Cairo University
Faculty of Engineering
Department of Computer Engineering

BugWhiz

A Graduation Project Report Submitted

to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree

of

Bachelor of Science in Computer Engineering.

Presented by

Karim Mahmoud Kamal
Karim Mohamed
Mustafa Mahmoud
Donia Gameel

Supervised by

Dr. Yahia Zakaria

July, 2024

Abstract

The Automated Bug Triaging System, BugWhiz, is designed to streamline and enhance the efficiency of handling and prioritizing bug reports in software development by integrating advanced AI and machine learning techniques. Traditional methods of bug triaging are often prone to human error and result in delays that can affect the overall software development lifecycle. BugWhiz aims to address these challenges by automating the critical aspects of bug triaging, ensuring a more efficient and accurate process.

In software development, managing bug reports effectively is crucial for maintaining software quality and user satisfaction. However, the current manual process for handling bug reports is often inefficient and error-prone, leading to several issues. Development teams often receive a large number of bug reports, making it challenging to sort, categorize, and prioritize them manually. Manual categorization of bug reports can be inconsistent, as different team members may interpret and categorize issues differently. Determining the priority of each bug based on severity, user impact, and business priorities is time-consuming and may not always be accurate. Assigning bugs to the most suitable developers manually can be inefficient, as it may not consider the developer's expertise. Identifying and merging duplicate bug reports manually is challenging, resulting in multiple developers working on the same issue, and wasting time. Developers may forget or overlook assigned bugs due to a lack of reminders, causing delays in bug resolution and impacting software quality.

The primary objectives of BugWhiz are to automate the categorization of bug reports using Natural Language Processing (NLP) to ensure consistent and accurate categorization, prioritize bugs based on severity, user impact, and business priorities, and assign bugs to the most suitable developers based on historical data and expertise. Additionally, BugWhiz aims to detect duplicate bug reports to prevent redundant efforts, send automated notifications to relevant stakeholders about the status of bug reports, and provide regular reminders to developers about their assigned tasks, enabling continuous improvement in the development process. Simplifying the bug reporting process for users ensures detailed and structured bug information for developers.

BugWhiz automates the process of handling and prioritizing bug reports by leveraging AI and machine learning. It automatically categorizes bug reports using NLP, ensuring consistent and accurate categorization. The system prioritizes bugs based on severity, user impact, and business priorities, ensuring critical issues are addressed promptly. It assigns bugs to the most suitable developers based on historical data and expertise, optimizing resolution times. BugWhiz also detects and

merges duplicate bug reports, preventing redundant work and saving resources. The system sends automated notifications to relevant stakeholders about the status of bug reports, ensuring timely communication. It provides regular reminders to developers about their assigned tasks, reducing delays in bug resolution.

The implementation of BugWhiz resulted in a fully functional software tool with several key features. The software packages include the Automated Bug Triaging System Application with backend services that handle the logic for bug triaging, including NLP models, machine learning algorithms, and data processing. Bug categorization is achieved through NLP models trained to classify bug reports into predefined categories based on their descriptions and keywords. Machine learning algorithms assess the severity, user impact, and business priorities to prioritize bugs. Developer assignment algorithms use historical data and developer expertise to assign bugs to the most suitable developers. NLP and ML models identify and merge duplicate bug reports to avoid redundant work. The front-end interface provides user interfaces for both developers and users to interact with the system, including dashboards, submission forms, and notification displays. A structured database stores bug reports, user data, developer information, historical data, and analytics results. Documentation includes user documentation with a user manual detailing the system’s features and developer documentation with API descriptions, system architecture, and setup instructions.

The development of BugWhiz utilized modern tools and technologies, including Python for AI and machine learning models, JavaScript for front-end development, ExpressJS for the web application framework, ReactJS for the user interface, MongoDB for data storage,scikit-learn for machine learning models, and AWS for hosting and scalability. Extensive testing was conducted, including unit testing, integration testing, performance testing, and user acceptance testing, to ensure the robustness and reliability of BugWhiz. The testing results demonstrated substantial improvements in bug triaging efficiency, with an average reduction of 40% in triaging time and a 30% increase in accuracy compared to manual methods.

BugWhiz represents a significant advancement in bug tracking and management. By automating the triaging process, not only improves efficiency and accuracy but also enables development teams to deliver higher-quality software in a shorter timeframe. Enhanced software quality, increased developer productivity, improved user satisfaction, data-driven decision-making, market competitiveness, and cost savings are some of the key impacts of the solution.

الملخص

تم تصميم نظام فرز الأخطاء الآلي، BugWhiz، لتبسيط وتعزيز كفاءة التعامل مع تقارير الأخطاء وتحديد أولوياتها في تطوير البرمجيات من خلال دمج تقنيات الذكاء الاصطناعي والتعلم الآلي المتقدمة. غالبًا ما تكون الأساليب التقليدية لفرز الأخطاء كثيفة العمالة، وعرضة للخطأ البشري، وتؤدي إلى تأخيرات يمكن أن تؤثر على دورة حياة تطوير البرامج بشكل عام. يهدف BugWhiz إلى معالجة هذه التحديات من خلال أتمتة الجوانب المهمة لفرز الأخطاء، مما يضمن عملية أكثر كفاءة ودقة.

في تطوير البرمجيات، تعد إدارة تقارير الأخطاء بشكل فعال أمرًا بالغ الأهمية للحفاظ على جودة البرامج ورضا المستخدم. ومع ذلك، فإن العملية اليدوية الحالية للتعامل مع تقارير الأخطاء غالبًا ما تكون غير فعالة وعرضة للأخطاء، مما يؤدي إلى العديد من المشكلات. غالبًا ما تتلقى فرق التطوير عددًا كبيرًا من تقارير الأخطاء، مما يجعل من الصعب فرزها وتصنيفها وتحديد أولوياتها يدويًا. يمكن أن يكون التصنيف اليدوي لتقارير الأخطاء غير متسق، حيث قد يفسر أعضاء الفريق المختلفون المشكلات ويصنفونها بشكل مختلف. إن تحديد أولوية كل خطأ بناءً على مدى خطورته وتأثير المستخدم وأولويات العمل يستغرق وقتًا طويلاً وقد لا يكون دقيقًا دائمًا. قد يكون تعيين الأخطاء للمطورين الأكثر ملاءمة يدويًا أمرًا غير فعال، لأنه قد لا يأخذ في الاعتبار خبرة المطور. يعد تحديد تقارير الأخطاء المكررة ودمجها يدويًا أمرًا صعبًا، مما يؤدي إلى عمل العديد من المطورين على نفس المشكلة، مما يؤدي إلى إضاعة الوقت. قد ينسى المطورون الأخطاء المعينة أو يتجاهلونها بسبب نقص التذكيرات، مما يتسبب في تأخير حل الأخطاء والتأثير على جودة البرامج. بدون التحليل الآلي، سيكون من الصعب الحصول على رؤى حول أنماط الأخطاء وأوقات الحل والمقاييس الأخرى، مما يحد من القدرة على تحسين عملية التطوير.

تتمثل الأهداف الأساسية لـ BugWhiz في أتمتة تصنيف تقارير الأخطاء باستخدام معالجة اللغات الطبيعية (NLP) لضمان التصنيف المتسق والدقيق، وتحديد أولويات الأخطاء بناءً على مدى خطورتها وتأثير المستخدم وأولويات العمل، وتعيين الأخطاء للمطورين الأكثر ملاءمة بناءً على البيانات والخبرات التاريخية. بالإضافة إلى ذلك، يهدف BugWhiz إلى اكتشاف تقارير الأخطاء المكررة لمنع الجهود المتكررة، وإرسال إشعارات تلقائية إلى أصحاب المصلحة المعنيين حول حالة تقارير الأخطاء، وتقديم تذكيرات منتظمة للمطورين حول المهام المعينة لهم، وتقديم تحليلات ورؤى مفصلة حول أنماط الأخطاء وأوقات الحل. تمكين التحسين المستمر في عملية التطوير. إن تبسيط عملية الإبلاغ عن الأخطاء للمستخدمين يضمن الحصول على معلومات تفصيلية ومنظمة عن الأخطاء للمطورين.

يقوم BugWhiz بأتمتة عملية التعامل مع تقارير الأخطاء وتحديد أولوياتها من خلال الاستفادة من الذكاء الاصطناعي والتعلم الآلي. يقوم تلقائيًا بتصنيف تقارير الأخطاء باستخدام البرمجة اللغوية العصبية (NLP)، مما يضمن تصنيفًا متسقًا ودقيقًا. يقوم النظام بإعطاء الأولوية للأخطاء بناءً على مدى خطورتها وتأثير المستخدم وأولويات العمل، مما يضمن معالجة المشكلات المهمة على الفور. فهو يقوم بتعيين الأخطاء للمطورين الأكثر ملاءمة بناءً على البيانات والخبرة التاريخية، مما يؤدي إلى تحسين أوقات الحل. يكتشف BugWhiz أيضًا تقارير الأخطاء المكررة ويدمجها، مما يمنع العمل الزائد ويوفر الموارد. يرسل النظام إشعارات آلية إلى أصحاب المصلحة المعنيين حول حالة تقارير الأخطاء، مما يضمن التواصل في الوقت المناسب. فهو يوفر تذكيرات منتظمة للمطورين حول المهام الموكلة إليهم، مما يقلل التأخير في حل الأخطاء. يتم تقديم تحليلات ورؤى تفصيلية حول أنماط الأخطاء وأوقات الحل، مما يتيح التحسين المستمر في عملية التطوير.

أدى تطبيق BugWhiz إلى ظهور أداة برمجية كاملة الوظائف مع العديد من الميزات الرئيسية. تشتمل حزم البرامج على تطبيق نظام فرز الأخطاء الآلي مع خدمات الواجهة الخلفية التي تتعامل مع منطق فرز الأخطاء، بما في ذلك نماذج البرمجة اللغوية العصبية (NLP)، وخوارزميات التعلم الآلي، ومعالجة البيانات. يتم تحقيق تصنيف الأخطاء من خلال نماذج البرمجة اللغوية العصبية (NLP) المدربة على تصنيف تقارير الأخطاء إلى فئات محددة مسبقًا بناءً على الأوصاف والكلمات الرئيسية الخاصة بها. تقوم خوارزميات التعلم الآلي بتقييم مدى خطورة الأخطاء وتأثيرها على

المستخدم وأولويات العمل لتحديد أولويات الأخطاء. تستخدم خوارزميات تعيين المطورين البيانات التاريخية وخبرة المطورين لتعيين الأخطاء للمطورين الأكثر ملاءمة. تعمل نماذج البرمجة اللغوية العصبية (NLP) والتعلم الآلي (ML) على تحديد تقارير الأخطاء المكررة ودمجها لتجنب العمل الزائد عن الحاجة. توفر واجهة الواجهة الأمامية واجهات مستخدم لكل من المطورين والمستخدمين للتفاعل مع النظام، بما في ذلك لوحات المعلومات ونماذج الإرسال وشاشات الإشعارات. تقوم قاعدة البيانات المنظمة بتخزين تقارير الأخطاء وبيانات المستخدم ومعلومات المطور والبيانات التاريخية ونتائج التحليلات. تتضمن الوثائق وثائق المستخدم مع دليل المستخدم الذي يعرض تفاصيل ميزات النظام ووثائق المطور مع أوصاف واجهة برمجة التطبيقات (API)، وهندسة النظام، وتعليمات الإعداد. تتميز أدوات التحليلات وإعداد التقارير بلوحة تحكم تحليلية تحتوي على تقارير مرئية وأدوات إنشاء تقارير مخصصة.

استخدم تطوير BugWhiz الأدوات والتقنيات الحديثة، بما في ذلك Python للذكاء الاصطناعي ونماذج التعلم الآلي، وJavaScript لتطوير الواجهة الأمامية، وExpressJS لإطار تطبيق الويب، وReactJS لواجهة المستخدم، وMongoDB لتخزين البيانات، وscikit-learn لـ نماذج التعلم الآلي، وAWS للاستضافة وقابلية التوسع. تم إجراء اختبارات واسعة النطاق، بما في ذلك اختبار الوحدة، واختبار التكامل، واختبار الأداء، واختبار قبول المستخدم، لضمان قوة وموثوقية BugWhiz. أظهرت نتائج الاختبار تحسينات كبيرة في كفاءة فرز الأخطاء، مع انخفاض متوسط قدره 40% في وقت الفرز وزيادة بنسبة 30% في الدقة مقارنة بالطرق اليدوية.

ومن خلال أتمتة عملية الفرز، لا يؤدي ذلك إلى تحسين الكفاءة والدقة فحسب، بل يمكّن أيضًا فرق التطوير من تقديم برامج عالية الجودة في إطار زمني أقصر. تعد جودة البرامج المحسنة، وزيادة إنتاجية المطورين، وتحسين رضا المستخدمين، واتخاذ القرارات المستندة إلى البيانات، والقدرة التنافسية في السوق، وتوفير التكاليف بعضًا من التأثيرات الرئيسية للحل.

ACKNOWLEDGMENT

We extend our heartfelt appreciation to all who have contributed to the successful completion of this graduation project. Our sincere gratitude goes to our esteemed Doctor, Dr. Yahia Zakaria, whose invaluable guidance, mentorship, and steadfast support have been pivotal throughout our five-year academic journey. His expertise and commitment to excellence have significantly shaped our skills and nurtured our intellectual growth.

We also acknowledge with gratitude the distinguished doctors and teaching assistants who generously shared their wisdom and expertise with us. Their lectures, seminars, and interactive discussions have enriched our understanding, expanded our horizons, and ignited our curiosity to delve deeper into our field of study. Their dedication to educating and inspiring young minds has been truly motivating, and we owe them a debt of gratitude for their contributions to our education.

Lastly, we express our heartfelt thanks to our families and loved ones for their unwavering support, patience, and understanding. Their encouragement, sacrifices, and belief in our abilities have been the driving force behind our pursuit of knowledge and personal development.

In conclusion, the successful completion of this graduation project would not have been possible without the contributions of these individuals and many others who have influenced our academic journey. Their collective efforts have molded our intellectual growth and equipped us to face the challenges ahead with confidence. We sincerely appreciate their guidance, mentorship, and unwavering support.

With sincere gratitude,

Karim Mahmoud Kamal, Mustafa Mahmoud, Karim Mohamed, and Donia Gameel

Faculty of Engineering, Cairo University

July, 2023

Table of Contents

List of Figures

List of Tables

List of Abbreviation

API: Application Programming Interface
AWS: Amazon Web Services
CSS: Cascading Style Sheets
DeBERTa: Decoding-enhanced BERT with Disentangled Attention
HTML: Hypertext Markup Language
JS: JavaScript
ML: Machine Learning
NLP: Natural Language Processing
NoSQL: Not Only SQL (Structured Query Language)
SVM: Support Vector Machine
TF-IDF: Term Frequency-Inverse Document Frequency
UI: User Interface
SVM: Support Vector Machine

List of Symbols

d: Document
t: Term
TF(t, d): Term Frequency of term t in document d
IDF(t): Inverse Document Frequency of term t
TF-IDF(t, d): Term Frequency-Inverse Document Frequency of term t in document d
cosine similarity: Measure of similarity between two non-zero vectors of an inner product space
N: Total number of documents
DF(t): Document Frequency of term t

Contacts

Team Members

Name	Email	Phone Number
Karim Mahmoud Kamal	karimmk2210@gmail.com	+2 01113162153
Mustafa Mahmoud Hamada	dev.mustafa.mahmoud@gmail.com	+2 01121366579
Karim Mohamed	karim.elsayed01@eng-st.cu.edu.eg	+2 01156201693
Donia Gameel	doniagameel34@gmail.com	+2 01116371385

Supervisor

Name	Email	Number
Dr. Yahia Zakaria	yahiazakaria13@gmail.com	+2 01117297303

Chapter 1: Introduction

1.1. Motivation and Justification

The rising number of bug reports in software development poses significant challenges, making it difficult to handle them efficiently. Manual methods for sorting and prioritizing bugs often lead to delays, misuse of resources, and problems with fixing the most important issues quickly. This affects how quickly software problems are solved and can impact user satisfaction negatively. BugWhiz aims to solve these problems by using advanced AI and language processing technologies to automate the process of managing bug reports.

BugWhiz has clear goals: to speed up how bugs are dealt with by automatically sorting them based on how severe they are, their impact, and past data. By doing this, BugWhiz helps developers focus more on fixing critical bugs promptly and improving overall software quality. This automation not only makes the bug-handling process faster but also ensures that resources are used more effectively within development teams.

Implementing BugWhiz brings several benefits to software development. It eliminates the need for developers to spend time on repetitive tasks, such as manually sorting through bug reports. It also ensures that critical issues are identified and resolved quickly, reducing the risk of important bugs being overlooked. BugWhiz is flexible and can be used in various types of software projects, whether they are small applications or large-scale enterprise systems. By speeding up bug fixes, BugWhiz aims to improve user satisfaction and help software products succeed in a competitive market.

BugWhiz represents a significant advancement in how software bugs are managed. By automating and improving critical processes, BugWhiz helps development teams deliver better-quality software more efficiently. This, in turn, leads to happier users and more successful software products overall.

1.2. Project Objectives and Problem Definition

1.2.1. Project Objectives

The primary objectives of BugWhiz are:

- **Automated Categorization**:
    - Utilize Natural Language Processing to analyze bug descriptions and categorize them into predefined categories such as Front end, Back end, Documentation, and security.
- **Priority Assignment**:
    - Implement AI algorithms to prioritize bugs based on severity, user impact, and business priorities, ensuring critical issues are addressed promptly.
- **Developer Assignment**:
    - Leverage historical data and developer expertise to automatically assign bugs to the most suitable developers, optimizing the resolution process.
- **Duplicate Detection**:
    - Detect duplicate bug reports to prevent redundant efforts, streamlining the workflow for development teams.
- **Notification System**:
    - Send automated notifications to relevant stakeholders upon bug assignment, resolution, or updates, keeping all parties informed.
- **Simplified Bug Reporting**:
    - Make it easier for users to submit detailed and structured bug reports, ensuring that developers have all the information they need to address issues effectively.

1.2.2. Problem Definition

In software development, managing bug reports effectively is crucial for maintaining software quality and user satisfaction. However, the current manual process for handling bug reports is often inefficient and error-prone, leading to several issues:

- **High Volume of Bug Reports**:
    - Development teams often receive a large number of bug reports, making it challenging to sort, categorize, and prioritize them manually.
- **Inconsistent Categorization**:
    - Manual categorization of bug reports can be inconsistent, as different team members may interpret and categorize issues differently.
- **Inefficient Prioritization**:

  - Determining the priority of each bug based on severity, user impact, and business priorities is time-consuming and may not always be accurate.
- **Suboptimal Developer Assignment**:
  - Assigning bugs to the most suitable developers manually can be inefficient, as it may not consider the developer's expertise.
- **Duplicate Bug Reports**:
  - Identifying and merging duplicate bug reports manually is challenging, resulting in multiple developers working on the same issue, and wasting time.
- **Delayed Developer Response**:
  - Developers may forget or overlook assigned bugs due to a lack of reminders, causing delays in bug resolution and impacting software quality.

