An Empirical Study of the Impact of Developer Proficiency on Bug fixing Efficiency and Accuracy

by

Khairatun Hissan(180042103)
Adiba Hasan(180042111)
Fatema-tuz-Zohora Sananda(180042124)

Supervised By:

Shohel Ahmed
Assistant Professor
Dept. of Computer Science and Engineering,
Islamic University of Technology

A thesis submitted to the Department of CSE in partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering

Department of Computer Science and Engineering
Islamic University of Technology (IUT)
Board Bazar, Gazipur-1704, Bangladesh.
May, 2023.

Declaration of Authorship

This is to certify that the work presented in this thesis, titled **"An Empirical Study of the Impact of Developer Proficiency on Bug Fixing Efficiency and Accuracy"**, is the outcome of research carried out by Khairatun Hissan, Adiba Hasan, Fatema-tuz-Zohora Sananda under the supervision of Shohel Ahmed. It is also declared that neither this thesis nor any part of it has been submitted anywhere else for the award of any degree, diploma, or other qualifications. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Khairatun Hissan

Student No.: 180042103,

Date: May 27, 2023.

Adiba Hasan

Student No.: 180042111,

Date: May 27, 2023.

Fatema-tuz-Zohora Sananda

Student No.: 180042124,

Date: May 27, 2023.

An Empirical Study of the Impact of Developer Proficiency on Bug Fixing Efficiency and Accuracy

Approved By:

Supervisor

Shohel Ahmed

Assistant Professor,

Department of Computer Science and Engineering,

Islamic University of Technology (IUT)

Date: May 27, 2023.

Acknowledgment

We would like to begin by expressing our sincere gratitude to Almighty Allah for his blessings, which enabled us to successfully complete this thesis research. It wouldn't be possible to be in our current situation without Allah's mercy.

We would like to thank Shohel Ahmed for his support, advice, and inspiration. The thesis would not be on the right track without his guidance and assistance. From the introduction of the thesis topics to implementation, his wise judgment, time, and input were all contributed throughout the thesis study, which enabled us to complete our thesis work correctly.

We want to express our gratitude to the jury members of my thesis committee for their insightful criticism and comments, which enabled us to further improve our work.
Finally, we want to express our sincere gratitude to our family and friends for their unwavering support.

Table of Contents

List of Figures

List of Tables

Abstract

In the modern software systems' evolution, solving bugs efficiently and reducing the life cycle of a bug has become increasingly essential. The developer's proficiency has a huge impact in this case. So our target is to study the effect of developer's proficiency on bug fixing efficiency and accuracy. We conducted an empirical study on a bug repository of an open-source project containing approximately 42574 issues. We proposed six factors, Total number of solved tasks, Mean time to solve a task, Task reopen ratio, Total number of fixed bugs, Mean time to fix a bug, and Bug reopen ratio for calculating developers' proficiency value. For validating our metric we also implemented Structural Equation Model (SEM) in our study. The analysis of your data revealed that the selected factors do indeed impact a developer's proficiency. Additionally, assigning bugs to proficient developers was found to reduce the bug life cycle. We also observed that, highly proficient developers may not always exhibit a high level of accuracy. Therefore, to effectively reduce the bug life cycle, it is crucial to focus on both the proficiency and accuracy levels of developers.

Chapter 1

Introduction

In this chapter, we present the overview of the whole research work. The outline includes a comprehensive motivation problem explanation as well as the objective of our study. Also it discussed the contribution made with our study. Thesis organization is noted at the last part of this chapter for readers easiness.

1.1 Overview

For businesses in the technology sector, determining proficiency of developers is a crucial duty. It is essential for assessing and comparing developers' technical abilities and performance, which facilitates wise recruiting, promotion, and project assignment decisions. Organizations must effectively analyze and comprehend the capabilities of their developers if they are to assure the efficacy of projects. This study intends to investigate the crucial elements of determining developer proficiency, concentrating on the assessment of technical abilities and performance metrics.

