (pp. 1-6).

A. Sheneamer, J. Kalita, A survey of software clone detection techniques. *International Journal of Computer Applications* 137(10) (2016) 1-21.

[22] L. Anthony, Writing research article introductions in software engineering: How accurate is a standard model?. *IEEE Transactions on Professional Communication* 42(1) (1999) 38-46

[23] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, H. Shah, A survey on bug prioritization. *Artificial Intelligence Review* 47(2) (2017) 145–180

[24] V. Kulkarni, A. Kolhe, J. Kulkarni, Intelligent software engineering: the significance of artificial intelligence techniques in enhancing software development lifecycle processes. *In International Conference on Intelligent Systems Design and Applications* Cham: Springer International Publishing (2021 December) (pp. 67-82).

[25] T. Xie, Intelligent software engineering: Synergy between AI and software engineering. *In Proceedings of the 11th Innovations in Software Engineering Conference* (2018) (pp. 1-1).

[26] G. Rodríguez-Pérez, G. Robles, A. Serebrenik, A. Zaidman, D. M. Germán, J. M. Gonzalez-Barahona, How bugs are born: a model to identify how bugs are introduced in software components. *Empirical Software Engineering* 25 (2020) 1294-1340

[27] R. W. Bello, O. M. Moradeyo, A. S. Olaniyan, Web Request Level Protection of Cyber Applications against Threats and Attacks. *International Journal of Scientific Engineering and Science* 2(1) (2018) 52-56

[28] J. Bader, A. Scott, M. Pradel, S. Chandra, Getafix: Learning to fix bugs automatically. *Proceedings of the ACM on Programming Languages* 3(OOPSLA) (2019) 1-27.

[29] S. Kim, T. Zimmermann, K. Pan, E. James Jr, Automatic identification of bug-introducing changes. *In 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)* (2006 September) (pp. 81-90).

[30] R. W. Bello, F. N. Otobo, Stored-knowledge based troubleshooting and diagnosing system. *International Journal of Scientific Engineering and Science* 2(5) (2018) 19-24

[31] Z. Ni, B. Li, X. Sun, T. Chen, B. Tang, X. Shi, Analyzing bug fix for automatic bug cause classification. *Journal of Systems and Software* 163 (2020) 110538.

[32] K. Liu, D. Kim, T. F. Bissyandé, S. Yoo, Y. Le Traon, Mining fix patterns for findbugs violations. *IEEE Transactions on Software Engineering* 47(1) (2018) 165-188.

[33] M. Perkusich, L. C. e Silva, A. Costa, F. Ramos, R. Saraiva, A. Freire, ..., A. Perkusich, Intelligent software engineering in the context of agile software development: A systematic literature review. *Information and Software Technology* 119 (2020) 106241