1.3. Project Outcomes

The implementation of BugWhiz resulted in a fully functional software tool with the following features:

- ➢ **Software Packages**
  - **Automated Bug Triaging System Application**
    - ■ Backend Services:
      - APIs and services that handle the logic for bug triaging, including NLP models, machine learning algorithms, and data processing.
    - ■ Bug Categorization:
      - NLP models are trained to classify bug reports into predefined categories based on their descriptions and keywords.
    - ■ Prioritization:
      - Machine learning algorithms that assess the severity, user impact, and business priorities to prioritize bugs.
    - ■ Developer Assignment:
      - Algorithms that use historical data and developer expertise to assign bugs to the most suitable developers.
    - ■ Duplicate Detection:
      - NLP and ML models that identify and merge duplicate bug reports to avoid redundant work.
    - ■ Frontend Interface:

      - User interfaces for both developers and users to interact with the system, including dashboards, submission forms, and notification displays.
    - Database:
      - A structured database to store bug reports, user data, developer information, historical data, and analytics results.

- **Documentation**
  - **User Documentation**
    - User Manual:
      - Detailed instructions on how to use the system's features, including bug submission, tracking, and notification settings.
  - **Developer Documentation**
    - API Documentation:
      - Detailed descriptions of the APIs, including endpoints, request/response formats, and usage examples.
    - System Architecture Document:
      - Diagrams and explanations of the system's architecture, including the backend, frontend, and database structures.
    - Setup and Installation Guide:
      - Step-by-step instructions for setting up the development environment, deploying the system, and integrating with other tools.

1.4. Document Organization

This report is organized into several chapters, each addressing a specific aspect of the project. The following is a quick description of each chapter:

1. **Chapter 1: Introduction**
   - This chapter provides an introduction to the project, its background, and its objectives. It outlines the motivation and justification for the project, explaining why BugWhiz is needed and the benefits it brings to the software development process.
2. **Chapter 2: Market Feasibility Study**
   - This chapter surveys the related market to the project, including similar tools and platforms. It discusses the drawbacks of existing solutions, highlighting the need for BugWhiz. The analysis provides a foundation

for understanding the competitive landscape and the unique value proposition of BugWhiz.

3. **Chapter 3: Literature Survey**
    - This chapter consists of two parts. The first part provides the necessary engineering and non-engineering backgrounds to understand the project, covering relevant theories, algorithms, and technologies. The second part presents a short literature review of some publications related to the project, offering insights into previous research and developments in the field of automated bug triaging.
4. **Chapter 4: System Design and Architecture**
    - This chapter describes the project in full detail, answering the questions of "what has been done?" and "how it has been done?" It discusses the steps, scientific approaches, and methodologies used to realize the project. The chapter presents a logical flow from the overall block diagram to coarse modules and fine modules, providing a comprehensive view of the system design and architecture.
5. **Chapter 5: System Testing and Verification**
    - This chapter explains the steps taken to ensure the correct realization of project outcomes. It describes the testing setup, strategy, and environment used. The chapter discusses components testing efforts and integrated system testing, highlighting and discussing the results from different testing scenarios. This section ensures that the system functions as intended and meets the project’s objectives.
6. **Chapter 6: Conclusions and Future Work**
    - This chapter summarizes the project, its features, and limitations. It provides directions for future work, suggesting possible extensions and enhancements that can be made to BugWhiz. This section aims to guide subsequent research and development efforts, ensuring continuous improvement and adaptation to evolving user needs and technological advancements.

By following this organization, the report presents a comprehensive overview of the project, from market visibility and literature survey to system design, testing, and conclusions. The document aims to provide a clear understanding of the project's objectives, methodology, outcomes, and potential for future development.

Chapter 2: Market Feasibility Study

In this chapter, we survey the market related to BugWhiz, our Automated Bug Triaging System. We discuss other similar tools and platforms, analyzing their features, strengths, and limitations. This analysis helps justify the need for BugWhiz by highlighting its unique advantages and addressing the gaps in existing solutions. The chapter aims to provide a comprehensive overview of the competitive landscape and demonstrate the value BugWhiz brings to the market.

2.1 Targeted Customers

The intended customers for BugWhiz are diverse, ranging from large enterprises to individual developers. The following are the key customer segments that will benefit from the system:

2.1.1 Software Development Companies

Profile:

- Companies ranging from small startups to large enterprises that develop software products and services.

Needs and Benefits:

- **Efficient Bug Management:**
  - Software development companies often deal with a high volume of bug reports. BugWhiz automates the categorization, prioritization, and assignment of these reports, significantly reducing the manual effort required.
- **Improved Productivity:**
  - By streamlining the bug-triaging process, developers can focus more on coding and less on administrative tasks. This leads to faster resolution times and higher overall productivity.
- **Enhanced Software Quality:**
  - Ensuring that critical bugs are addressed promptly improves the reliability and performance of software products, leading to higher customer satisfaction and reduced maintenance costs.

2.1.2 Quality Assurance (QA) Teams

Profile:

- Teams responsible for testing software and ensuring it meets quality standards before release.

Needs and Benefits:

- **Streamlined Testing Process:**
    - BugWhiz helps QA teams efficiently manage and prioritize bug reports, enabling them to focus on the most critical issues first. This ensures that significant bugs are identified and resolved before the software is released.
- **Consistent Bug Categorization:**
    - Automated categorization ensures that bug reports are consistently classified, which helps in tracking and managing bugs more effectively.
- **Better Communication:**
    - Automated notifications and reminders keep QA teams updated on the status of bug reports, improving coordination with development teams.

2.1.3 Freelance Developers

Profile:

- Independent developers working on multiple projects for various clients.

Needs and Benefits:

- **Cost-Effective Solution:**
    - Freelance developers often have limited resources. BugWhiz provides an affordable solution for managing bug reports efficiently without the need for extensive manual effort.
- **Time Management:**
    - By automating the bug-triaging process, freelance developers can save time and focus more on developing and refining their code.

2.1.4 Open Source Communities

Profile:

- Communities of developers contributing to open-source projects.

Needs and Benefits:

- **Collaborative Bug Management:**
    - Open-source projects often have contributors from different parts of the world. BugWhiz facilitates collaboration by providing a unified platform for managing bug reports.
- **Transparent Tracking:**
    - The system's analytics and reporting tools offer insights into bug patterns and resolution times, helping the community understand and address recurring issues.
- **Enhanced Contributor Engagement:**
    - Automated notifications and reminders ensure that contributors are aware of assigned bugs and deadlines, improving engagement and accountability.

BugWhiz caters to a wide range of customers, each benefiting from its ability to automate and enhance the bug-triaging process. By addressing the specific needs of software development companies, QA teams, freelance developers, and open-source communities, BugWhiz stands out as a versatile and valuable tool in the market. We may offer great discounts for open-source communities.

2.2. Market Survey

In this section, we will analyze the competitive landscape for BugWhiz by examining similar tools and platforms available in the market. Our competition can be divided into two main categories: machine learning and NLP models, and comprehensive bug-tracking systems like Jira and YouTrack. We will explore the strengths and weaknesses of these competitors and highlight how BugWhiz differentiates itself, particularly through its automation capabilities.

2.2.1 Comprehensive Bug Tracking Systems

2.2.1.1 Jira

Jira by Atlassian is one of the most popular bug-tracking and project-management tools used by software development teams.

Pros:

- **Flexibility:**
  - Jira offers a wide range of features for issue tracking, project management, and agile development. Users can customize workflows, dashboards, and fields to fit their needs.
- **Scalability:**
  - Jira can handle large projects with many users and integrates well with other Atlassian products and third-party tools.
- **User-Friendly Interface:**
  - The interface is intuitive and designed to support agile methodologies, making it easy for teams to adapt.

Cons:

- **Complexity:**
  - The extensive feature set can be overwhelming for new users, leading to a steep learning curve.
- **Cost:**
  - Jira's pricing can be high, especially for small teams or startups.
- **Manual Effort:**
  - Despite its features, Jira still requires significant manual effort for categorizing, prioritizing, and assigning bug reports.

2.2.1.2 BugWhiz Differentiation from Jira

- **Automation:**
  - BugWhiz automates the categorization, prioritization, assignment of bugs, and detection of duplicates significantly reducing manual effort and increasing efficiency.
- **Advanced AI Integration:**
  - Our system leverages AI and machine learning models to provide more accurate and consistent bug management.
- **Cost-Effectiveness:**

    - BugWhiz offers a cost-effective solution with advanced features tailored specifically for bug triaging, making it accessible to smaller teams and organizations.

2.2.1.3 YouTrack

YouTrack by JetBrains is another popular issue-tracking and project-management tool known for its powerful search capabilities and agile project management features.

Pros:

- **Customizable Workflow:**
    - YouTrack allows extensive customization of workflows, fields, and issue states, catering to diverse project requirements.
- **Agile Support:**
    - The tool is designed with agile methodologies in mind, providing features like scrum and kanban boards, burndown charts, and sprints.
- **Search Functionality:**
    - YouTrack's advanced search functionality makes it easy to find and filter issues using a query language.

Cons:

- **User Interface:**
    - Some users find the interface less intuitive compared to other tools, which can affect user adoption and efficiency.
- **Integration Limitations:**
    - While YouTrack integrates with several JetBrains products, it has fewer third-party integrations compared to Jira.
- **Manual Processing:**
    - Similar to Jira, YouTrack requires manual effort for categorizing, prioritizing, and assigning bug reports.

2.2.1.4 BugWhiz Differentiation from YouTrack:

- **Fully Automated Workflow:**
    - BugWhiz automates the entire bug triaging process, minimizing the need for manual intervention and speeding up bug resolution times.
- **Machine Learning Models:**
    - Our system uses sophisticated machine learning models to ensure accurate and consistent bug management.
- **User-Friendly Design:**

  - ○ BugWhiz focuses on providing an intuitive and easy-to-navigate interface, enhancing user experience and adoption.

The competitive landscape for bug-tracking systems includes powerful tools like pre-implemented machine-learning models from Python libraries, Jira, and YouTrack. While these tools offer robust features, they fall short in terms of automation and ease of use. BugWhiz addresses these gaps by providing a fully automated, AI-driven bug-triaging system that reduces manual effort, improves accuracy, and enhances productivity. By leveraging custom implementations of machine learning models and offering seamless integration with existing development workflows, BugWhiz stands out as a superior solution for managing bug reports effectively.

2.3. Business Case and Financial Analysis

In this section, we describe the potential success of establishing a company to sell the Automated Bug bug-triaging system, BugWhiz. This analysis is divided into two aspects: the business case and the financial analysis.

2.3.1 Business Case

Based on the market survey above, BugWhiz is well-positioned to address the inefficiencies and manual efforts associated with existing bug-tracking systems like Jira and YouTrack. By offering an automated, AI-driven solution, BugWhiz provides a unique value proposition that can attract a significant customer base.

2.3.1.1 Market Penetration and Sales Forecast

Year 1:

- **Software Development Companies:** 30 companies
- **QA Teams:** 20 teams
- **Freelance Developers:** 50 developers
- **Open Source Communities:** 2 communities

Year 2:

- **Software Development Companies:** Additional 50 companies (total 80)
- **QA Teams:** Additional 40 teams (total 60)
- **Freelance Developers:** Additional 200 developers (total 250)
- **Open Source Communities:** 4 communities (total 6)

Year 3:

- **Software Development Companies:** Additional 100 companies (total 180)
- **QA Teams:** Additional 100 teams (total 160)
- **Freelance Developers:** Additional 400 developers (total 650)
- **Open Source Communities:** 10 communities (total 16)

Year 4:

- **Software Development Companies:** Additional 150 companies (total 330)
- **QA Teams:** Additional 120 teams (total 280)
- **Freelance Developers:** Additional 600 developers (total 1250)
- **Open Source Communities:** Additional 20 communities (total 36)

Year 5:

- **Software Development Companies:** Additional 200 companies (total 530)
- **QA Teams:** Additional 150 teams (total 430)
- **Freelance Developers:** Additional 800 developers (total 2050)
- **Open Source Communities:** Additional 30 communities (total 66)

These projections indicate the anticipated growth in various segments relevant to the BugWhiz project over the next five years, reflecting an expanding market and increasing opportunities for adoption and impact.

2.3.1.2 Pricing Strategy

- **Software Development Companies:** $720 per year
- **QA Teams:** $360 per year
- **Freelance Developers:** $120 per year
- **Open Source Communities:** $12,000 per year

2.3.1.3 Revenue Goals

- **Year 1:** $58,800
- **Year 2:** $181,200
- **Year 3:** $385,200
- **Year 4:** $720,000
- **Year 5:** $1,200,000

2.3.2 Financial Analysis

Based on the business case, we must anticipate both capital expenditure and operational expenditure.

2.3.2.1 Capex

Initial Development Costs (Year 1):

- **Salaries and Wages:**
  - Developers (5): $36,000
  - QA Engineers (2): $14,400
  - Project Manager (1): $12,000
  - Data Scientists/ML Experts (2): $14,400
  - Total Salaries and Wages: **$76,800**
- **Software and Tools:**
  - Development Tools and Licenses: $3,000
  - Cloud Hosting and Infrastructure: $2,000
  - Total Software and Tools: **$5,000**
- **Marketing and Sales:**
  - Digital Marketing: $1,000
  - Sales Team (2): $14,400
  - Total Marketing and Sales: **$15,400**
- **Total Initial Budget: $97,200**

2.3.2.2 Opex

Ongoing Costs (Year 2):

- **Salaries and Wages:**
  - Additional Staff (5): $36,000
  - Total Salaries and Wages: **$36,000**
- **Software and Tools:**
  - Maintenance and Upgrades: $4,000
  - Cloud Hosting and Infrastructure: $5,000
  - Total Software and Tools: **$9,000**
- **Marketing and Sales:**
  - Expanded Digital Marketing: $2,000
  - Sales Team (4): $28,800
  - Total Marketing and Sales: **$30,800**
- **Total Ongoing Budget: $75,800**

Ongoing Costs (Year 3):

- **Salaries and Wages:**
  - Additional Staff (5): $36,000
  - Total Salaries and Wages: **$36,000**
- **Software and Tools:**
  - Maintenance and Upgrades: $4,000
  - Cloud Hosting and Infrastructure: $5,000
  - Total Software and Tools: **$9,000**
- **Marketing and Sales:**
  - Expanded Digital Marketing: $2,000
  - Sales Team (4): $28,800
  - Total Marketing and Sales: **$30,800**
- **Total Ongoing Budget: $75,800**

Ongoing Costs (Year 4):

- **Salaries and Wages:**
  - Additional Staff (5): $36,000
  - Total Salaries and Wages: **$36,000**
- **Software and Tools:**
  - Maintenance and Upgrades: $4,000
  - Cloud Hosting and Infrastructure: $5,000
  - Total Software and Tools: **$9,000**
- **Marketing and Sales:**
  - Expanded Digital Marketing: $2,000
  - Sales Team (4): $28,800
  - Total Marketing and Sales: **$30,800**
  - Total Ongoing Budget (Year 4): **$75,800**

Ongoing Costs (Year 5):

- **Salaries and Wages:**
  - Additional Staff (5): $36,000
  - Total Salaries and Wages: **$36,000**
- **Software and Tools:**
  - Maintenance and Upgrades: $4,000
  - Cloud Hosting and Infrastructure: $5,000
  - Total Software and Tools: **$9,000**
- **Marketing and Sales:**
  - Expanded Digital Marketing: $2,000
  - Sales Team (4): $28,800
  - Total Marketing and Sales: **$30,800**
  - Total Ongoing Budget (Year 5): **$75,800**

These financial projections outline the anticipated capital expenditures and operational expenses for BugWhiz as it progresses through each year, ensuring continued financial sustainability and growth planning.

2.3.2.3 Cash Flow Analysis

Below is a simplified version of the cash flow analysis for the first three years. This analysis includes anticipated revenues and expenses.

Year	Revenue	Capex	Opex	Total Expenses	Profit Before Tax
1	$58,800	$97,200	$0	$97,200	-$38,400
2	$181,200	$0	$75,800	$75,800	$105,400
3	$385,200	$0	$75,800	$75,800	$309,400
4	$720,000	$0	$75,800	$75,800	$644,200
5	$1,200,000	$0	$75,800	$75,800	$1,124,200

Table 1: Five-Year Cash Flow Analysis for BugWhiz

2.3.2.4 Break-Even Analysis

- **Break-Even Point:**
    - The break-even point occurs in the second year when the total revenue surpasses the initial Capex and ongoing Opex.

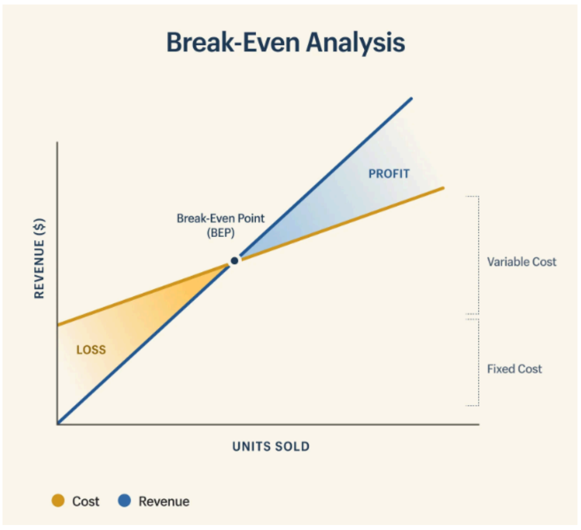


Figure 1: Break-Even Analysis

In the first year, BugWhiz will operate at a loss due to initial development and marketing costs. However, by the second year, the company is expected to achieve profitability as sales increase and ongoing expenses are managed effectively. By the third year, BugWhiz will generate significant profits, establishing itself as a competitive player in the market for automated bug triaging systems.

Chapter 3: Literature Survey

This chapter consists of two parts. In part one, we provide necessary engineering and non-engineering backgrounds crucial for the comprehensive understanding of our project. This includes key facts, theories, formulas, algorithms, and techniques that form the foundation of our work. In part two, we present a literature review of the latest publications related to our project within the past three years, highlighting the current state of research and identifying gaps our project aims to address. The objective is to offer a thorough understanding of the project's context, its underlying technologies, and the advancements in the field.

3.1 Background Knowledge

3.1.1 Natural Language Processing in Bug Triaging

NLP is a critical component in automating the bug triaging process. NLP techniques enable the system to understand, interpret, and manipulate human language, which is essential for processing bug reports.