Technical skill and performance skill assessment is done by using our metrics to calculate number of bug fixed and task solved and their reopen ratio. Companies can find developers who have the skillsets required for particular positions and responsibilities by calculating proficiency of developers. They can use it to find knowledge gaps and design specialized training plans to improve developer abilities. Using the result companies can assign developers in the project in the field in which they have experience or expertise.

Organizations will be able to create high-performing teams, spend resources wisely, and guarantee the successful completion of software development projects with a greater understanding of developer proficiency. In the end, firms may boost their competitiveness and provide high-quality software solutions to match the constantly changing needs of the industry by enhancing developer performance and productivity.

1.2 Motivation

Project managers might allocate maintenance resources more effectively if they had a thorough grasp of how familiar developers and assignees were with issues. To our knowledge, no research has been done on the impact of developer familiarity on problem fixes. [2] [4]

Non-functional defects, such as performance and security bugs, are mentioned in much too broad terms [5] or within too narrow contexts (such as cloud computing systems or Android apps). [6] [7] This makes it even harder for researchers to identify the topic problems for their studies, forcing them to recommend doing time-consuming, expensive, and frequently fruitless searches in software repositories from beginning. [7]

It is observed that [2] -

1. One prevalent aspect in cases of bug fixing is familiarity: developers are more likely to be assigned to fix the flaws they themselves introduced;

2. The impacts of familiarity on bug fixing are complex. Although developers cure their own defects more quickly (and effectively), they are more likely to create new bugs when addressing existing ones (which results in less efficiency).

Using developer familiarity we want to propose a proficiency calculating metric which takes into account both bug fixing and task committing.

1.3 Objective

The ability of developers to solve problems and resolve bugs is essential for the production of high-quality software that is delivered on schedule. However, a thorough evaluation and comparison of the effects of technical abilities and performance indicators on developer proficiency in these fields is required. We discovered that bug fixing proficiency and task solving proficiency are not simultaneously taken into account when allocating developers to them after conducting an exhaustive study regarding various sorts of issues and how to detect and solve them. [8]

Existing research frequently lacks a comparative approach or concentrates on specific components of proficiency. Therefore, these current methodologies take into account either bug fixing or source commit actions, which may lead to suggestions from inactive or unskilled developers. Only taking into account one piece of information cannot make up for another, which reduces the accuracy of developer suggestions. By examining and assessing the relationship between performance measures and developer proficiency in bug fixing and task solving, we want to close this gap.

So from the above cases we designed our study to explore the impacts of developer proficiency on bug and task fixing time and reopen count by answering the following four research questions (RQs):

RQ1: What metrics effect the developer proficiency?

We studied from our dataset that developer proficiency depends on task and bug solving proficiency and their reopen count.

RQ2: Does a developer's proficiency affect bug-fixing efficiency and accuracy?

After calculating developer proficiency using the metric we checked each developer's proficiency in the dataset and then studied the relation between developer proficiency and bug-fixing efficiency and accuracy.

RQ3: Does a developer's bug fixing efficiency and accuracy increases over time?

In our study, after calculating developer proficiency using the metric, we checked each developer's proficiency over three time range to check whether with time, their efficiency and accuracy increases or not.

1.4 Contribution

For the research questions we collected the dataset of Apache project Spark. Then we proposed a metric to calculate task solving proficiency and bug fixing proficiency. Using these two we calculated total proficiency of the developers and their profieciency in a certain time range. From there we draw a conclusion that whether developer's proficiency has any impact on bug fixing accuracy and efficiency and also whether developer's proficiency increases over time. From there we can also state that bug fixing lifecycle will reduce if developers are assigned to bugs and task on that area in which they are proficient on.

1.5 Thesis Organization

In chapter 2, we included all the necessary background studies that would be used further in the paper.

In chapter 3, we presented all the related works that have influenced our study directly or indirectly.