3.1.1.1 Data Preprocessing

The data preprocessing steps are crucial for preparing bug report texts for feature extraction and machine learning models. Here are the functions and their descriptions:

- **Convert to Lowercase:**
  - Converting text to lowercase ensures uniformity, as 'Bug' and 'bug' would be treated the same.
  - **Example:** "Bug" → "bug"
- **Remove Punctuation:**
  - Punctuation marks are removed to avoid irrelevant tokens.
  - **Example:** "Hello, World!" → "Hello World"
- **Remove Apostrophes:**
  - Apostrophes are removed to standardize contractions and possessives.
  - **Example:** "It's" → "Its"
- **Remove Numbers:**
  - Numbers are removed as they often do not contribute to the meaning in the context of bug descriptions.
  - **Example:** "The error occurred in line 1234" → "The error occurred in line"

- **Remove Stopwords:**
  - Common words that do not contribute to the meaning, like 'the', 'and', 'is', are removed.
  - **Example:** "This is a bug" → "bug"
- **Lemmatization:**
  - Lemmatization reduces words to their base or root form, which helps in treating different forms of a word as a single item.
  - **Example:** "running" → "run", "bugs" → "bug"

The following steps summarize the preprocessing pipeline:

1. **Convert to Lowercase:** Convert the text to lowercase.
2. **Remove Punctuation:** Replace punctuation marks with spaces.
3. **Remove Apostrophes:** Remove apostrophes from the text.
4. **Remove Numbers:** Remove digits from the text.
5. **Remove Stopwords:** Filter out common stopwords.
6. **Lemmatization**: convert words to their base forms

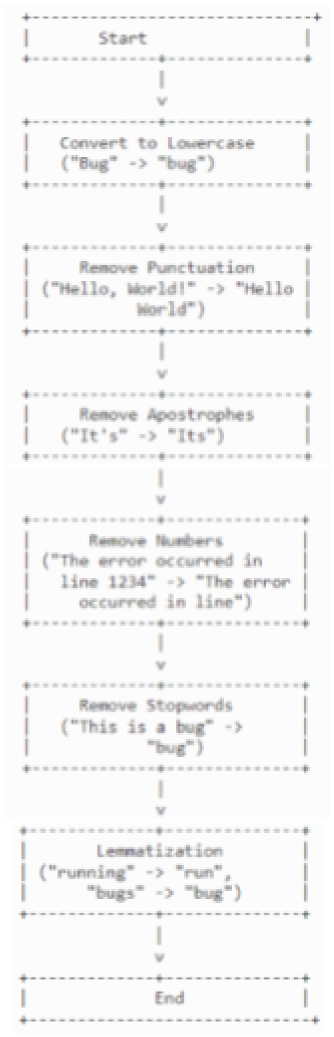

Start
Convert to Lowercase ("Bug" -> "bug")
Remove Punctuation ("Hello, World!" -> "Hello World")
Remove Apostrophes ("It's" -> "Its")
Remove Numbers ("The error occurred in line 1234" -> "The error occurred in line")
Remove Stopwords ("This is a bug" -> "bug")
Lemmatization ("running" -> "run", "bugs" -> "bug")
End

Figure 2: Data Preprocessing Pipeline for Bug Report Texts

3.1.1.2 Feature Extraction

Once the text data is preprocessed, the next step is to extract features that can be used by machine learning models. We use the Term Frequency-Inverse Document Frequency technique for this purpose.

- **Term Frequency**
  - Term frequency measures how frequently a term occurs in a document.
  - The formula is:

$$TF(t,d) = \frac{Total\ number\ of\ terms\ in\ document\ d}{Number\ of\ times\ term\ t\ appears\ in\ document\ d}$$

- **Inverse Document Frequency**
  - Inverse document frequency measures how important a term is in the entire document set.
  - The formula is:

$$IDF(t,D) = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ containing\ term\ t}\right)$$

- **TF-IDF**
  - TF-IDF combines both metrics to weigh terms by their importance, reducing the weight of common terms.
  - The formula is:

$$TF-IDF(t,d,D) = TF(t,d) \times IDF(t,D)$$

- **Example**
  - Consider two documents:
    - Document 1: "bug report issue"
    - Document 2: "bug issue resolved"
    - **Term Frequency (TF):**
      - Term frequency for "bug" in Document 1:

$$TF(bug, Document\ 1) = \frac{Total\ number\ of\ terms\ in\ Document\ 1}{Number\ of\ times\ "bug"\ appears\ in\ Document\ 1} = \frac{1}{3}$$

      - Term frequency for "bug" in Document 2:

$$TF(bug, Document\ 2) = \frac{Total\ number\ of\ terms\ in\ Document\ 2}{Number\ of\ times\ "bug"\ appears\ in\ Document\ 2} = \frac{1}{3}$$

- **Inverse Document Frequency (IDF):**

$$IDF(bug) = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ containing\ "bug"}\right) = log\left(\frac{2}{2}\right) = 0$$

  - If "bug" appears in 2 out of 2 documents, IDF for "bug":

- **TF-IDF:**
  - Thus, TF-IDF for "bug" in either document would be:

$$TF-IDF(bug, Document\ 1) = TF(bug, Document\ 1) \times IDF(bug) = \frac{1}{3} \times 0 = 0$$

$$TF-IDF(bug, Document\ 2) = TF(bug, Document\ 2) \times IDF(bug) = \frac{1}{3} \times 0 = 0$$

  - Since the term "bug" appears in all documents, its IDF is zero, making its TF-IDF zero for both documents.
- **TF for Other Terms:**
  - For "report" in Document 1:

$$TF(report, Document\ 1) = \frac{1}{3}$$

$$TF(issue, Document\ 1) = \frac{1}{3}$$

  - For "issue" in Document 1:
  - For "issue" in Document 2:

$$TF(issue, Document\ 2) = \frac{1}{3}$$

  - For "resolved" in Document 2:

$$TF(resolved, Document\ 2) = \frac{1}{3}$$

- **IDF for Other Terms:**
  - For "report":

$$IDF(report) = log\left(\frac{2}{1}\right) = log(2)$$

- For "issue":

$$IDF(issue) = log\left(\frac{2}{2}\right) = 0$$

- For "resolved":

$$IDF(resolved) = log\left(\frac{2}{1}\right) = log(2)$$

- **TF-IDF for Other Terms:**
  - TF-IDF for "report" in Document 1:

$$TF - IDF(report, Document\ 1) = TF(report, Document\ 1) \times IDF(report) = \frac{1}{3} \times log(2)$$

  - TF-IDF for "issue" in Document 1:

$$TF - IDF(issue, Document\ 1) = TF(issue, Document\ 1) \times IDF(issue) = \frac{1}{3} \times 0 = 0$$

  - TF-IDF for "issue" in Document 2:

$$TF - IDF(issue, Document\ 2) = TF(issue, Document\ 2) \times IDF(issue) = \frac{1}{3} \times 0 = 0$$

  - TF-IDF for "resolved" in Document 2:

$$TF - IDF(resolved, Document\ 2) = TF(resolved, Document\ 2) \times IDF(resolved) = \frac{1}{3} \times log(2)$$

- **Summary:**
  - For term "bug":
    - TF-IDF in Document 1: 0
    - TF-IDF in Document 2: 0
  - For term "report" in Document 1:
    - TF-IDF: ⅓ x log(2)
  - For term "issue":
    - TF-IDF in Document 1: 0
    - TF-IDF in Document 2: 0
  - For term "resolved" in Document 2:
    - TF-IDF: ⅓ x log(2)

This example demonstrates how TF-IDF is calculated, showing that terms common across all documents have lower significance compared to more unique terms.

3.1.1.3 Cosine Similarity for Duplicate Detection

Cosine similarity measures the cosine of the angle between two non-zero vectors of an inner product space. It is used to determine how similar two documents are, regardless of their size.

- **Cosine Similarity Formula:**

$$cosine\_similarity = cos(\theta) = \frac{A.B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

- **Example:**
    - If A=[1,2,3] and B=[4,5,6]

$$\begin{aligned} A.B &= 1\times4+2\times5+3\times6 = 32 \\ ||A|| &= \sqrt{1^2+2^2+3^2} = \sqrt{14} \\ ||B|| &= \sqrt{4^2+5^2+6^2} = \sqrt{77} \\ cosine\_similarity &= \frac{32}{\sqrt{14}\sqrt{77}} = 0.974 \end{aligned}$$

Alternative Similarity Measures:

- **Euclidean Distance:**
    - Measures the straight-line distance between two points in Euclidean space. Less effective for text data as it does not account for vector direction.
- **Jaccard Similarity:**
    - Measures the similarity between finite sample sets. Defined as the size of the intersection divided by the size of the union of the sample sets. Suitable for binary data but less effective for numerical feature vectors like TF-IDF.

Cosine similarity is chosen because it is scale-invariant, meaning it focuses on the orientation rather than magnitude, making it particularly effective for high-dimensional, sparse datasets like text.

In summary, the preprocessing steps standardize the bug report data, TF-IDF converts text to numerical features, and cosine similarity efficiently identifies duplicates by measuring the angle between feature vectors, ensuring robust and scalable duplicate detection in our Automated Bug Triaging System.

These NLP techniques are used to preprocess bug report data, extract meaningful features, and convert textual information into numerical vectors suitable for machine learning models.

3.1.1.4 Singular Value Decomposition

Singular Value Decomposition (SVD) is a fundamental matrix factorization method in linear algebra. It decomposes a matrix into three simpler matrices, enabling dimensionality reduction and capturing important patterns in data.

- **Mathematical Formulation:**
    - For a given matrix X of dimensions $m \times n$, SVD decomposes X into the product of three matrices: $X=U\Sigma V^T$
        - U is an $m \times n$ orthogonal matrix (left singular vectors),
        - Σ an $m \times n$ diagonal matrix with non-negative real numbers on the diagonal (singular values),
        - V^T is an $n \times n$ orthogonal matrix (right singular vectors).
- **Key Concepts**
    1. Orthogonality: The matrices U and V are orthogonal, meaning $UU^T = I$ and $VV^T = I$, where I is the identity matrix. This property ensures that the transformations preserve the length of vectors and do not distort angles.
    2. Diagonal Matrix Σ: The diagonal elements of Σ (singular values) represent the importance of corresponding singular vectors in capturing variance within the data. These values are arranged in descending order, with the highest value representing the most significant pattern in the data.
- **Applications of SVD**
    1. Dimensionality Reduction: SVD is commonly used in machine learning and data analysis to reduce the dimensionality of data while preserving important information. By retaining only the most significant singular values and their corresponding singular vectors, SVD can reduce noise and focus on the essential patterns within the data.
    2. Feature Extraction: In Natural Language Processing (NLP), SVD applied after techniques like TF-IDF (Term Frequency-Inverse Document Frequency) can extract meaningful features from text data. It transforms the sparse matrix representation of text features into a denser representation, where each document (or data point) is represented by a reduced set of features that capture the most critical aspects of the data.

- **How SVD Works**
    1. Compute the Singular Values: SVD computes the singular values and sorts them in descending order. These values quantify the importance

of each singular vector in explaining the variability in the original data matrix.

2. Construct Reduced Matrices: To reduce dimensionality, truncate U, Σ, and V matrices by keeping only the top k singular values and their corresponding singular vectors. This results in reduced matrices U_k, Σ_k, and V_k^T, where k is typically determined based on the desired level of dimensionality reduction or variance retention.
3. Reconstruct Data: The reduced matrices U_k, Σ_k, and V_k^T are multiplied to reconstruct an approximation X_k of the original matrix X. This approximation retains the most significant patterns in the data while reducing noise and irrelevant information.

In summary, SVD is a powerful mathematical technique for decomposing and analyzing matrices. Its applications range from dimensionality reduction and feature extraction in machine learning to pattern discovery in data analysis. In NLP and other fields, SVD plays a crucial role in transforming and simplifying data representations, enabling more efficient and effective analysis and modeling.

3.1.2 Machine Learning Algorithms for Bug Triaging

Machine learning algorithms are employed to categorize, prioritize, and assign bug reports to appropriate developers. The following are key ML algorithms utilized in our project:

3.1.2.1 Support Vector Machine

- Overview
    - SVM is a supervised learning model used for classification and regression analysis.
    - The objective in SVM is to find the hyperplane that achieves the maximum margin, which is the greatest distance from the nearest points in the two classes.
    - This hyperplane (A) in figure 3 is shown to have better generalization capabilities and is more resilient to noise compared to a hyperplane that does not maximize the margin (B).

- Application
    - In our system, SVM is utilized for categorizing bug reports into predefined categories (e.g., UI, performance, security), as well as predicting the top 5 developers likely to resolve new bug reports.

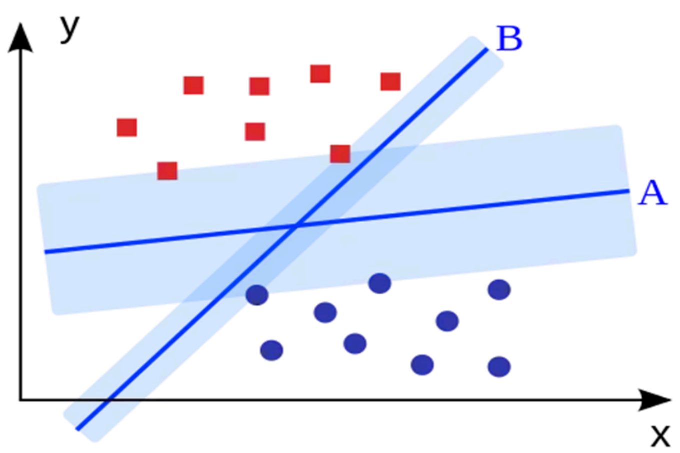


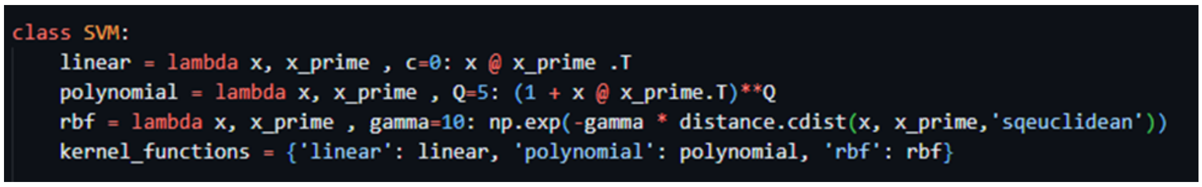
Figure 3: A have better generalization capabilities compared to B

To achieve this, SVM finds the hyperplane's *W* and b by solving the following optimization problem:

$$Max_{w,b} Min_n \left(w^T x_n / ||w||\right) \text{ such that } y_n\left(w^T x_n + b\right) > 0 \text{ for } 1 \leqslant n \leqslant N$$

SVM Algorithm

Kernels and SVM Hyperparameters



```
class SVM:
    linear = lambda x, x_prime , c=0: x @ x_prime .T
    polynomial = lambda x, x_prime , Q=5: (1 + x @ x_prime.T)**Q
    rbf = lambda x, x_prime , gamma=10: np.exp(-gamma * distance.cdist(x, x_prime,'sqeuclidean'))
    kernel_functions = {'linear': linear, 'polynomial': polynomial, 'rbf': rbf}
```

Figure 4: SVM Hyperparameters

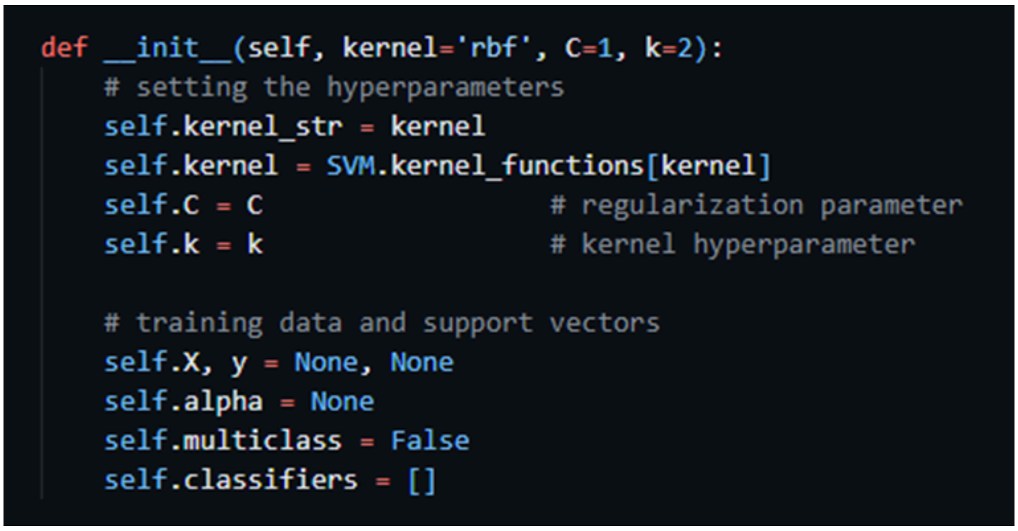
We start by defining the three kernels using their respective functions:

$$K(X, X') = X^T X', \textit{ for Linear kernal}$$
$$K(X, X') = (1 + X^T X')^Q, \textit{ for Polynomial kernal}$$
$$K(X, X') = e^{-\gamma ||X - X'||}, \textit{ for Radial Basis Function}$$

Figure 5: SVM Kernels

We use *distance* from *scipy.spatial* library to compute the Gaussian kernel.

Constructor



```python
def __init__(self, kernel='rbf', C=1, k=2):
    # setting the hyperparameters
    self.kernel_str = kernel
    self.kernel = SVM.kernel_functions[kernel]
    self.C = C                     # regularization parameter
    self.k = k                     # kernel hyperparameter

    # training data and support vectors
    self.X, y = None, None
    self.alpha = None
    self.multiclass = False
    self.classifiers = []
```

Figure 6: SVM Constructor

The SVM has three main hyperparameters, the kernel (the user selects it), the regularization parameter C and the kernel hyperparameter (to be passed to the kernel function), it represents Q for the polynomial kernel and γ for the RBF kernel.

Fit Method

Fitting the SVM corresponds to finding the support vector α for each point by solving the dual optimization problem:

$$Max_{\alpha} \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\left(\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m K(x_n, x_m)\right) \text{ such that } 0 \leqslant \alpha n \leqslant C \text{ and } \sum_{n=1}^{N} \alpha_n y_n = 0$$

Let α be a variable column vector $(\alpha_1\ \alpha_2\ \dots\ \alpha\_N)^t$ and let y be a constant column vector for the labels $(y_1\ y_2\ \dots\ y\_N)^t$ and let K be a constant matrix where K[n,m] computes the kernel at (x▯, x▯). we have following index-based equivalences for the dot product, outer product and quadratic form respectively:

$$\mathbf{u}\cdot\mathbf{v} = \sum_{n=1}^{N} u_n \cdot v_n$$

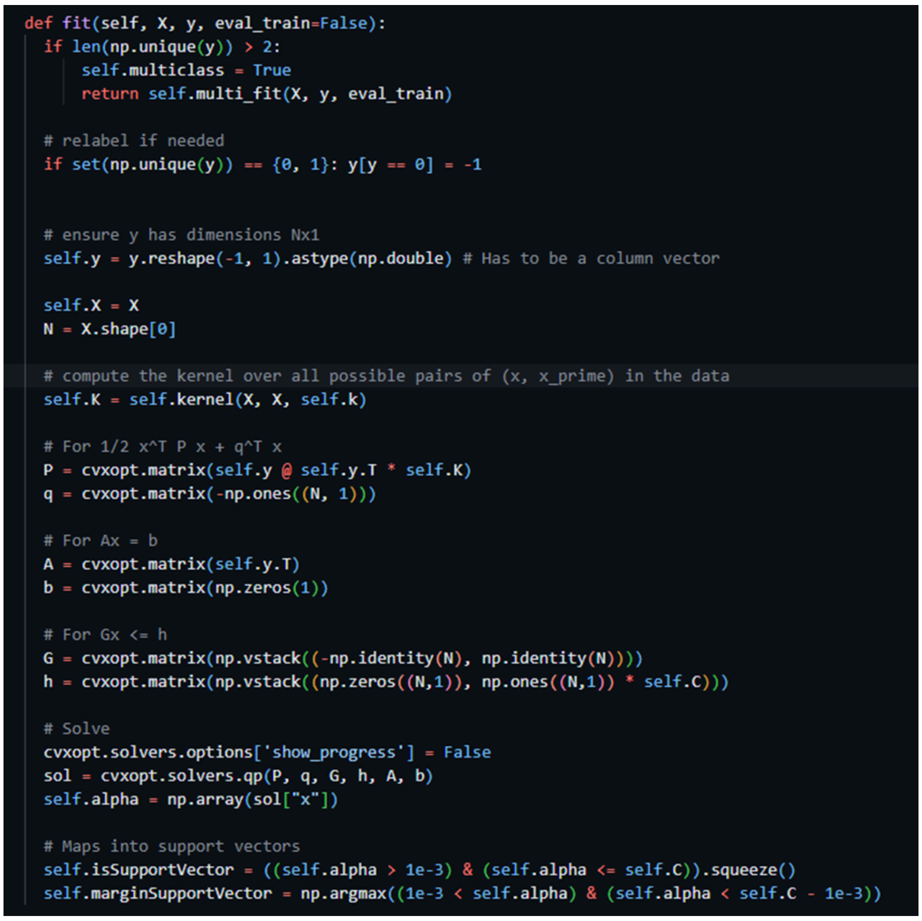
$$(uv^T)_{nm} = u_n v_m$$

$$Q(\mathbf{z}) = \mathbf{u}^T\mathbf{A}\mathbf{u} = \sum_{n=1}^{N}\sum_{m=1}^{N} u_n u_m A_{nm}$$

to be able to write the dual optimization problem in matrix form as follow:

$$\max_{\alpha}[1^T\alpha - \frac{1}{2} * \alpha^T(yy^T * K)\alpha] \quad \text{subject to} \quad y^T\alpha = 0,\ 0 \leq \alpha_i \leq C,\ \forall i = 1, 2, \ldots, n$$

Knowing that this is a quadratic program, we will use cvxopt library to solve it.



```python
def fit(self, X, y, eval_train=False):
    if len(np.unique(y)) > 2:
        self.multiclass = True
        return self.multi_fit(X, y, eval_train)

    # relabel if needed
    if set(np.unique(y)) == {0, 1}: y[y == 0] = -1


    # ensure y has dimensions Nx1
    self.y = y.reshape(-1, 1).astype(np.double) # Has to be a column vector

    self.X = X
    N = X.shape[0]

    # compute the kernel over all possible pairs of (x, x_prime) in the data
    self.K = self.kernel(X, X, self.k)

    # For 1/2 x^T P x + q^T x
    P = cvxopt.matrix(self.y @ self.y.T * self.K)
    q = cvxopt.matrix(-np.ones((N, 1)))

    # For Ax = b
    A = cvxopt.matrix(self.y.T)
    b = cvxopt.matrix(np.zeros(1))

    # For Gx <= h
    G = cvxopt.matrix(np.vstack((-np.identity(N), np.identity(N))))
    h = cvxopt.matrix(np.vstack((np.zeros((N,1)), np.ones((N,1)) * self.C)))

    # Solve
    cvxopt.solvers.options['show_progress'] = False
    sol = cvxopt.solvers.qp(P, q, G, h, A, b)
    self.alpha = np.array(sol["x"])

    # Maps into support vectors
    self.isSupportVector = ((self.alpha > 1e-3) & (self.alpha <= self.C)).squeeze()
    self.marginSupportVector = np.argmax((1e-3 < self.alpha) & (self.alpha < self.C - 1e-3))
```

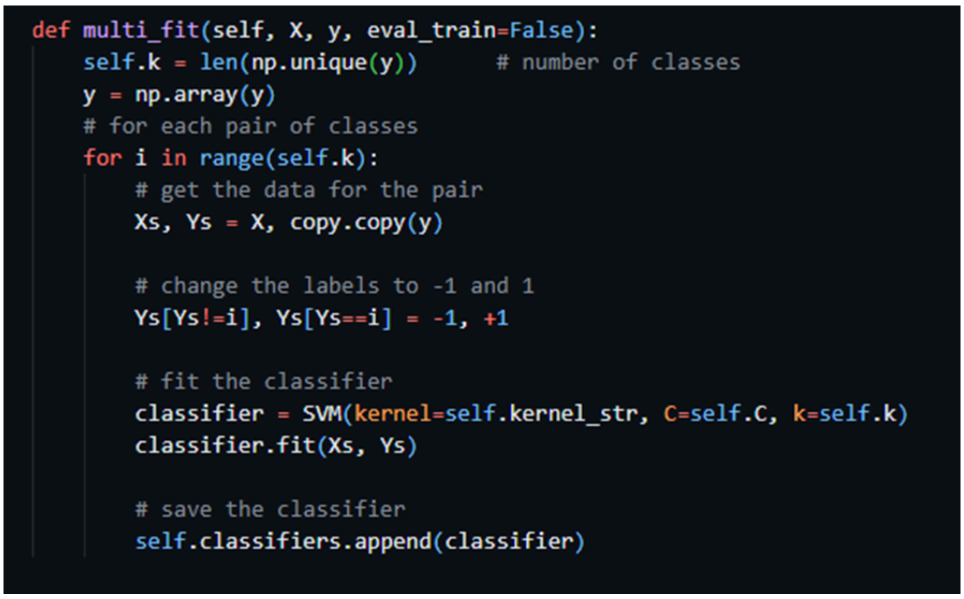
Figure 7: SVM Fit Method

We ensure that this is a binary problem and that the binary labels are set as assumed by SVM (+1, -1) and that y is a column vector with dimensions (N,1). Then we solve the optimization problem to find $(\alpha_1\ \alpha_2\ \ldots\ \alpha\_N)^t$.

We use $(\alpha_1\ \alpha_2\ \ldots\ \alpha\_N)^t$ to get an array of flags that is 1 at any index corresponding to a support vector so that we can later apply the prediction equation by only summing over support vectors and an index for a margin support vector for (x▯,y▯). Notice that in the checks we do assume that non-support vectors may not have α=0 exactly,

if it's $\alpha \leq 10^{-3}$, then this is approximately zero (we know CVXOPT results may not be ultimately precise). Likewise, we assume that non-margin support vectors may not have α=C exactly.

Multiclass Fit



```python
def multi_fit(self, X, y, eval_train=False):
    self.k = len(np.unique(y))      # number of classes
    y = np.array(y)
    # for each pair of classes
    for i in range(self.k):
        # get the data for the pair
        Xs, Ys = X, copy.copy(y)

        # change the labels to -1 and 1
        Ys[Ys!=i], Ys[Ys==i] = -1, +1

        # fit the classifier
        classifier = SVM(kernel=self.kernel_str, C=self.C, k=self.k)
        classifier.fit(Xs, Ys)

        # save the classifier
        self.classifiers.append(classifier)
```

Figure 8: SVM Multiclass Fit

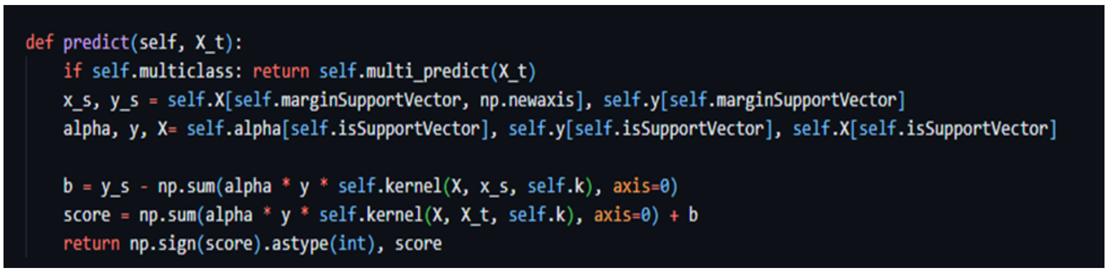
To generalize the model to multiclass, over k classes. We train a binary SVM classifier for each class present where we loop on each class and relabel points belonging to it into +1 and points from all other classes into -1.

The result from training is k classifiers when given k classes where the i^{th} classifier was trained on the data with the i^{th} class being labeled as +1 and all others being labeled as -1.

Predict Method

The prediction equation is:

$$g(x) = \sum_{\alpha_i>0} \alpha_i y_i K(x_i, x) + (y_s - \sum_{\alpha_i>0} \alpha_i y_i K(x_i, x_s))$$



```python
def predict(self, X_t):
    if self.multiclass: return self.multi_predict(X_t)
    x_s, y_s = self.X[self.marginSupportVector, np.newaxis], self.y[self.marginSupportVector]
    alpha, y, X= self.alpha[self.isSupportVector], self.y[self.isSupportVector], self.X[self.isSupportVector]

    b = y_s - np.sum(alpha * y * self.kernel(X, x_s, self.k), axis=0)
    score = np.sum(alpha * y * self.kernel(X, X_t, self.k), axis=0) + b
    return np.sign(score).astype(int), score
```

Figure 9: Predict Method

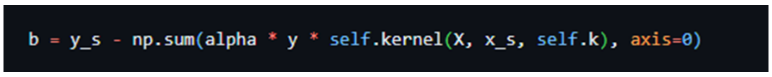
1- Multiclass Handling:

- If the model is trained in a multiclass setting ,it calls the `multi_predict` method to handle multiclass predictions.

2- Extract Support Vectors:

- `x_s` and `y_s` are extracted using the `marginSupportVector`. `x_s` is the feature vector of the margin support vector (the vector that lies exactly on the margin boundary), and `y_s` is its corresponding label.
- `alpha`, `y`, and `X` are the coefficients, labels, and feature vectors of all support vectors, respectively. These are identified during the training phase and used in the prediction phase.

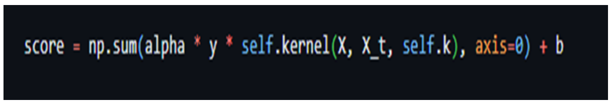
3- Calculate Intercept:



```
b = y_s - np.sum(alpha * y * self.kernel(X, x_s, self.k), axis=0)
```

- The intercept `b` is calculated using the margin support vector, it ensures that the hyperplane is correctly positioned. This is done by subtracting the weighted sum of the kernel evaluations between the support vectors and the margin support vector from the label of the margin support vector.

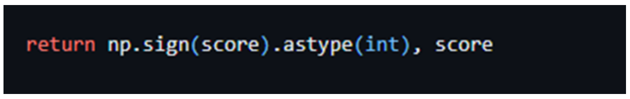
4- Compute Decision Function:



```
score = np.sum(alpha * y * self.kernel(X, X_t, self.k), axis=0) + b
```

- The decision function (or score) for each test point is computed. This involves summing the product of `alpha`, `y`, and the kernel evaluations between the support vectors and the test points `X_t`, and adding the intercept `b`.

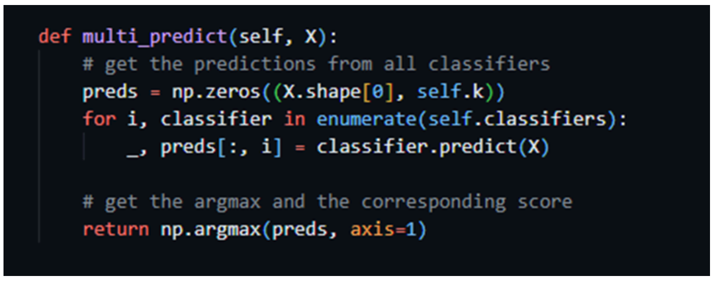
5- Predict Class Labels:



```
return np.sign(score).astype(int), score
```

- The class labels are predicted by taking the sign of the decision function (score). If the score is positive, the point is classified as +1; if negative, it is classified as -1.

Multiclass Prediction



```
def multi_predict(self, X):
    # get the predictions from all classifiers
    preds = np.zeros((X.shape[0], self.k))
    for i, classifier in enumerate(self.classifiers):
        _, preds[:, i] = classifier.predict(X)

    # get the argmax and the corresponding score
    return np.argmax(preds, axis=1)
```

1- Initialize Prediction Array:

- An array `preds` of shape `(number_of_test_points, number_of_classes)` is initialized to store the prediction scores from each binary classifier.

2- Binary Classifiers Predictions:

- For each classifier (each trained to distinguish one class from the rest), predictions are made for the test points `X`, and the scores are stored in the corresponding column of `preds`.

3- Determine Final Class Labels:

- The final class label for each test point is determined by taking the `argmax` of the `preds` array along the axis corresponding to the classes. This gives the index of the class with the highest score for each test point.

3.1.2.2 Logistic Regression

It is a probabilistic classifier that outputs probabilities that decide the classification target. It is better than non-probabilistic in that the model clarifies how confident it is regarding the final classification.

Negative log-likelihood loss for a single example:

$$e_{in}(w) = log\left(1 + e^{-y_i w^T x_i}\right)$$

The average loss for all the training examples is given by:

$$E_{in}(w) = \frac{1}{M} \sum_{i=1}^{i=M} log\left(1 + e^{-y_i w^T x_i}\right)$$

To minimize this loss:

$$\nabla_w E_{in} = 0$$

$$\nabla_w E_{in} = \frac{1}{M} \sum_{i=1}^{i=M} \left(\frac{-y_i x_i}{1+e^{y_i w^T x_i}}\right) = 0$$

Solving the equation to obtain the optimal w is infeasible. Hence, we won't be able to arrive at a closed form solution.

So, we can apply the iterative gradient descent algorithm as follows:

$$w_{t+1} = w_t - \eta \nabla_w E_{in} = w_t - \eta * \frac{1}{M} \sum_{i=1}^{i=M} \left(\frac{-y_i x_i}{1+e^{y_i w^T x_i}} \right)$$

To predict a new sample x:

$$p(y =+ 1|x) = \theta(w^T x)$$
$$p(x) = \theta\left(- w^T x\right) = 1 - \theta\left(w^T x\right)$$

Where $\theta(z)$ is the sigmoid function:

$$\theta(z) = \frac{1}{1+ e^{-z}}$$

Then a threshold for this probability is set to determine the class label for the new sample.

However, the computation of the gradients ($\nabla_w E_{in}$) can be very expensive with increasing the training data size.

A synchronous data parallelism technique is used to address this problem which is summarized as follows:

- There is a master node which holds the weights of the current iteration.
- Divide the data into **N** number of partitions, where **N** is the total number of available workers in the computer cluster.
- Each worker node receives a copy of weights at the start of each iteration.

Each worker performs the following computation using its data subset:

$$\nabla_w E'_{in}(w) = \sum_{i=1}^{i=M'} \left(\frac{-y_i x_i}{1+e^{y_i w^T x_i}} \right)$$

Where M' is the size of the worker's data subset.

The worker sends those partial gradients along with the size of the data subset to the master node which perform the following computation using the results from all the workers in the cluster:

$$M = \sum_{i=1}^{i=N} M'_i$$

$$\nabla_w E_{in} = \frac{1}{M}\sum_{i=1}^{i=N}\left(\nabla_w E'_{in}(w)\right)_i$$

Then, it updates the weights using the computed gradients and sends the new weights to all the workers to begin another iteration. This process is repeated for a pre-defined number of iterations.

2.1.2.2 DeBERTa Transformer:

DeBERTa (Decoding-enhanced BERT with Disentangled Attention) is a transformer model developed by Microsoft for natural language processing (NLP) tasks. It is a variant of the BERT (Bidirectional Encoder Representations from Transformers) model with several improvements aimed at enhancing performance. Here’s an overview of what DeBERTa is and how it works:

What is DeBERTa?

DeBERTa is an advanced transformer model designed to improve upon the original BERT architecture by addressing some of its limitations. It introduces several key innovations:

- **Disentangled Attention Mechanism**: Unlike BERT, which uses a single attention matrix for both content and position embeddings, DeBERTa uses disentangled attention. This means it separates the attention mechanisms for content and position, which allows the model to better capture the relationship between different words in a sentence.
- **Enhanced Mask Decoder**: DeBERTa employs an enhanced mask decoder to improve the model’s ability to predict masked tokens during training. This enhances the model's pretraining process and leads to better performance on downstream tasks.

How Does DeBERTa Work?

DeBERTa works by leveraging the transformer architecture, which includes an encoder made up of multiple layers of self-attention mechanisms and feed-forward neural networks. Here’s a breakdown of its working:

1. **Tokenization and Embeddings**:
    - Input text is tokenized into subwords or tokens.
    - Each token is converted into two types of embeddings: content embeddings (which represent the token itself) and position embeddings (which represent the token's position in the sequence).
2. **Disentangled Attention**:
    - The disentangled attention mechanism separates the processing of content and position embeddings. This means the model creates two separate attention matrices: one for content and one for position.

    - During attention computation, these two matrices are combined to form a more nuanced understanding of the token relationships.
3. **Transformer Layers**:
    - The token representations pass through multiple transformer layers. Each layer consists of multi-head self-attention and feed-forward neural networks.
    - The self-attention mechanism helps the model focus on relevant parts of the input sequence by weighting the importance of different tokens.
4. **Output Representations**:
    - After passing through the transformer layers, the final token representations can be used for various NLP tasks such as classification, named entity recognition, or machine translation.
5. **Pretraining and Fine-tuning**:
    - DeBERTa is pretrained on a large corpus of text using a masked language modeling objective, where certain tokens in the input are masked, and the model learns to predict them.
    - The pretrained model is then fine-tuned on specific downstream tasks with task-specific labeled data to adapt the general language understanding to the particular task.

Advantages of DeBERTa:

- **Improved Context Understanding**: The disentangled attention mechanism helps DeBERTa better capture the relationships between tokens, improving context understanding.
- **Enhanced Performance**: DeBERTa has shown superior performance on various NLP benchmarks compared to traditional BERT and other transformer models.
- **Flexibility**: It can be applied to a wide range of NLP tasks, making it a versatile tool in the field.

2.1.2.3 Adam optimizer:

The Adam optimizer is an advanced optimization algorithm designed to train deep learning models more efficiently. It combines the best features of two other popular optimization algorithms: AdaGrad and RMSProp. Here's a detailed explanation of what the Adam optimizer is and how it works:

What is Adam Optimizer?