In chapter 4, we described the whole methodology, starting from the dataset collection, data processing, and lastly the methodology of the research as a whole.

In chapter 5, we discussed the result generated in our methodology stage. The

elaboration of the result is also included in this chapter.

In chapter 6, the thesis work's future pursuit and our plan to follow it through. And with that, it concludes the thesis work.

Chapter 2

Background Study

Inside this chapter, we discuss about the background studies that are related to our research.

2.1 Bug

In the world of software development, a bug is a fault, flaw, or defect in a program that causes unexpected behavior or inaccurate output. Numerous factors might cause bugs to develop, including poor logic, poor data management, and incompatibility issues. Bugs can lead to unexpected behavior, program crashes, erroneous output, security issues, or inaccuracies. Finding and fixing bugs is a critical stage in the software development process to ensure the program functions as intended, meets the necessary requirements, and respects the accepted quality standards. We concentrated on both functional and non-functional bugs in our investigation.

2.2 Task

A task in software development is a particular action or piece of work that needs to be completed in order to accomplish a given project objective. To design and manage tasks, a project management system or task tracking application is typically utilized. They represent distinctive bits of work that are a component of the overall development process. It may be necessary to build code for specific features, fix issues, make documentation, conduct tests, or carry out any other task necessary for software development. Tasks are usually assigned to certain developers or teams and frequently have associated deadlines, priorities, and dependencies in order to facilitate effective collaboration and progress monitoring within the project. Effective task management results in timely completion and the achievement of project milestones.

2.3 Developer's Proficiency

Developer proficiency refers to a software developer's level of ability, knowledge, and expertise in their area of specialty. It includes a range of technical talents, problem-solving methods, domain expertise, and soft skills that give developers the ability to efficiently design, create, and manage software solutions.

Here, we combine task-solving ability with bug-fixing ability to consider developer proficiency.

2.4 Bug Fixing Proficiency of Developer

Bug fixing proficiency refers to a developer's ability to effectively and efficiently identify and resolve software bugs or defects. It encompasses their skills, knowledge, experience, and approach in addressing and resolving issues reported by users or discovered during the development process. A developer with strong bug fixing proficiency can contribute to improving software quality, user satisfaction, and the overall stability of the system.

Here are some factors that contribute to bug fixing proficiency:

- **Number of Bugs Fixed:** The number of bugs fixed refers to the quantity of software bugs or defects that a developer has successfully resolved within a given timeframe. It is a metric used to measure a developer's productivity and effectiveness in addressing and resolving reported issues.

- **Context and Nature of the Project:** Task solving proficiency can be influenced by the nature of the project, the team's workflow, and the specific development phase. For example, in the early stages of a project, more time may be dedicated to planning and design, resulting in a lower number of tasks completed. Similarly, if the project involves extensive research or debugging, fewer tasks may be completed compared to tasks focused on implementation.

- **Bug Severity and Complexity:** It is essential to consider the severity and complexity of the bugs when evaluating the number of bugs fixed. Some bugs may be relatively simple and straightforward to resolve, while others may be more complex and time-consuming. Focusing solely on the quantity of bugs without considering their severity or impact on the software's functionality may not provide an accurate assessment of a developer's performance.

- **Bug Classification and Categorization:** To ensure a fair and meaningful comparison, bugs should be classified and categorized consistently. Different types

of bugs may require varying levels of effort and expertise to fix. By categorizing bugs based on their nature or impact, it becomes possible to analyze the distribution of bug fixes and identify any patterns or areas that require additional attention.

- **Average Time to fix Bugs:** The average time to fix bugs is a metric that measures the average duration it takes for a developer or a development team to resolve reported software bugs or defects. It provides insights into the efficiency and effectiveness of the bug fixing process. The average time to fix bugs should be evaluated in the context of the severity and impact of the bugs being addressed. Critical or high-priority bugs may require immediate attention and faster resolution, while lower priority or cosmetic issues may have longer timeframes for resolution. The severity of bugs should be considered when assessing the average time to fix. [9]

- **Bug Reopen Count:** Bug reopen count refers to the number of times a bug or issue is reopened or reactivated after it has been marked as resolved or closed. It is a metric that tracks the frequency with which bugs resurface or require further attention after they were considered fixed. It is a indicator of bug quality. By reducing the bug reopen count, development teams can enhance software quality, improve customer satisfaction, and streamline their bug fixing processes. [3]

2.5 Task Solving Proficiency of Developer

Task solving proficiency refers to a developer's ability to effectively and efficiently solve tasks or problems encountered during software development. It encompasses their problem-solving skills, domain knowledge, technical expertise, and experience in applying appropriate strategies to overcome challenges and achieve task objectives. A developer's task solving proficiency can have a significant impact on the quality, timeliness, and success of software development.

Here are some factors that contribute to task solving proficiency:

- **Number of Tasks Solved/Committed:** The number of tasks solved or committed refers to the quantity of tasks that a developer has successfully completed or delivered within a given time-frame. The number of tasks solved by an individual developer is a significant factor behind the task solving proficiency of that developer. It indicates the developer's productivity and ability to meet task requirements and deadlines. The specific measurement of tasks solved or committed can vary depending on the development process or project management methodology being used.

- **Context and Nature of the Project:** Task solving proficiency can be influenced by the nature of the project, the team's workflow, and the specific development phase. For example, in the early stages of a project, more time may be dedicated to planning and design, resulting in a lower number of tasks completed. Similarly, if the project involves extensive research or debugging, fewer tasks may be completed compared to tasks focused on implementation.

- **Average Time to solve/commit Tasks:** The total time or average time to commit tasks by an individual developer has a huge imapact on his proficiency. The more proficient he is, the less time he should take to solve or commit a task. [10]

- **Task Reopen Count:** Task reopen count refers to the number of times a task or issue has been reopened or reactivated after it was considered resolved or closed. It is a metric used to track the frequency of tasks or issues that require additional attention or fixing even after they were initially addressed. It is a factor influencing the Task Solving Proficiency of the developer who resolved or closed the said task.

- **Task severity and Complexity:** It is essential to consider the severity and complexity of the tasks when evaluating the number of tasks solved or committed for understanding the proficiency. [10] Some tasks may be relatively small and straightforward, while others may be more complex and time-consuming. Focusing solely on the quantity of tasks without considering their complexity may not accurately reflect a developer's performance.

- **Task Definition:** Clearly defining what constitutes a "task" is crucial for accurate measurement. Tasks can vary depending on the project management methodology being used, such as user stories, tickets, or specific deliverables. Consistency in defining tasks ensures a fair comparison across developers and allows for meaningful analysis of productivity.

Chapter 3

Related Works

Inside this chapter, we discuss about the works that are related to our research.

3.1 Bug Fixing

Bug fixing has become an important topic of research now, there have been few research on this topic. The paper [11] is a Systematic Literature Review and Classification" by Shafiq, Hafiz, and Arshad (2014) presents a systematic literature review and classification of automated debugging and bug fixing solutions. The authors aim to provide an overview of the existing research in this field and classify the different techniques and approaches used in automated debugging and bug fixing.

The authors conducted a comprehensive review of the literature related to automated debugging and bug fixing, considering papers published up to the year 2014. They analyzed a total of 127 papers and identified various techniques and approaches used in automated debugging and bug fixing solutions.

The writers divided the selected solutions into various categories based on their analysis. They [11] classified the procedures according to their attributes, such as the kinds of bugs they deal with, the kinds of analyses done, the debugging techniques employed, and the degree of automation offered.

The study [11] emphasizes a number of significant discoveries. First of all, it demonstrates that a variety of methods and techniques are employed in automated debugging and issue solving. These include dynamic analysis, constraint solving, symbolic execution, program slicing, and many others.