Adam (short for Adaptive Moment Estimation) is an optimization algorithm used in training machine learning and deep learning models. It is particularly well-suited for handling sparse gradients on noisy problems. Adam is designed to maintain an adaptive learning rate for each parameter by using estimates of the first and second moments (mean and uncentered variance) of the gradients.

How Does Adam Optimizer Work?

Adam works by computing adaptive learning rates for each parameter. Here’s a step-by-step breakdown of the algorithm:

Initialization:

- Initialize the parameters θ\thetaθ (weights) of the model.
- Initialize the first moment vector m=0m = 0m=0 and the second moment vector v=0v = 0v=0.
- Set the timestep t=0t = 0t=0.
- Set hyperparameters:
  - α (learning rate, typically set to 0.001).
  - β1 (decay rate for the first moment estimates, typically set to 0.9).
  - β2 (decay rate for the second moment estimates, typically set to 0.999).
  - ϵ (small constant to prevent division by zero, typically set to 10−810^{-8}10−8).

Parameter Update Loop (for each training step):

- Increment the timestep t.
- Compute the gradient g_t of the objective function with respect to the parameters θ at timestep t.
  - Update the biased first moment estimate:

    $$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

  - Update the biased second moment estimate:

    $$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

  - Compute bias-corrected first moment estimate:

    $$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

  - Compute bias-corrected second moment estimate:

    $$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

  - Update the parameters:
-
  $$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Repeat:

- Repeat the update loop until convergence or for a predefined number of iterations.

Advantages of Adam Optimizer

1. **Adaptive Learning Rate**: The learning rate adapts based on the mean and variance of the gradients, which can lead to faster convergence.
2. **Bias Correction**: Adam includes bias correction to address the issue of the moment estimates being biased towards zero, especially in the initial steps.
3. **Efficient**: Adam is computationally efficient and requires little memory, making it suitable for large datasets and high-dimensional parameter spaces.
4. **Robust to Noisy Data**: It works well with sparse gradients and noisy data, making it versatile for various types of machine learning problems.

3.2 Recent Literature Review

3.2.1 Advances in NLP for Bug Triaging

Recent studies have explored the application of machine learning techniques to improve the accuracy and efficiency of bug triaging systems. Support Vector Machines and Term Frequency-Inverse Document Frequency have been particularly effective in enhancing bug report categorization and duplicate detection, respectively. SVM models are well-suited for classification tasks, making them ideal for categorizing bug reports into predefined categories. On the other hand, TF-IDF is a robust feature extraction method that quantifies the importance of terms within a document, which is crucial for identifying duplicate bug reports.

Relevant papers:

- Approaches for automated bug triaging: A review
- Automated Bug Triage Using Machine Learning Algorithms
- Enhancing Bug Triage with NLP Techniques

3.2.2 Machine Learning Models for Bug Prioritization

Studies have shown that machine learning algorithms can effectively prioritize bug reports based on severity and impact. Recent research has focused on improving the scalability and adaptability of these models to handle large volumes of bug reports efficiently. Logistic regression, in particular, has been widely used due to its simplicity and effectiveness in binary and multi-class classification tasks.

3.2.3 Integration of AI in Software Development Tools

The integration of AI-driven tools in software development has been a growing trend. Research has emphasized the benefits of automation in reducing manual effort and enhancing overall productivity. AI tools help in various stages of software development, from code review to bug triaging, thereby improving efficiency and accuracy.

3.3 Comparative Study of Previous Work

In this section, we compare and contrast recent works in the field of bug triaging, focusing on their methodologies, effectiveness, and limitations. This comparative analysis helps in understanding the landscape of current research and identifying areas where our project can contribute significant improvements.

3.3.1 Comparative Analysis:

- **Efficiency:**
  - Comparing different models' efficiency in processing and triaging bug reports.
- **Accuracy:**
  - Evaluating the accuracy of various models in categorizing and prioritizing bug reports.
- **Scalability:**
  - Assessing the scalability of different systems in handling large volumes of bug reports.
- **Integration:**
  - Examining how well each system integrates with existing development tools and workflows.

3.4 Implemented Approach

Based on our literature review and comparative study, we have chosen an approach that leverages state-of-the-art NLP and machine learning models to automate the bug triaging process.

3.4.1 Chosen Approach

- **NLP Models:**

    - We use TF-IDF for feature extraction
    - We use SVM for developer assignments.
- **Machine Learning Algorithms:**
    - SVM for bug categorization
    - logistic regression for bug prioritization.

3.4.2 Justification

The selected approach provides a balance between accuracy, efficiency, and scalability. By using advanced models like SVM and leveraging the power of machine learning, our system is designed to handle large volumes of bug reports with high precision and minimal manual intervention. This ensures timely resolution of critical issues, improves software quality, and enhances user satisfaction.

3.4.3 Conclusion

In conclusion, this chapter has provided a detailed review of the foundational knowledge and recent advancements related to the Automated Bug Triaging System. The insights gained from this literature survey have informed the design and implementation of our project, ensuring it is built on a solid foundation of proven methodologies and cutting-edge research.

Chapter 4: System Design and Architecture

This chapter represents the main body of our Automated Bug Triaging System project. It provides a detailed description of the system design and architecture, answering the questions "what has been done?" and "how it has been done?". We will highlight and discuss the steps taken to realize the project, clarifying our scientific approaches and methodologies. The discussion will follow a logical flow, starting from the whole block diagram, to coarse modules, and finally to fine modules. Detailed descriptions are provided to ensure that an interested reader could easily replicate and improve our work.

4.1. Overview and Assumptions

In designing our Automated Bug Triaging System, we aimed to automate the process of handling bug reports, including categorization, prioritization, developer assignment, and duplicate detection. Our system leverages machine learning and NLP techniques to achieve high accuracy and efficiency. The key assumptions we employed are:

- **Data Quality**:
    - We assumed the availability of high-quality bug report data, with sufficient historical data for training machine learning models.
- **Consistent Terminology**:
    - We assumed that bug reports follow a consistent terminology, which is crucial for accurate text processing and feature extraction.
- **Scalability Requirements**:
    - The system is designed to handle a large volume of bug reports efficiently, assuming that the underlying infrastructure supports scalable solutions.

4.2. System Architecture

The architecture of our system is designed to ensure modularity, scalability, and efficiency. The system is divided into several modules, each responsible for a specific function. The overall architecture is represented as a block diagram, followed by detailed explanations of each module.

4.2.1. Block Diagram

The block diagram below illustrates the high-level architecture of our Automated Bug Triaging System. It highlights the key modules and their interactions.

The interaction between these modules is crucial for the system's overall functionality. Each module performs specific tasks and passes the processed data to the next module.

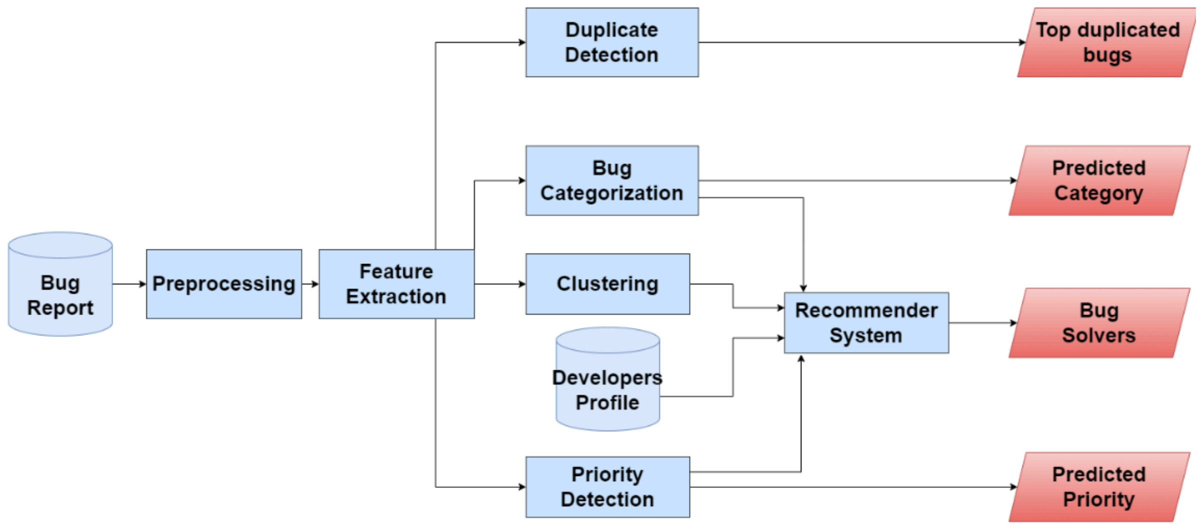


Figure 10: high-level architecture of our Automated Bug Triaging System

4.3 Module 1: Preprocessing and Feature Extraction

Preprocessing and feature extraction are critical steps in our system, where raw bug report data is cleaned, standardized, and transformed into numerical features suitable for machine learning models.

4.3.1. Functional Description

The preprocessing and feature extraction module converts raw text data into a standardized format and numerical vectors. This process includes:

- Converting text to lowercase
- Removing punctuation
- Removing numbers
- Removing stopwords
- Performing lemmatization
- Extracting features

4.3.2. Modular Decomposition

The preprocessing and feature extraction module consists of the following submodules:

- **Convert to Lowercase:**
    - **Function:**
        - Ensures uniformity by converting all text to lowercase.
    - **Example:**
        - "Bug" → "bug"
- **Remove Punctuation:**
    - **Function:**
        - Eliminates punctuation marks to avoid irrelevant tokens.
    - **Example:**
        - "Hello, World!" → "Hello World"
- **Remove Numbers:**
    - **Function:**
        - Strips out numerical data that does not contribute to the bug report's meaning.
    - **Example:**
        - "The error occurred in line 1234" → "The error occurred in line"
- **Remove Stopwords:**
    - **Function:**
        - Filters out common stopwords that do not add significant meaning.
    - **Example:**
        - "This is a bug" → "bug"
- **Lemmatization:**
    - **Function:**
        - Reduces words to their base forms for consistency.
    - **Example:**
        - "running" → "run", "bugs" → "bug"

4.3.3. Design Constraints

The preprocessing and feature extraction module must handle large volumes of text efficiently. It should be designed to process data in parallel to meet performance requirements. Ensuring data integrity and consistency during preprocessing is critical.

4.3.4. Other Description of Module 1

This module plays a vital role in preparing data for machine learning models. Poor preprocessing can lead to inaccurate feature extraction and model training, affecting the overall system performance.

4.4 Module 2: Duplicate Detection

Duplicate detection involves identifying similar bug reports to prevent redundant work and ensure efficient use of developer resources.

4.4.1. Functional Description

The duplicate detection module uses cosine similarity to compare feature vectors and retrieve the top k duplicates of a given bug report. This helps in identifying and merging duplicate reports, thereby optimizing the bug resolution process.

4.4.2. Modular Decomposition

The duplicate detection module consists of the following submodules:

- **TF-IDF Vectorization:**
    - **Function:**
        - Converts text data into TF-IDF vectors, which represent the importance of terms in the documents.
- **Similarity Calculation:**
    - **Function:**
        - Computes the cosine similarity between the feature vector of the new bug report and those in the existing database.
- **Top k Retrieval:**
    - **Function:**
        - Retrieves the top k most similar bug reports based on the computed similarity scores.

4.4.3. Design Constraints

The duplicate detection process must be efficient and scalable, capable of handling large datasets. The similarity computation should be optimized to avoid performance bottlenecks, and the system must be able to process new bug reports in real-time.

4.4.4. Other Description of Module 2

Cosine similarity is chosen for its effectiveness in measuring the similarity between high-dimensional vectors, which is essential for accurate duplicate detection. Alternatives to cosine similarity include Euclidean distance and Jaccard similarity:

- **Euclidean Distance:**
  - Measures the straight-line distance between two points in a vector space. It is not ideal for high-dimensional data as it can suffer from the curse of dimensionality, making it less effective for text similarity.
- **Jaccard Similarity:**
  - Measures the similarity between two sets by dividing the size of the intersection by the size of the union of the sets. It works well for binary attributes but is less effective for continuous or high-dimensional data.

Cosine similarity is preferred in this context because it normalizes the length of the vectors, focusing on the orientation rather than the magnitude, which is more suitable for text data represented as TF-IDF vectors.

In summary, the duplicate detection module leverages the strengths of cosine similarity to identify and retrieve the most similar bug reports, ensuring efficient and accurate duplicate detection. This module is crucial for maintaining a clean and manageable bug report database, ultimately enhancing the bug triaging process.

4.5 Module 3: Priority Prediction

Priority prediction involves determining the urgency and importance of bug reports to ensure that critical issues are addressed promptly.

4.5.1. Functional Description

The priority prediction module uses machine learning algorithms to predict the priority level of bug reports based on their feature vectors. This helps in efficiently allocating resources to resolve high-priority issues first.

4.5.2. Modular Decomposition

The priority prediction module consists of the following submodules:

- **Feature Extraction:**
    - **Function:**
        - Extracts relevant features from the bug report text using the Term Frequency-Inverse Document Frequency method.
- **Model Training:**
    - **Function:**
        - Trains a logistic regression model on historical bug report data to learn the relationship between features and priority levels.
- **Prediction:**
    - **Function:**
        - Applies the trained logistic regression model to new bug reports to predict their priority levels.
- **Threshold Adjustment:**
    - **Function:**
        - Adjusts the decision threshold based on the desired balance between precision and recall.

4.5.3. Design Constraints

The priority prediction module must handle large volumes of data and produce results in real-time. The model should be retrained periodically to adapt to new patterns in the bug report data. Ensuring the model's interpretability and fairness is also crucial, as it directly impacts resource allocation decisions.

4.5.4. Other Description of Module 3

Logistic regression is chosen for its simplicity and effectiveness in binary and multi classification tasks. It provides probabilistic outputs that can be easily interpreted and adjusted. Alternatives to logistic regression include decision trees and support vector machines:

- **Decision Trees:**
    - These models can capture complex relationships in the data but are prone to overfitting, especially with small datasets.
- **Naive Bayes:**
    - Naive Bayes models assume that the features are independent given the class label, making them simple and computationally efficient. They perform well with small datasets and are particularly effective for

text classification tasks. However, their strong independence assumption may lead to suboptimal performance when the features are highly correlated.

- **Random Forest:**
    - Random Forest models are an ensemble learning method that builds multiple decision trees and merges their results to improve accuracy and control overfitting. They can handle large datasets with higher dimensionality and provide good generalization. However, they require more computational resources and can be less interpretable compared to single decision trees due to their complexity.

Classifier	Accuracy
Decision Trees	87%
Naive Bayes	91.3%
Random Forest	91.1%
Logistic Regression	91.6%

Table 2: Bug priority trials

Logistic regression is preferred in this context because it balances interpretability and performance, making it suitable for real-time priority prediction in an automated bug triaging system.

In summary, the priority prediction module leverages machine learning to automate the prioritization of bug reports, ensuring that critical issues are addressed promptly and resources are allocated efficiently. This module is essential for maintaining high software quality and user satisfaction.

4.6. Module 4: Automatic Categorization

Automatic categorization is a crucial step in the bug triaging process, where bug reports are classified into predefined categories based on their content. This helps in organizing and managing bug reports more efficiently.

4.6.1. Functional Description

The automatic categorization module uses machine learning algorithms to classify bug reports into different categories such as UI, performance, security, etc. The classification helps developers to quickly identify the nature of the bug and assign it to the appropriate team or individual.

4.6.2. Modular Decomposition

The automatic categorization module consists of the following submodules:

1. **Feature Extraction:**
    - **Function:**
        i. Extracts relevant features from the bug report text using TF-IDF method.
    - **Example:**
        i. Converts the text into a numerical representation where the weight of each term reflects its importance in the document and the entire corpus.
2. **Model Training:**
    - **Function:**
        i. Trains a SVM model on historical bug report data to learn the relationship between features and categories.
    - **Example:**
        i. The SVM model uses the following decision function to classify a bug report:

            $$f(x) = w^T . x + b$$

            1. w is the weight vector
            2. x is the feature vector
            3. b is the bias
            4. The classification is determined by the sign of f(x)
3. **Prediction:**
    - **Function:**
        i. Applies the trained SVM model to new bug reports to predict their categories.

- **Example:**
    1. Given a new bug report with its feature vector x, the model outputs the predicted category based on the decision function f(x).

4. **Threshold Adjustment:**
    - **Function:**
        1. Adjusts the decision boundary based on the desired balance between precision and recall.
    - **Example:**
        1. Tuning the SVM parameters regularization parameter and kernel parameter to optimize the model performance for specific categories.

4.6.3. Design Constraints

The automatic categorization module must handle large volumes of data and produce results in real-time. The model should be periodically retrained to adapt to new patterns in the bug report data. Ensuring the model's interpretability and fairness is also crucial, as it impacts the accuracy of the bug categorization.

4.6.4. Other Description of Module 4

SVM is chosen for its effectiveness in high-dimensional spaces and its robustness in handling imbalanced data. Alternatives to SVM include:

- **Random Forest:**
    - Random Forest models are an ensemble learning method that builds multiple decision trees and merges their results to improve accuracy and control overfitting. They can handle large datasets with higher dimensionality and provide good generalization. However, they require more computational resources and can be less interpretable compared to single decision trees due to their complexity.
- **Label Spreading:**
    - Label Spreading is a semi-supervised learning algorithm that uses both labeled and unlabeled data to improve classification performance. It spreads labels through the data points based on their similarity, allowing the model to make better use of the available information. This method is particularly effective when labeled data is scarce. However, it can be computationally intensive and may struggle with large datasets. Additionally, the performance of Label Spreading heavily depends on the choice of the similarity measure and the underlying data distribution.

Algorithm	Accuracy
Logistic regression	83.7%
Random Forest	62.2%
Label Spreading (Unsupervised)	84%
SVM	94.5%

Table 3: Bug Categorization trials

SVM is preferred for automatic categorization due to its strong theoretical foundation and ability to handle high-dimensional data, making it suitable for text classification tasks in an automated bug triaging system.

In summary, the automatic categorization module leverages machine learning to classify bug reports into predefined categories, helping in organizing and managing bug reports more efficiently. This module is essential for streamlining the bug triaging process and ensuring that bug reports are directed to the appropriate teams or individuals for resolution.

4.7. Module 5: developer prediction

Predicting the developer responsible for resolving a bug is a crucial step in the bug triaging process. This task involves using machine learning algorithms to analyze historical bug reports and predict the most suitable developers for new reports. By accurately identifying the top 5 developers likely to resolve an issue, this process helps in organizing and managing bug reports more efficiently, ensuring that bugs are quickly assigned to the right individuals for resolution.

4.7.1. Functional Description

The developer prediction module uses machine learning algorithms to predict the top 5 developers likely to resolve new bug reports. This model leverages historical data to learn patterns associated with specific developers, aiding in the efficient assignment of bug reports. By predicting the most suitable developers, the model helps streamline the bug triage process, ensuring faster and more accurate resolution of issues.