Additionally, the authors discovered that various issue kinds, including syntax mistakes, logical errors, and runtime errors, call for various debugging approaches. The report also addresses the many debugging techniques used, including fault localisation, fault prediction, and fault correction.

In another study a formal method to bug fixing in software development is presented in the paper [1]. The authors suggest a way for systematizing the detection and

correction of flaws using formal verification techniques.

The authors begin by highlighting the value of bug repair in software development and the difficulties involved. They contend that conventional debugging methods frequently rely on trial and error, which makes the procedure laborious and prone to mistakes. They [1] provide a formal strategy that makes use of formal verification methods to overcome these problems.

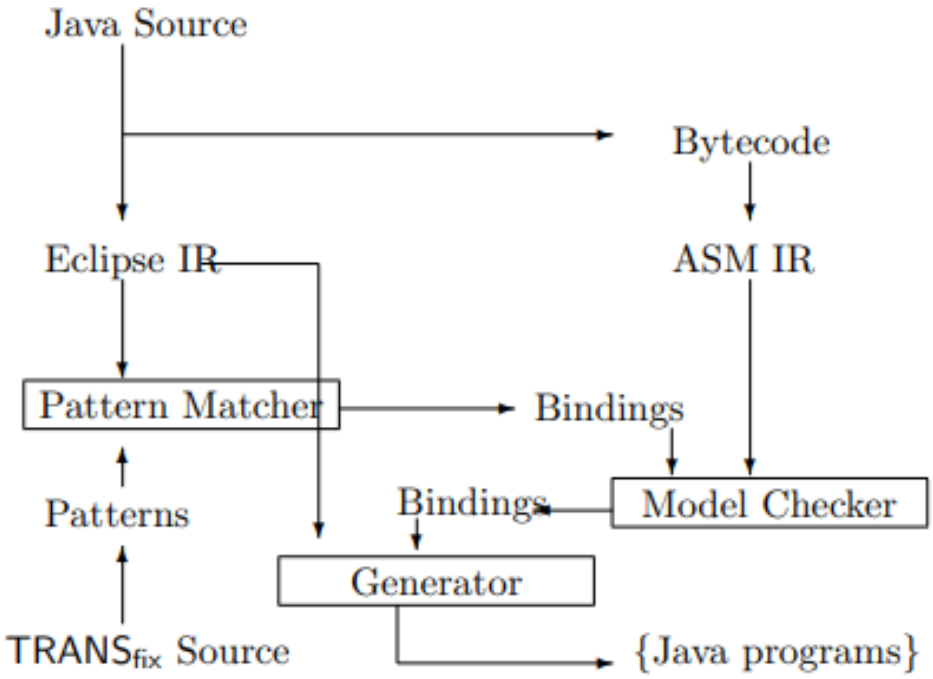

Figure 3.1: Bug Fixing Architecture of the Research [1]

The main procedures of their [1] formal bug-fixing methodology are described in the paper. First, a mathematical model is used to formally specify the software's flawed behavior. The issue in the model is then found and localized using formal verification techniques. The authors suggest a methodical procedure for finding the fault and then updating the model to correct it.

The authors give case studies on actual software systems to illustrate the viability of their [1] strategy. They emphasize how their approach effectively found and resolved defects in these systems as proof that their formal bug-fixing methodology is applicable in practice.

3.2 Developer's Proficiency

Developer familiarity refers to the level of knowledge and experience a developer has with a particular codebase or software project. It encompasses their understanding of the code structure, design patterns, dependencies, and overall system behavior.

Many studies ([12–14]) have shown that human factors have a significant influence on software quality. Some previous studies ([15, 16]) indicated that with the increase of team size, the communication and coordination in the team would be worse. It may slow down the development of software and cause more defects ([13, 17]). The similar result was also found in Windows 7 that when more developers worked on a binary, the failures of it would increase ([12, 14]).