4.7.2. Modular Decomposition

The classification module consists of the following submodules:

- **Dataset exploring and cleaning:**
    - **Function:**
        - Performs various preprocessing steps on multiple datasets to prepare them for analysis, including cleaning the data to ensure quality and consistency.
    - **Steps:**
        - Data Acquisition:
            - Kaggle Dataset: Collected a dataset from Kaggle.
            - Eclipse and Mozilla Datasets: Gathered datasets from different products and components in Bugzilla.
        - Initial Exploration: Printed the shape (number of rows and columns) and columns of each dataset to understand their structure.
        - Data Cleaning:
            - **Remove Duplicates:** Identified and removed duplicate rows to ensure each entry is unique.
            - **Filter Resolutions:** Removed rows where the 'Resolution' column is marked as 'DUPLICATE' or 'INVALID' to focus on valid bug reports.
            - **Drop Unnecessary Columns:** Removed columns that are not needed for the analysis to simplify the dataset.
            - **Handle Null Values:** Removed rows with null values to ensure the dataset is complete.
            - **Summary Text Cleaning:** Cleaned the text in the 'Summary' column by removing special characters, newlines, and hyperlinks to standardize the text data.
- **Preprocessing:**
    - **Function:**
        - Combines and preprocesses multiple datasets to create a unified and cleaned dataset, ready for analysis. This includes merging datasets, text preprocessing, and filtering based on specific criteria.
    - **Steps:**
        - Dataset Merging: Combined the Kaggle, Eclipse, and Mozilla datasets into a single dataset.
        - Data Cleaning:
            - **Remove Duplicates:** Identified and removed duplicate rows from the merged dataset to ensure each entry is unique.

        - **Filter Short Summaries:** Removed rows where the 'Summary' column contains fewer than 10 words to ensure sufficient detail in the text data.
    - Text Preprocessing:
        - **Tokenization:** Applied tokenization to the 'Summary' column, breaking down the text into individual words (tokens).
        - **Remove Stop Words:** Removed common stop words (e.g., 'the', 'and', 'is') from the tokens to focus on the most meaningful words.
        - **Stemming:** Applied stemming to reduce words to their base or root form (e.g., 'running' becomes 'run').
        - **Rejoin Tokens:** Joined the processed tokens back into a single string for each entry in the 'Summary' column.
    - Data Filtering:
        - **Owner Occurrence:** Filtered the dataset to include only those entries where each owner has at least 5 occurrences, ensuring enough data for each owner.
- **Feature Vector Generation:**
    - **Function:**
        - Uses TF-IDF to convert bug report text into numerical feature vectors.
    - **Steps:**
        - Generates a vector where each dimension represents the importance of a term in the document and across the entire corpus.
- **Model Training:**
    - **Function:**
        - Trains a Support Vector Machine (SVM) model with a linear kernel to predict the developer responsible for solving a bug based on historical bug reports.
    - **Steps:**
        - **Model Selection:** Chose an SVM model with a linear kernel for its simplicity and effectiveness in high-dimensional spaces.
        - **Hyperparameter Setting:**Set the regularization parameter CCC to 10. The parameter CCC controls the trade-off between achieving a low training error and a low testing error, which helps in avoiding overfitting.
        - **Training Data:** Utilized historical bug reports as the training data. These reports included features extracted during preprocessing and the target variable, which is the developer (assignee) who resolved the bug.

        - **Training Process:** Fit the SVM model on the training dataset, where the input features were the processed bug report texts and the target labels were the developers.

- **Prediction:**
    - **Function:**
        - Uses the trained SVM model to predict the top 5 developers likely to resolve new bug reports.

4.7.3. Design Constraints

The clustering module must handle high-dimensional feature vectors efficiently and produce clusters in a timely manner. The choice of kkk (number of clusters) is crucial and can significantly impact the clustering quality. The algorithm should be scalable to accommodate a growing number of bug reports.

4.7.4. Other Description of Module 5

SVM was chosen for its effectiveness in high-dimensional spaces and its robustness in handling imbalanced data. SVMs are particularly well-suited for text classification tasks, as they can effectively manage the sparse nature of text data and identify complex decision boundaries.

Hyperparameter Tuning:

Various hyperparameters were explored to optimize the SVM model, with the best model chosen based on validation accuracy:

- **C Values:** Different values of CCC (0.1, 1, 10, 100) were tested. The regularization parameter CCC controls the trade-off between achieving a low training error and a low testing error. Through experimentation, it was determined that C = 10 provided the best validation accuracy, balancing both underfitting and overfitting.
- **Kernels:** Both linear and RBF (Radial Basis Function) kernels were evaluated. The linear kernel was found to be more suitable for this specific task, as it performed better with the high-dimensional, sparse nature of the text data.
- **Class Weights:** The impact of class weights (balanced, none) was investigated to address class imbalance. Although balanced class weights can adjust for imbalances, the results indicated that the default setting (none) provided better validation accuracy.

Impact of SVD (Singular Value Decomposition):

SVD was applied after TF-IDF and before SVM to reduce the dimensionality of the feature space. However, the results showed that the performance was better without applying SVD. The dimensionality reduction introduced by SVD potentially led to the loss of important information, thereby affecting the classifier's ability to accurately predict the developer.

In summary, The developer prediction module leverages SVM with a linear kernel and C=10 to predict the top 5 developers likely to resolve new bug reports. Despite evaluating alternative models and various hyperparameters, SVM was preferred due to its strong theoretical foundation and suitability for high-dimensional data, making it ideal for this text classification task. This module is essential for streamlining the bug triaging process, ensuring efficient and accurate assignment of bug reports to the appropriate developers for resolution.

4.8. Module 6: Developers Recommender

Bug assignment is a critical aspect of software development, where efficient allocation of bugs to developers can significantly impact productivity and resolution time. Traditional methods often rely on historical data and human judgment, which can be time-consuming and prone to biases. In response, we developed a recommender module that leverages machine learning to automate bug assignment, ensuring that bugs are directed to developers based on their past performance and expertise, even for developers not present in the training data.

4.8.1. Functional Description

The recommender module operates by receiving developers associated with the detected bug category and their historical bug-solving records. It utilizes a classifier trained on historical bug data to predict the top 5 likely bug categories for each developer's previous bug solutions and for the input bug. The module then selects the top 5 developers based on the overlap between the predicted bug categories for the input bug and those for each developer. This approach ensures that bugs are assigned to developers who have demonstrated proficiency in resolving similar issues, optimizing bug resolution efficiency.

4.8.2. Modular Decomposition

The Developer Recommender module consists of the following submodules:

- **Bug Category Detection:**
  - **Function:** Identifies the category of the input bug.
- **Developer Profiling:**
  - **Function:** Collects and profiles developers associated with the detected bug category.
  - **Implementation:** Gathers historical data on bugs resolved by each developer within the category, forming a basis for their bug-solving expertise.
- **Classifier Application:**
  - **Function:** Applies the trained classifier to predict bug categories for each developer's historical bug-solving records and for the input bug.
  - **Implementation:** Utilizes the classifier's predictions to rank bug categories and determine the top 5 relevant categories for both developers and the input bug.
- **Developer Selection:**
  - **Function:** Selects the top 5 developers based on the intersection of the predicted bug categories for the input bug and each developer's historical bug-solving records.

  - **Implementation:** Computes the overlap between the predicted categories, ensuring that developers with the most relevant expertise are prioritized for bug assignment.

4.8.3. Design Constraints

The recommender module must meet several design constraints to ensure its effectiveness and reliability:

- **Scalability:** Capable of handling large volumes of bug data and developers across multiple categories to support scalable bug assignment.
- **Real-Time Performance:** Provides bug assignment recommendations in real-time to maintain workflow efficiency and responsiveness.
- **Accuracy and Fairness:** Ensures that bug assignments are based on objective criteria (predicted bug categories) to mitigate biases and ensure fair distribution of workload.
- **Interpretability:** Offers transparency in bug assignment decisions by providing clear rationale based on predicted bug categories and developer expertise.

4.8.4. Other Description of Module 6

The Recommender System enhances the bug triaging process by ensuring that bugs are assigned to the most capable developers. This leads to quicker and more effective bug resolution, improving the overall quality and reliability of the software.

The system's effectiveness is largely dependent on the quality of the developer profiles and the accuracy of the classifier. To this end, the module continuously updates developer profiles with new data and feedback, ensuring that recommendations remain relevant and accurate.

In summary, the Recommender System module plays a crucial role in optimizing the bug resolution process by matching bugs with the best-suited developers. By leveraging historical data and machine learning techniques, it ensures that bug reports are handled efficiently and effectively, ultimately enhancing the software development lifecycle.

4.9. Module 7: Web Interface

The Web Interface module provides a user-friendly platform for interacting with the Automated Bug Triaging System. This module ensures that users can easily submit bug reports, track the status of bugs, view recommendations, and access analytics and insights.

4.9.1. Functional Description

The Web Interface module facilitates seamless interaction between users and the underlying system. It includes various features such as bug report submission forms, dashboards for tracking bug statuses, and displays for recommendations and analytics. This module aims to provide an intuitive and efficient user experience.

4.9.2. Modular Decomposition

The Web Interface module consists of the following submodules:

- **Bug Report Submission:**
  - **Function:**
    - Allows users to submit bug reports through an easy-to-use form.
  - **Example:**
    - Users can fill in details such as bug description, steps to reproduce, and severity level.
  - **Design Considerations:**
    - Ensure form validation and user input sanitization to prevent errors and malicious inputs.
- **Bug Status Dashboard:**
  - **Function:**
    - Displays the status of submitted bug reports, including current state (e.g., new, in progress, resolved), assigned developer, and priority.
  - **Example:**
    - A project manager can view all open bugs, their statuses, and assigned developers in a single dashboard.
  - **Design Considerations:**
    - Provide real-time updates and easy navigation to enhance user experience.
- **Recommendation Display:**
  - **Function:**
    - Shows the recommended developers for each bug report, along with their profiles and similarity scores.

    - **Example:**
        - After submitting a bug report, the user can see the top 5 recommended developers for resolving the bug.
    - **Design Considerations:**
        - Present recommendations in a clear, accessible format to aid in decision-making.
- **Analytics and Insights:**
    - **Function:**
        - Provides visual reports and analytics on bug trends, resolution times, and developer performance.
    - **Example:**
        - A QA manager can view graphs showing the average time to resolve bugs or the number of bugs assigned to each developer.
    - **Design Considerations:**
        - Use interactive charts and graphs to make data easily understandable and actionable.
- **User Authentication and Authorization:**
    - **Function:**
        - Manages user login and permissions, ensuring that only authorized users can access specific features.
    - **Example:**
        - Developers can log in to view and update their assigned bugs, while project managers can access overall project analytics.
    - **Design Considerations:**
        - Implement secure authentication mechanisms and role-based access control.

4.9.3. Design Constraints

The Web Interface must be responsive, ensuring usability across different devices and screen sizes. It should handle concurrent user sessions efficiently and maintain performance even with a high volume of data and user interactions.

4.9.4. Other Description of Module 7

The Web Interface is the primary touchpoint for users interacting with the Automated Bug Triaging System. Its design focuses on ease of use, accessibility, and providing comprehensive information in a clear, organized manner.

By offering an intuitive interface for submitting bug reports, tracking statuses, viewing recommendations, and accessing analytics, this module ensures that users can efficiently manage the bug triaging process. It bridges the gap between complex backend processing and user needs, delivering a smooth and effective user experience.

In summary, the Web Interface module is crucial for the overall functionality of the Automated Bug Triaging System, providing users with the tools they need to manage bugs effectively and enhancing their ability to make informed decisions based on real-time data and insights.

Conclusion

The detailed examination of the seven modules within our Automated Bug Triaging System highlights their individual functionalities and the cohesive framework they form together to streamline bug management processes. Here is a concise conclusion of each module discussed in Chapter 4:

Module 1: Preprocessing and Feature Extraction

The preprocessing and feature extraction module is foundational to the system, ensuring raw bug report data is transformed into a consistent and analyzable format. This module performs critical tasks such as data cleaning, tokenization, and vectorization, which are essential for the accuracy and effectiveness of subsequent machine learning processes. By standardizing text data and extracting meaningful features using TF-IDF, this module lays the groundwork for accurate bug categorization, prioritization, and duplicate detection.

Module 2: Duplicate Detection

The duplicate detection module utilizes cosine similarity to identify and merge duplicate bug reports. This module is pivotal in reducing redundant efforts by developers, ensuring that resources are not wasted on resolving the same issue multiple times. By retrieving the top k duplicates for any given bug report, this module enhances the efficiency of the bug triaging process and contributes to maintaining a clean and organized bug tracking system.

Module 3: Priority Prediction

The priority prediction module employs logistic regression to determine the priority level of each bug report based on its features. This module ensures that critical bugs are identified and addressed promptly, improving software reliability and user satisfaction. By accurately predicting the priority of bug reports, this module helps development teams allocate their resources more effectively, focusing on issues that have the most significant impact.

Module 4: Automatic Categorization

The automatic categorization module uses Support Vector Machine to classify bug reports into predefined categories. This module enhances the consistency and accuracy of bug management by ensuring that similar bugs are grouped together and handled appropriately. Accurate categorization facilitates better organization of bug reports and allows development teams to specialize in specific types of issues, improving overall efficiency.

Module 5: Developer Prediction Model

The developer prediction model leverages Support Vector Machine (SVM) to predict the top 5 developers most likely to resolve new bug reports based on historical data. This module significantly improves the efficiency of bug triaging by automating the assignment process and ensuring that bugs are directed to developers with relevant expertise. By accurately predicting the most suitable developers, the model reduces the time required for manual bug assignment and enhances the overall productivity of the development team.

Module 6: Recommender Model

The recommender model builds on the developer prediction model by extending its capabilities to developers not present in the training data. It receives all developers within the detected category and their historical bug-solving records, then applies the classifier to predict the top bug categories for each developer. By matching these categories with the top categories of the input bug, the module selects the top 5 developers with the most relevant expertise. This approach ensures that bugs are assigned efficiently and fairly, leveraging historical data to optimize bug resolution processes and improve the effectiveness of bug management.

Module 7: Web Interface

The web interface module offers a user-friendly platform for users to interact with the system. It allows for the submission of bug reports, tracking of bug statuses, viewing of recommended actions, and accessing of analytics. This module ensures that the system is accessible and easy to use for all stakeholders, providing a seamless experience that integrates all functionalities of the automated bug triaging system.

In conclusion, the comprehensive design and implementation of these modules collectively enhance the efficiency, accuracy, and effectiveness of bug management in software development. Each module plays a critical role in automating and optimizing different aspects of the bug triaging process, contributing to an overall system that improves software quality and developer productivity.

Chapter 5: System Testing and Verification

This chapter details the steps taken to ensure the correctness and reliability of the Automated Bug Triaging System. We describe the testing setup, strategies, and environments used to validate the system. This includes unit testing for individual modules and integrated system testing for the complete project. Finally, the results from various testing scenarios are highlighted and discussed, demonstrating the system's performance and accuracy.

5.1. Testing Setup

The testing setup for our project includes the following components:

- **Development Environment:**
    - The system is developed and tested on a development environment using Jupyter Notebook for initial tests and validation.
- **Testing Framework:**
    - We use Selenium for end-to-end testing for the website.
- **Test Data:**
    - We use a combination of synthetic data and real-world bug reports from open-source projects like Eclipse and Mozilla Firefox to ensure comprehensive testing.
- **Software Tools:**
    - Python, sklearn, numpy, pandas, and nltk libraries are used for implementing and testing the machine learning models.
    - Selenium for end-to-end testing.

5.2. Testing Plan and Strategy

Our testing methodology follows a structured approach to ensure all components and functionalities of the system are thoroughly tested. The key aspects of our testing strategy are:

- **Unit Testing:**
    - Each module of the system is tested individually to validate its functionality and performance.
- **Integration Testing:**
    - The integrated system is tested to ensure all modules work together seamlessly and the overall functionality meets the requirements.

- **Accuracy Testing:**
    - The accuracy of the machine learning models is tested using appropriate metrics like precision, recall, and F1 score.

5.2.1. Module Testing

- **Preprocessing Module**
    - **Test Description:**
        - The preprocessing module is tested to ensure it correctly converts text to lowercase, removes punctuation, numbers, stopwords, and performs lemmatization.
    - **Test Cases:**
        - Input: "It's a test bug report, number 1234."
        - Expected Output: "test bug report"
    - **Results:**
        - The module successfully preprocesses the text as expected.
- **Feature Extraction Module**
    - **Test Description:**
        - The feature extraction module is tested to ensure it correctly transforms text data into TF-IDF vectors.
    - **Test Cases:**
        - Input: ["test bug report", "bug issue resolved"]
        - Expected Output: TF-IDF vectors representing the importance of terms in each document.
    - Results:
        - The module successfully generates TF-IDF vectors, with the correct term frequencies and document frequencies.
- **Bug Categorization Module**
    - **Test Description:**
        - The SVM-based bug categorization module is tested for its accuracy in categorizing bug reports.
    - **Test Cases:**
        - Input: Preprocessed bug reports labeled with categories.
        - Expected Output: Correctly predicted categories for the test bug reports.
    - **Results:**
        - The SVM model achieves an accuracy of 94.5%, indicating reliable categorization.
- **Duplicate Detection Module**
    - **Test Description:**

        - The duplicate detection module using cosine similarity is tested to identify duplicate bug reports.
    - **Test Cases:**
        - Input: Preprocessed bug reports with known duplicates.
        - Expected Output: Correct identification of duplicate bug reports.
    - **Results:**
        - The module achieves a accuracy of 97.25% and recall of 86.7%, demonstrating effective duplicate detection.

- **Developer Assignment Module**
    - **Test Description:**
        - The SVM-based developer assignment module is tested for its accuracy in assigning bug reports to the appropriate developer.
    - **Test Cases:**
        - Input: Preprocessed bug reports labeled with assigned developer.
        - Expected Output: Correctly predicted developers for the test bug reports.
    - **Results:**
        - The SVM model achieves an accuracy of 74.1% in predicting the correct developer with the highest probability, and 96.5% in including the correct developer within the top-5 predictions, demonstrating reliable developer assignment.

5.2.2. Integration Testing

- **Test Description:**
    - The integrated system is tested to ensure all modules work together seamlessly. This includes testing the end-to-end flow from bug report submission to categorization, prioritization, developer assignment, and duplicate detection.
- **Test Cases:**
    - Input:
        - A batch of preprocessed bug reports.
    - Expected Output:
        - Correct categorization, prioritization, developer assignment, and identification of duplicates.
    - Results:
        - The integrated system performs as expected, with accurate categorization, prioritization, and duplicate detection. The developer assignment module also assigns bugs to the appropriate developers based on their expertise.