In a recent paper [2], they investigated how developer familiarity with a codebase affects the bug fixing process in software development. The authors conducted an empirical study to analyze the impact of developer familiarity on bug fixing performance and outcomes. The study involved analyzing the bug fixing activities of 162 developers from a large software development organization. The developers worked on a total of 3,464 bug reports, allowing for a comprehensive examination of the effects of familiarity on bug fixing. They analyzed -

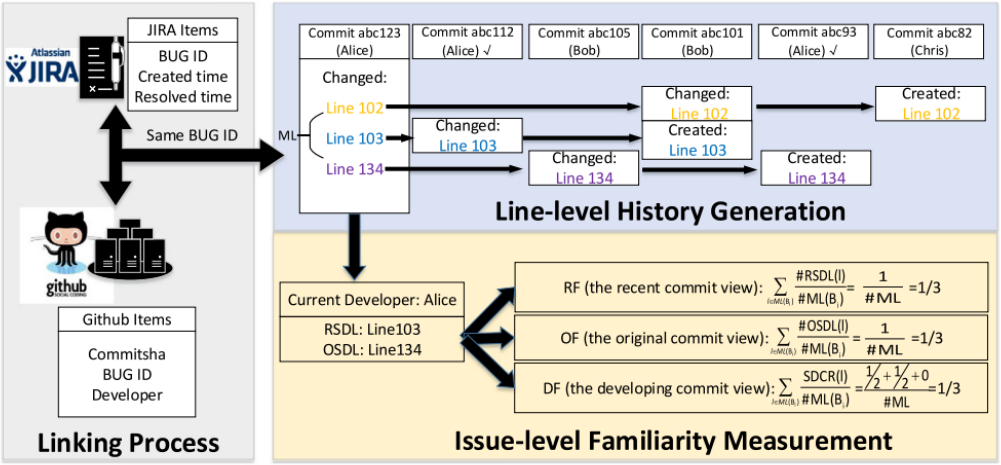

Figure 3.2: An Overview of the Three Steps in the mentioned Research Methodology [2]

- **Developer Familiarity Metrics:** The researchers used several metrics to measure developer familiarity, including the number of prior bug fixes in the same module, the number of prior changes to the same module, and the time since the last modification to the same module. These metrics were used to gauge different aspects of developer familiarity with the codebase.

- **Bug Fixing Performance:** The study evaluated bug fixing performance based on three aspects: bug resolution time, bug resolution rate, and bug fixing effort.

These metrics were used to assess the efficiency and effectiveness of developers in addressing reported bugs.

In recent studies numerous work have been done to relate developer skills with bug fixes. By assigning a bug to an expert developer of that particular categories will reduce the time to fix the bug and it's chance to reopen.

The research by Zaman et al. [18] worked on who fix a certain bug of Firefox projects and through this it gets easier to assign bug later. Their analysis found that developers with comparatively higher expertise were given security bug fixes. The research by Imseis et al. [19] worked on the same topic but on Chromium project bug repository. They discovered that more experienced developers were given security issues and experience is measured by the number of bugs a person has already fixed. Our study will verify that if we assign developers to bugs based on their skills then the span of the lifecycle of the bug fixation will minimise.

Some researchers also suggested ways to anticipate bug-fixing efforts. Panjer [20], for instance, suggested using machine-learning algorithms to estimate the time needed to resolve a defect. Using neural networks on a NASA dataset, Zeng and Rine [21] suggest predicting the bug-fix effort. Marks et al. [22] found that a Random Forest method can accurately forecast the class (low or high fix effort) of a bug repair with a success rate of 65%. Song [23] suggested association rule-based techniques for anticipating bug associations and bug-fixing effort. Weiß et al. [9] investigated the lifespan of bugs and suggested using the kNN approach to estimate the amount of effort needed to address bugs in the JBoss project.

Existing bug triage approaches for developer recommendation systems are mainly based on machine learning (ML) techniques. These approaches have shown low prediction accuracy and high bug tossing length (BTL) [3]. Another study [3] proposed a method for ranking software developers based on their expertise score to aid in bug triaging processes. Bug triaging involves assigning bugs to appropriate developers for resolution based on their expertise and availability.

The expertise score was computed using various parameters, including code churn, past bug fixing experience, and contribution to relevant code modules. They presented a bug triage model that utilizes the expertise scores of developers to determine the most appropriate developer for bug resolution. The model takes into account factors such as expertise score, workload, and bug priority to make informed bug assignment decisions. The proposed method was evaluated using a dataset obtained from an open-source bug tracking system. The evaluation aimed to assess the effectiveness of the expertise score-based ranking in accurately assigning bugs to developers with the appropriate expertise.

The results of the evaluation indicated that the proposed expertise score-based rank-

ing method improved the accuracy of bug assignment in bug triaging. The approach effectively identified developers with the relevant expertise, leading to more efficient bug resolution.

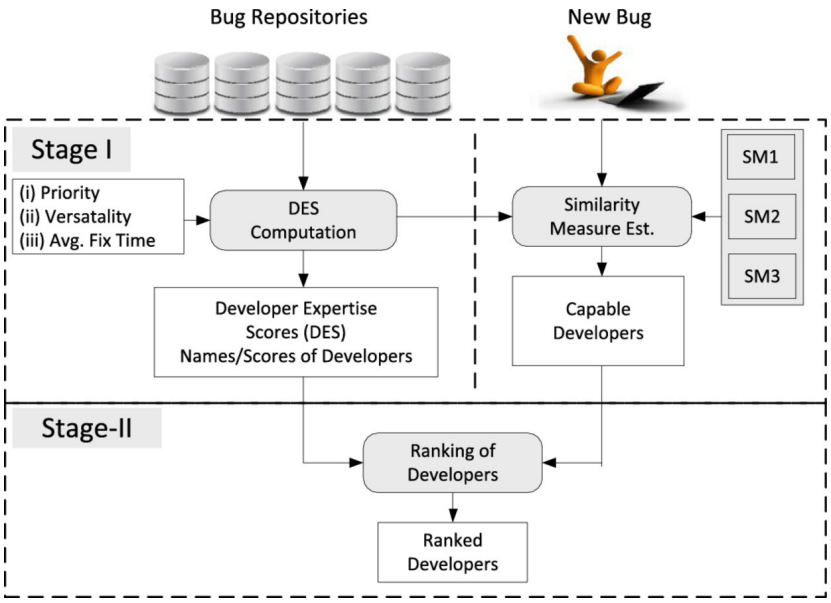

Figure 3.3: Methodology of Yadav *et al.* [3]

Understanding the work practices of software developers by exploring the factors that influence developers' task performance and engagement is essential for improving productivity and software development outcomes. A study [10] investigates developers' participation and engagement in different tasks. It explores the extent to which developers actively engage in tasks, collaborate with others, and contribute their expertise and knowledge. The speed at which developers resolve tasks identifies as a factor that contribute to faster task resolution. It also examines how task characteristics, such as complexity and novelty, influence the time it takes for developers to complete tasks.

The results of the study [10] indicate that task complexity, task novelty, and task interdependence significantly impact developers' work practices. Developers tend to have higher participation and engagement in tasks that are less complex and less novel. The speed of task resolution is influenced by task complexity, with more complex tasks taking longer to complete.

Chapter 4

Dataset Generation and Methodology

The whole dataset pre-processing and methodology will be discussed in this chapter. The dataset preprocessing starts with the extraction of the bug repository from Jira and the methodology describes all the data analysis processes considered for answering the research questions.

4.1 Dataset Description

In this section, the dataset description which is collected from the Jira bug repository will be discussed. Also, this section discusses the effective factors that we have considered to calculate the impact of a developer's proficiency on bug-fixing accuracy and efficiency.