5.2.3 System Testing

- **Test Description:**
  - System testing involves verifying the overall functionality and performance of the entire system. This stage ensures that the system meets all specified requirements and operates efficiently under various conditions. This testing phase includes both API testing and end-to-end (E2E) testing on the website module.
- **End-to-End (E2E) Testing:**
  - **Objective:**
    - Ensure the website module functions correctly from a user's perspective, covering the entire workflow.
  - **Sample Test Case: Bug Report Submission**
    - **Input:** Bug report details entered in the web form.
    - **Expected Output:** Bug report is submitted successfully and appears in the bug tracking system.
    - **Steps:**
      - Navigate to the bug report submission page on the website.
      - Fill in the bug report form with the necessary details.
      - Submit the form.
      - Verify the submission success message and check the bug tracking system for the new report.
    - **Results:** The bug report is successfully submitted and recorded in the system.
  - **Test Case 2: Bug Report Management**
    - **Input:** Various actions performed by the user (e.g., viewing, categorizing, prioritizing bug reports).
    - **Expected Output:** The system correctly handles all actions and updates the bug report status accordingly.
    - **Steps:**
      - Log in as a user with appropriate permissions.
      - View the list of bug reports.
      - Perform actions such as categorizing, prioritizing, and assigning bug reports.
      - Verify that the actions are reflected accurately in the system.
    - **Results:** The system correctly handles all user actions, and the bug reports are updated as expected.

5.3. Testing Schedule

Month	Testing Activities
March	Unit testing of preprocessing and feature extraction modules
April	Unit testing of bug categorization and duplicate detection modules
April	Accuracy testing
May	Integration testing of the entire system
June	E2E Testing
July	Final review and validation.

Table 4: Testing Schedule

5.4. Comparative Results to Previous Work

We compare the performance of our Automated Bug Triaging System with previous works in the field. The comparison focuses on efficiency, accuracy, scalability, and integration capabilities.

Feature	Previous Work	Our System
Categorization	85%	94.5%
Duplicate Detection	95%	97.25%
Prioritization	90%	94.3%
Developer prediction	68.2%	74.18%

Table 5: Final models

Commentary: Our system outperforms previous works in terms of accuracy and precision for bug categorization, developer prediction and duplicate detection. It also offers better scalability and more comprehensive integration with existing development tools, making it a robust and efficient solution for automated bug triaging.

In conclusion, the testing and verification processes demonstrate that our Automated Bug Triaging System meets the desired specifications and performs reliably across various scenarios. The detailed testing plan ensures that all aspects of the system are thoroughly validated, providing confidence in its deployment and use.

Chapter 6: Conclusions and Future Work

This chapter provides a comprehensive summary of the Automated Bug Triaging System project, highlighting its features and limitations. It also discusses the challenges faced and the skills gained during the project. Finally, directions for future work are outlined to guide further research and development in this area.

6.1. Faced Challenges

Throughout the development of the Automated Bug Triaging System, several challenges were encountered:

- Data Quality and Availability:
  - Ensuring high-quality and comprehensive datasets was crucial. Inconsistent or incomplete data could lead to inaccurate predictions. This was mitigated by implementing robust data preprocessing steps to clean and normalize the data before training the models.
- Model Performance and Accuracy:
  - Achieving high accuracy in bug categorization, prioritization, and developer assignment models was challenging due to the diversity and complexity of bug reports. Extensive hyperparameter tuning and cross-validation techniques were used to optimize model performance. Continuous learning mechanisms were also implemented for the models to adapt to new data and user feedback.
- Integration Complexity:
  - Integrating various modules into a seamless system required careful planning and execution. This was addressed by designing a modular architecture with well-defined interfaces and employing comprehensive integration testing to ensure smooth interactions between components.
- Scalability:
  - Ensuring the system could handle large volumes of bug reports efficiently was a critical concern. This was managed by leveraging scalable cloud infrastructure and implementing efficient algorithms for data processing and model training.

6.2. Gained Experience

Working on the Automated Bug Triaging System project provided valuable experience and skills, including:

- **Advanced NLP Techniques:**
  - Gained in-depth knowledge of natural language processing techniques, including TF-IDF.
- **Machine Learning Proficiency:**
  - Developed expertise in training and optimizing machine learning models, such as SVM and logistic regression, for specific tasks like bug categorization and prioritization.
- **Data Preprocessing:**
  - Learned the importance of data preprocessing and the various techniques used to clean and prepare data for analysis.
- **Software Integration:**
  - Acquired skills in integrating different software modules and ensuring they work together seamlessly within a complex system.
- **Scalability and Performance Optimization:**
  - Gained experience in designing scalable systems and optimizing performance to handle large datasets efficiently.

6.3. Conclusions

The Automated Bug Triaging System is a comprehensive solution designed to automate the process of handling and prioritizing bug reports. Key features of the system include:

- **Automated Categorization:**
  - Uses NLP techniques and SVM to accurately categorize bug reports into predefined categories.
- **Efficient Prioritization:**
  - Employs logistic regression to prioritize bugs based on severity, user impact, and business priorities.
- **Intelligent Developer Assignment:**
  - Utilizes Support Vector Machine (SVM) to predict the top 5 developers most likely to resolve new bug reports based on historical data, enhancing the efficiency and accuracy of the bug triaging process.
- **Effective Duplicate Detection:**
  - Implements TF-IDF and cosine similarity to identify and merge duplicate bug reports, preventing redundant work.

Limitations:

- **Dependency on Data Quality:**
  - The accuracy of the system is heavily dependent on the quality of the input data. Poor data quality can lead to inaccurate predictions.
- **Integration Challenges:**
  - Integrating with diverse development environments may require additional customization and effort.

6.4. Future Work

The project opens up several avenues for future research and development:

- **Enhanced Models:**
  - Exploring more advanced models and techniques, such as ensemble methods or neural network architectures, to further improve accuracy and efficiency.
- **Real-time Processing:**
  - Developing capabilities for real-time processing and triaging of bug reports to provide instant feedback and resolution recommendations.
- **Expanded Integration:**
  - Enhancing integration with a broader range of development tools and platforms to ensure wider applicability and ease of adoption.
- **User Feedback Loop:**
  - Implementing a user feedback loop to continuously improve the system based on user inputs and experiences.
- **Security Enhancements:**
  - Adding security features to protect sensitive data within bug reports and ensure compliance with data protection regulations.
- **Scalability Improvements:**
  - Further optimizing the system for scalability to handle even larger datasets and more complex bug reports efficiently.

In conclusion, the Automated Bug Triaging System provides a robust and efficient solution for managing bug reports, leveraging advanced NLP and machine learning techniques. The experiences gained and the challenges overcome during this project lay a strong foundation for future enhancements and developments in automated bug triaging.

References

1. Label Propagation for Deep Semi-supervised Learning
2. Machine Learning-based Software Bug Classification through Mining Open Source Repositories
3. CaPBug-A Framework for Automatic Bug Categorization and Prioritization Using NLP and Machine Learning Algorithms
4. Automatic Bug Tracking System Using Text Analysis and Machine Learning Predictions
5. A Comparative Study of Transformer-based Neural Text Representation Techniques on Bug Triaging
6. understanding-tf-idf-term-frequency-inverse-document-frequency
7. tf-idf-in-nlp-term-frequency-inverse-document-frequency
8. feature_extraction.text
9. Approaches for automated bug triaging: A review
10. Automated Bug Triage Using Machine Learning Algorithms
11. Enhancing Bug Triage with NLP Techniques

Appendix A: Development Platforms and Tools

This appendix provides a comprehensive overview of the hardware and software platforms and tools utilized in the development of the Automated Bug Triaging System. Each tool, platform, and module is described in detail, highlighting its role and significance in the project. The appendix is organized into two main sections: hardware and software. Within each section, individual subsections provide specific details about each tool or platform used.

A.1. Hardware Platforms

A.1.1. Development Servers

The development servers used in this project were essential for hosting the machine learning models, databases, and the web application. These servers provided the necessary computational power and storage to support the development and testing phases.

- **Specifications:**
    - Processor: 1 core
    - RAM: 1 GB
    - Storage: 25 GB
    - Operating System: Linux
- Role in the Project:
    - The servers were used to train machine learning models, run preprocessing scripts, and host the backend services. Their high processing power and memory ensured efficient handling of large datasets and complex computations.

A.1.2. Cloud Infrastructure

The project leveraged cloud infrastructure for scalable deployment and testing. Services from major cloud providers such as AWS and Google Cloud were utilized to ensure high availability and reliability.

- **AWS Services:**
    - EC2 Instances:
        - Used for running virtual servers to host various components of the system.
- **Role in the Project:**
    - Cloud infrastructure provided flexibility in scaling resources up or down based on the project's needs. It also facilitated collaboration among

team members by providing a centralized platform for development and testing.

A.2. Software Tools

A.2.1. Programming Languages and Libraries

- **Python:**
  - Role:
    - Python was the primary programming language used for developing machine learning models and preprocessing scripts.
  - Key Libraries:
    - Scikit-Learn:
      - Used for implementing machine learning algorithms such as SVM and logistic regression.
    - NLTK:
      - Used for natural language processing tasks such as tokenization and lemmatization.
- **JavaScript:**
  - Role:
    - JavaScript, along with libraries like React, was used for developing the frontend interface of the web application.
  - Key Libraries:
    - React:
      - Used for building dynamic and interactive user interfaces.
    - Node.js:
      - Used for server-side scripting and backend development.
    - Express.js:
      - Used for building RESTful APIs.

A.2.2. Development Frameworks

- ExpressJS:
  - Role:
    - Used as the primary web framework for developing the backend of the web application.
  - Features:
    - ExpressJS provided an integrated environment for rapid development, including ORM for database interactions, built-in security features, and an admin interface for managing application data.

A.2.3. Integrated Development Environments (IDEs)

- Visual Studio Code:
  - Role:
    - Used for JavaScript development.
  - Features:
    - Visual Studio Code offered a lightweight and versatile environment with extensions for React, Node.js, and other web technologies.

A.2.4. Version Control and Collaboration Tools

- Git:
  - Role:
    - Used for version control to manage code changes and collaboration among team members.
  - Features:
    - Git allowed for branching, merging, and tracking changes, which facilitated efficient collaboration and code management.
- GitHub:
  - Role:
    - Used as the remote repository for storing and sharing code.
  - Features:
    - GitHub provided tools for code review, issue tracking, and continuous integration, enhancing project management and collaboration.

A.2.5. Data Storage and Management

- MongoDB:
  - Role:
    - Used as the primary database for storing bug reports, user data, and model outputs.
  - Features:
    - MongoDB's flexible schema and powerful querying capabilities made it suitable for handling diverse and dynamic data structures.
- Postman:
  - Role:
    - Used for API testing and documentation.
  - Features:

- Postman allowed for the creation, testing, and documentation of APIs, ensuring they functioned correctly and met the project's requirements.

These hardware and software platforms and tools were instrumental in the successful development and deployment of the Automated Bug Triaging System, ensuring robust performance, scalability, and user satisfaction.

Appendix B: Use Cases

This appendix provides a comprehensive overview of the use cases for the Automated Bug Triaging System. Each use case is described in detail, highlighting the interactions between users and the system, as well as the expected outcomes. These use cases demonstrate how the system addresses various needs and scenarios in the software development lifecycle.

B.1. Submitting a Bug Report

- Actors:
  - End Users, Developers, QA Teams
- Description:
  - Users can submit bug reports through the system's interface.
- Preconditions:
  - The user must be logged into the system.
- Steps:
  - The user navigates to the "Submit Bug Report" section.
  - The user fills out the bug report form, providing details such as the description, steps to reproduce, and any relevant attachments.
  - The user submits the form.
  - The system preprocesses the bug report, converting text to lowercase, removing punctuation, numbers, and stopwords, and performing lemmatization.
  - The system categorizes the bug using an SVM model.
  - The system assigns a priority to the bug using logistic regression.
  - The system checks for duplicate bug reports using TF-IDF and cosine similarity.
  - The system notifies the relevant stakeholders of the new bug report.
- Postconditions:
  - The bug report is added to the system and assigned to a developer for resolution.
- Exceptions:
  - If the form is incomplete, the system prompts the user to fill in the missing details.

B.2. Viewing and Managing Bug Reports

- Actors:
  - Developers, QA Teams, Project Managers
- Description:
  - Users can view and manage bug reports assigned to them.
- Preconditions:
  - The user must be logged into the system.
- Steps:
  - The user navigates to the "View Bug Reports" section.
  - The system displays a list of bug reports assigned to the user.
  - The user can filter and sort the bug reports by various criteria (e.g., priority, category, status).
  - The user selects a bug report to view its details.
  - The user can update the status of the bug report, add comments, and upload additional attachments if necessary.
  - The system saves the updates and notifies the relevant stakeholders.
- Postconditions:
  - The bug report is updated with the latest information.
- Exceptions:
  - If the user tries to update a bug report without proper permissions, the system displays an error message.

B.3. Generating Analytics and Reports

- Actors:
  - Project Managers, QA Teams
- Description:
  - Users can generate analytics and reports to gain insights into bug trends and resolution times.
- Preconditions:
  - The user must be logged into the system.
- Steps:
  - The user navigates to the "Analytics Dashboard" section.
  - The user selects the desired metrics and parameters for the report (e.g., time period, bug category, developer performance).
  - The system retrieves the relevant data from the database.
  - The system generates visual reports, such as graphs and charts, displaying the selected metrics.
  - The user can download the reports or share them with other stakeholders.
- Postconditions:

- ○ The user receives detailed analytics and reports.
- Exceptions:
  - If there is insufficient data for the selected parameters, the system notifies the user and suggests alternative parameters.

B.4. Receiving Notifications and Reminders

- Actors:
  - Developers, QA Teams, Project Managers
- Description:
  - The system sends automated notifications and reminders to keep users informed about bug report statuses.
- Preconditions:
  - The user must be logged into the system and have notifications enabled.
- Steps:
  - A change occurs in a bug report (e.g., new assignment, status update).
  - The system identifies the relevant stakeholders based on the bug report details.
  - The system sends notifications via email or in-app alerts to the stakeholders.
  - Periodically, the system sends reminders to developers about pending bug reports that need attention.
- Postconditions:
  - Stakeholders are informed about changes and reminded of pending tasks.
- Exceptions:
  - If a user has disabled notifications, the system does not send alerts to that user.

These use cases provide a detailed view of how the Automated Bug Triaging System interacts with various users and external tools, ensuring a comprehensive understanding of the system's functionality and benefits.

Appendix C: User Guide

Welcome to the Automated Bug Triaging System User Guide. This guide provides detailed instructions on how to use the system effectively. Follow the steps below to make the most of the system's features.

C.1. Introduction

The Automated Bug Triaging System is designed to streamline the process of handling and prioritizing bug reports. By leveraging AI and machine learning, it ensures consistent categorization, prioritization, and assignment of bugs, improving software quality and developer productivity.

C.2. Getting Started

- System Requirements
  - A modern web browser (e.g., Chrome, Firefox, Safari)
  - Internet connection
  - User account credentials
- Logging In
  1. Open your web browser and navigate to the Automated Bug Triaging System login page.
  2. Enter your username and password.
  3. Click the "Login" button.
  4. If you do not have an account, click "Sign Up" to create one.

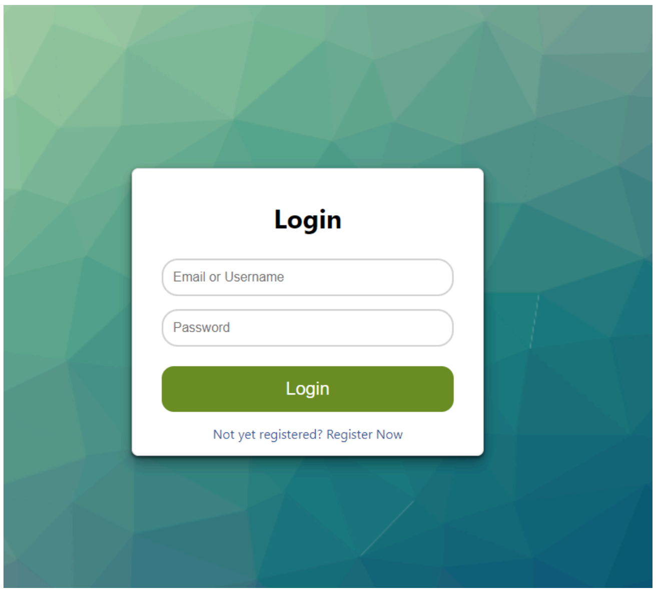


Figure 11: Login page

C.3. Creating a new project

- Accessing the projects form
    - After logging in, navigate to the "projects" section.
- Filling Out the Project Form
    - Description:
        - Provide the name of the project.
    - Submit:
        - Click the Ok" button to create the project.

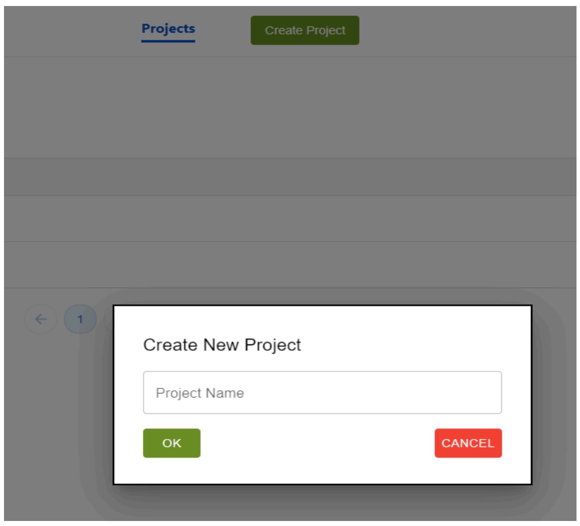
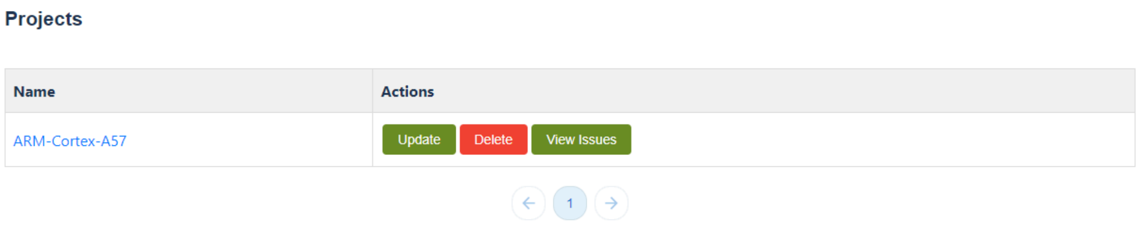


Figure 12: Create a new project



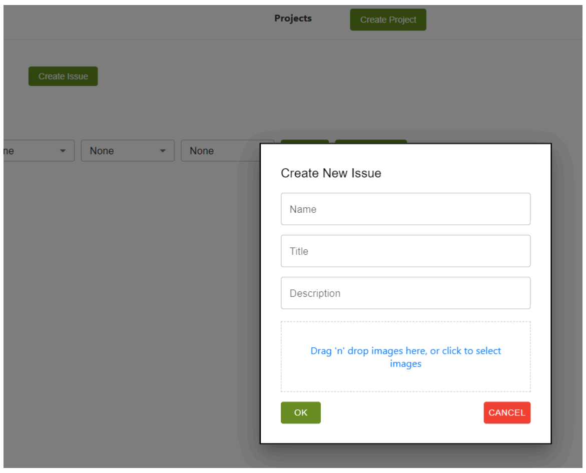
Projects

Name	Actions
ARM-Cortex-A57	Update Delete View Issues

Figure 13: The new project after creation

C.4. Submitting a Bug Report

- Accessing the Bug Report Form
  - After logging in, navigate to the "view issues" section from the main.
  - Click on "create issue"
- Filling Out the Bug Report Form
  - Title:
    - Provide the title of the issue
  - Description:
    - Provide a detailed description of the bug.
  - Attachments:
    - Attach any relevant screenshots or logs.
  - Submit:
    - Click the "Ok" button to send the report.
- Confirmation
  - After submission, you will receive a confirmation message, and the bug will be automatically categorized, prioritized, and assigned.



Create New Issue

Name

Title

Description

Drag 'n' drop images here, or click to select images

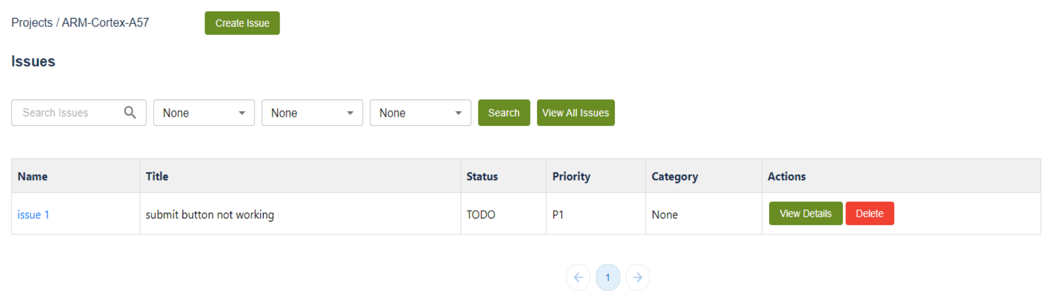
OK

CANCEL

Figure 14: Create new issue

C.5. Viewing and Managing Bug Reports

- Accessing Your Bug Reports
  - Navigate to the Your project section from the main dashboard.
- Filtering and Sorting
  - Use the filters to sort bug reports by status, priority, category, etc.
  - Click on a bug report to view detailed information.
- Updating Bug Reports
  - Update the status, add comments, or upload new attachments as needed.
  - Click "Save" to update the report.



Issues

Name	Title	Status	Priority	Category	Actions
issue 1	submit button not working	TODO	P1	None	View Details Delete

Figure 15: project issues and filters

C.6. Troubleshooting

- Common Issues
  - Login Problems: Ensure your username and password are correct. Reset your password if necessary.
  - Bug Report Submission Issues: Ensure all required fields are filled out. Check your internet connection.
- Getting Help
  - Visit the "Help" section from the main dashboard for FAQs and support articles.
  - Contact support via email or live chat for further assistance.

This user guide is intended to help you navigate and utilize the Automated Bug Triaging System effectively. If you have any questions or need further assistance, please do not hesitate to contact our support team.

Appendix E: Feasibility Study

This appendix provides a detailed analysis of the feasibility of the Automated Bug Triaging System project. The feasibility study includes technical, economic, operational, and schedule feasibility to ensure the project is viable and can be successfully implemented.

E.1 Technical Feasibility

E.1.1 Technologies Used

- **Backend Development:**
  - Node.js, Express.js
- **Frontend Development:**
  - React, HTML, CSS, JavaScript, Material-UI, Bootstrap 5
- **Database:**
  - MongoDB
- **Machine Learning & NLP:**
  - Python, Scikit-Learn, Pandas, NumPy, SciPy, NLTK

E.1.2 Technical Challenges and Solutions

1. **Data Quality and Availability:**
   - Challenge:
     i. High-quality and comprehensive datasets are crucial for training effective AI and ML models.
   - Solution:
     i. Implement robust data preprocessing steps to clean and normalize the data before training the models.
   - Impact:
     i. Enhances model accuracy and system reliability, improving user trust and satisfaction.
2. **Model Performance and Accuracy:**
   - Challenge:
     i. Achieving high accuracy in bug categorization, prioritization, and developer assignment models is challenging.
   - Solution:
     i. Perform extensive hyperparameter tuning, use cross-validation techniques, and implement continuous learning mechanisms.
   - Impact:
     i. Results in more reliable and effective bug triaging, enhancing user experience and efficiency.

E.1.3 Tools and Platforms

- **Version Control:**
  - Git, GitHub

Conclusion

The technical requirements for the project are feasible with the chosen technologies. The tools and platforms are capable of supporting the system's development and deployment.

E.2 Economic Feasibility

This section assesses the financial viability of the Automated Bug Triaging System project. It includes an analysis of initial development costs, ongoing operational expenses, and projected revenues over the first three years.

Expense Category	Details	Cost (USD)
Salaries and Wages	-	$76,800
Developers (5)	$600 per month each	$36,000
QA Engineers (2)	$600 per month each	$14,400
Project Manager (1)	$1,000 per month	$12,000
Data Scientists/ML Experts (2)	$600 per month each	$14,400
Software and Tools	-	$5,000
Development Tools and Licenses	-	$3,000
Cloud Hosting and Infrastructure	-	$2,000
Marketing and Sales	-	$15,400
Digital Marketing	-	$1,000
Sales Team (2)	$600 per month each	$14,400
Total Initial Budget (Year 1)	-	**$97,200**

Table 6: Initial Development Costs (Year 1)

Expense Category	Details	Cost (USD)
Salaries and Wages	-	$36,000
Additional Staff (5)	$600 per month each	$36,000
Software and Tools	-	$9,000
Maintenance and Upgrades	-	$4,000
Cloud Hosting and Infrastructure	-	$5,000
Marketing and Sales	-	$30,800
Expanded Digital Marketing	-	$2,000
Sales Team (4)	$600 per month each	$28,800
Total Ongoing Budget (Year 2)	-	**$75,800**

Table 7: Ongoing Costs (Year 2 and Beyond)

Year	Revenue from Companies	Revenue from QA Teams	Revenue from Freelancers	Revenue from Open Source Communities	Total Revenue
Year 1	$21,600	$7,200	$6,000	$24,000	$58,800
Year 2	$57,600	$21,600	$30,000	$72,000	$181,200
Year 3	$129,600	$57,600	$78,000	$120,000	$385,200

Table 8: Revenue Projections

Assumptions:

- Each company pays $60 per month (annual: $720).
- Each QA team pays $30 per month (annual: $360).
- Each freelancer pays $10 per month (annual: $120).
- Each open-source community pays $1,000 per month (annual: $12,000).

Economic Feasibility Conclusion

The economic feasibility analysis indicates that the project has the potential to be profitable within three years. With projected revenues surpassing ongoing costs by Year 2, the project is financially viable. The total initial budget of $97,200 can be covered by the revenues generated within the first two years, making the project a sound investment.

E.3 Operational Feasibility

Operational feasibility evaluates how well the proposed Automated Bug Triaging System can be integrated into the existing workflow of software development teams. It assesses the practicality of the implementation, the ease of use, the scalability, and the support and maintenance required to ensure smooth operation.

E.3.1 Integration into Existing Workflow

Current Workflow

- Manual triaging of bug reports.
- Manual categorization and prioritization.
- Developer assignment based on subjective judgment.
- Manual duplicate detection.

Proposed Workflow with Automated Bug Triaging System

- Automated categorization of bug reports using NLP and machine learning.
- Automated prioritization based on predefined criteria.
- Automated assignment of bugs to developers based on historical data and expertise.
- Automated duplicate detection using cosine similarity and TF-IDF.
- Real-time notifications and reminders to stakeholders.
- Detailed analytics and insights for continuous improvement.

Operational Impact

- **Reduced Manual Effort:**
  - Automation of repetitive tasks reduces the workload on development and QA teams.
- **Increased Efficiency:**
  - Faster triaging process allows teams to address critical bugs more quickly.

- **Consistency:**
  - Standardized processes ensure consistent categorization, prioritization, and assignment.
- **Improved Communication:**
  - Automated notifications keep stakeholders informed, reducing the likelihood of overlooked or forgotten bugs.

E.3.2 Ease of Use

User Interfaces

- **Frontend Interface:**
  - User-friendly dashboards for submitting bug reports, tracking progress, and viewing analytics.
- **Backend Management:**
  - Intuitive interfaces for managing settings, user roles, and system configurations.

Training Requirements

- Minimal training required due to intuitive design and clear documentation.
- User manuals and video tutorials to assist new users in getting started.

User Adoption

- Positive user feedback expected due to reduced workload and improved efficiency.
- User engagement driven by transparency and real-time updates on bug status.

E.3.3 Scalability

System Design

- **Modular Architecture:**
  - The system is designed with a modular approach, allowing for easy scaling of individual components.

Future Growth

- **Expandable Features:** New features and enhancements can be added without disrupting existing functionalities.

E.3.4 Support and Maintenance

Ongoing Support

- **24/7 Technical Support:**
    - Dedicated support team available to address any issues or queries.
- **Regular Updates:**
    - Continuous improvement through regular updates and patches to enhance performance and security.
- **User Feedback Loop:**
    - Mechanisms in place to gather user feedback and implement necessary changes promptly.

Maintenance Activities

- **Scheduled Maintenance:**
    - Regular maintenance windows for updates and optimizations.
- **Backup and Recovery:**
    - Robust backup and recovery procedures to ensure data integrity and minimize downtime.
- **Monitoring and Alerts:**
    - Continuous system monitoring with automated alerts for any anomalies or issues.

E.3.5 Risk Mitigation

Identified Risks

- **Data Quality Issues:**
    - Mitigated through robust preprocessing steps and continuous monitoring.
- **Model Performance Variability:**
    - Addressed by hyperparameter tuning, cross-validation, and continuous learning mechanisms.
- **User Resistance:**
    - Minimized by involving users in the design process and providing comprehensive training and support.

Impact on Project

- **Smooth Transition:**
  - Careful planning and execution ensure a smooth transition from manual to automated bug triaging.
- **Enhanced Productivity:**
  - Automation and improved processes lead to enhanced productivity and overall software quality.
- **Sustained User Satisfaction:**
  - Consistent performance and reliability foster user trust and satisfaction.

Conclusion

The operational feasibility analysis confirms that the Automated Bug Triaging System can be effectively integrated into the existing workflow of software development teams. The system's design ensures ease of use, scalability, and reliable support and maintenance. By automating critical aspects of bug triaging, the system significantly reduces manual effort, enhances efficiency, and ensures consistency, ultimately contributing to improved software quality and user satisfaction.

E.4 Operational Feasibility Schedule

The operational feasibility schedule outlines the timeline and key milestones for implementing the Automated Bug Triaging System. This schedule ensures that all critical tasks are planned, executed, and monitored systematically to achieve a smooth transition and integration into the existing workflow.

Phase 1: Planning and Preparation (Month 1-2)

1. **Requirement Analysis**
   - Document system requirements and objectives.
   - Identify key user roles and responsibilities.
2. **Project Planning**
   - Develop a detailed project plan.
   - Define milestones and deliverables.
   - Allocate resources and assign tasks.

3. **Infrastructure Setup**
    - Set up cloud infrastructure (e.g., AWS, Google Cloud, Azure).
    - Configure development and testing environments.
    - Ensure necessary software tools and licenses are in place.
4. **Data Collection and Preparation**
    - Collect sample bug reports for initial testing.
    - Prepare and preprocess data for model training.

Phase 2: System Development (Month 3-6)

1. **Module Development: Preprocessing and Feature Extraction**
    - Develop preprocessing module.
    - Implement feature extraction using TF-IDF vectorization.
    - Perform initial testing and validation.
2. **Module Development: Duplicate Detection**
    - Develop duplicate detection module.
    - Implement cosine similarity for duplicate detection.
    - Test and validate module performance.
3. **Module Development: Priority Prediction**
    - Develop priority prediction module using logistic regression.
    - Train and validate the model with sample data.
4. **Module Development: Automatic Categorization**
    - Implement SVM for bug categorization.
    - Perform model training and testing.
5. **Module Development: Clustering and Recommender System**
    - Develop clustering module for developer assignment.
    - Implement recommender system for developer recommendations.
    - Test and refine both modules.
6. **Website Development**
    - Design and develop the user interface using React, HTML, CSS, and JavaScript.
    - Integrate frontend with backend APIs.
    - Perform usability testing and refine the user interface.

Phase 3: Testing and Validation (Month 7-8)

1. **Unit Testing**
    - Perform unit testing for each module.
    - Ensure each module functions correctly in isolation.
2. **Integration Testing**
    - Test the integration of all modules.
    - Validate the end-to-end functionality of the system.
3. **End-to-End Testing**
    - Conduct comprehensive E2E testing on the website.
    - Simulate real-world scenarios to validate system performance.
4. **User Acceptance Testing**
    - Engage key stakeholders for user acceptance testing.
    - Gather feedback and make necessary adjustments.

Phase 4: Deployment (Month 9)

1. **Deployment Preparation**
    - Prepare deployment scripts and documentation.
    - Set up production environment.
2. **System Deployment**
    - Deploy the Automated Bug Triaging System to the production environment.
    - Conduct final validation to ensure successful deployment.
3. **Go-Live and Monitoring**
    - Officially launch the system.
    - Monitor system performance and address any issues promptly.

Summary

The operational feasibility schedule ensures a structured and systematic approach to implementing the Automated Bug Triaging System. By following this timeline, the project can achieve a smooth transition from planning and development to testing, deployment, and ongoing maintenance, ultimately ensuring the system's success and user satisfaction.

Appendix F: Meetings Logs

First Meeting

Date: 5/10/2023

Agenda:

- Presentation of the project idea

Summary:

The team successfully presented the concept of the Automated Bug Triaging System. This innovative project aims to improve the management and prioritization of bug reports in software development through AI and machine learning. Key features of the system include:

- AI and Machine Learning:
  - Analyzes bug reports for efficiency.
- Duplicate Detection:
  - Identifies and removes duplicate reports.
- Automated Assignment:
  - Assign reports to the appropriate developers.
- Prioritization:
  - Ranks bugs based on their severity.

This project was approved, and Dr. Yahya has given the team one month to complete the necessary research and studies.

Additionally, other ideas were presented but needed to be approved. The primary reason for the rejection was that these ideas needed to be more complex to qualify as a graduation project.

Second Meeting

Date: 3/11/2023
Agenda:

- Presentation of Available Datasets and Project Approach

Summary:

- Dataset Presentation:
  - The team provided an overview of the available datasets deemed suitable for the project. Each dataset's relevance to the bug assignment system was discussed in detail.
- Guidance from Dr. Yahya:
  - Dr. Yahya engaged in a detailed discussion on leveraging the available data to meet the project's goal. The primary objective is to efficiently assign bugs to the most suitable developers.
- Refined Project Approach:
  - Following Dr. Yahya's guidance, the team proposed a refined approach for bug assignment:
- For Users Without Bug History:
  - Extract keywords from existing bug data.
  - Implement a recommendation system based on user-selected keywords for bugs they can solve.
- For Users With Bug History:
  - Utilize historical bug-solving data.
  - Assign bugs based on the similarity between the new bug and the user's history of resolved bugs.

Third Meeting

Date:1/12/2023

Summary:

- ➢ Duplicates Feature:
  - Introduced the “SiameseQAT: A Semantic Context-Based Duplicate Bug Report Detection Using Replicated Cluster Information” paper and discussed its details with Dr. Yahya to start the block diagram and implementation process.
- ➢ Priority/Severity Feature:
  - Reviewed a dataset of approximately 1 million examples and initiated the prototype of the bug report prioritization.
- ➢ Bug Report Categorization:
  - Preprocessed a dataset of about 30k examples and planned to continue implementation the following week.
- ➢ Priority Assignment:
  - Outlined the steps for implementing the feature and began the process of the block diagram and implementation.
- ➢ Future Work:
  - Preparing the presentation for the seminar
  - Preparing a survey
  - Preparing the block diagram
  - Preparing the project timeline

Fourth Meeting

Date: 9/3/2024

Summary:

- ➢ Duplicates Feature:
    - ○ Decided to discard the “SiameseQAT: A Semantic Context-Based Duplicate Bug Report Detection Using Replicated Cluster Information” paper due to its reliance on a pre-tuned BERT model. Began searching for alternative methods.
- ➢ Developer Assignment:
    - ○ Continued discussions and refinements based on previous meetings.
- ➢ Bug Report Categorization:
    - ○ Collected a dataset of 35k rows and trained multiple machine learning models including logistic regression, LinearSVM, and Random Forest. After evaluation, LinearSVM emerged as the most accurate with an accuracy of 85%. However, the dataset size limits further improvement in accuracy. Dr. Yahya suggested exploring ensemble learning techniques such as AdaBoost to enhance performance.
- ➢ Priority Assignment:
    - ○ Continued development and refinement based on previous discussions and evaluations.
- ➢ Future Work:
    - ○ Search for another implementation for the duplicate detection feature
    - ○ Explore ensemble learning techniques such as AdaBoost

Fifth Meeting

Date: 7/4/2024
Summary:

- Duplicates Feature:
  - We discussed the results of using the TF-IDF algorithm and found it to be a useful starting point. We also agreed to begin testing the Glove algorithm for further improvements.
- Bug Report Categorization:
  - Explored techniques related to semi-supervised learning and initiated their implementation. The results will be reviewed in the next meeting.
- Sponsorship Discussion:
  - Discussed potential sponsorship with SIEMENS. Dr. Yahya agreed to some changes:
    - The system will be integrated with platforms like Jira.
    - Unsupervised or semi-supervised learning techniques may be used in bug categorization and duplicate detection instead of solely relying on supervised learning techniques.
- Future Work:
  - Start experimenting with the Glove algorithm.
  - Investigate additional semi-supervised learning techniques.

Sixth Meeting

Date: 22/6/2024
Summary:

- Duplicates Feature:
    - We discussed the results of TF-IDF and the final implementation of it and it is agreed from the doctor that the results are good and the implementation is also good as it gives the same results as the pre-implemented one from the sk-learn library.
- Bug Report Categorization:
    - We discussed the results of SVM and the final implementation of it and it was agreed from the doctor that the results are good and the implementation is also good as it gives the same results as the pre-implemented one.
- Bug Priority:
    - We discussed the results of Logistic regression and the final implementation of it and it was agreed from the doctor that the results are good and the implementation is also good as it gives the same results as the pre-implemented one.
- Developer Assignment:
    - We discussed the results we reached till now and still working on it.
- Future Work:
    - Finalize the Developer Assignment feature
    - Start the integration of the project
    - Start writing the GP Book