In this study, we collected bug reports of Spark from Apache Software Foundation (ASF) that are all managed in JIRA ITS. The Apache Spark project has well-structured issue-tracking system in JIRA and a version control system in Github. The project has a total of 42574 issues within the starting date 12/12/2012 and ending date 1/17/2023 in JIRA. Among these, we considered only the issues those were fixed. The number of fixed issues is 25156. In this dataset, there were around 1032 unique developers who worked on 37 unique components. We filtered the Bug issues and Task issues from the dataset. There are around 8936 fixed Bug Issues and 16220 fixed task Issues in the dataset. We have considered all the tasks, subtasks, improvements, new features as task issues in our analysis. We followed the Interquartile range or (IQR) method for detecting and removing the outliers from our dataset.

4.2 Data Processing

In the empirical study on the impact of developer proficiency on bug-fixing efficiency and accuracy, the collected dataset will undergo several data processing steps to pre-

Table 4.1: Dataset Description

Number of total issues	42574
Number of issues with fixed status	25156
Number of fixed bug issues	8936
Number of fixed task issues	16220
Number of developers	1032
Number of components	37

pare it for analysis. The following steps will be undertaken:

4.2.1 Removing Outlier:

The dataset undergoes pre-processing to transform and clean the data as needed. This may involve removing irrelevant columns, handling missing values, and addressing any data inconsistencies or anomalies. For removing the outliers we have used IQR method. The IQR method is briefly described here in this section

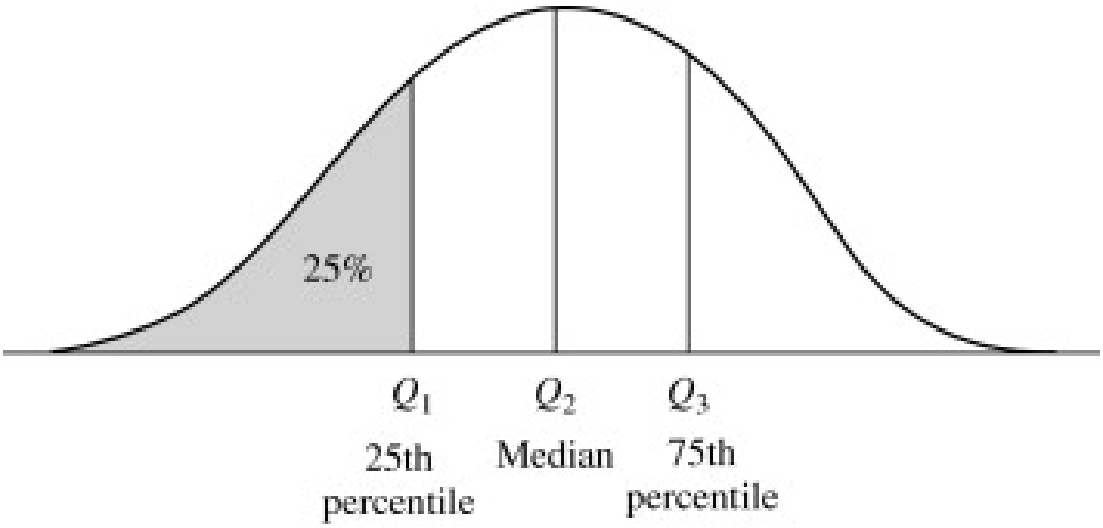

Figure 4.1: IQR Method

Interquartile Range Method (IQR)

[24]The interquartile range (IQR) method is commonly used to identify and remove outliers from a dataset. Outliers are data points that significantly deviate from the rest of the data and can skew statistical analyses and modeling results.

Here's how the IQR method works:

1. Calculate the IQR, which is the range between the first quartile (Q1) and the third quartile (Q3). The quartiles divide the dataset into four equal parts, with Q1 representing the 25th percentile and Q3 representing the 75th percentile.

2. Determine the lower bound and upper bound for outliers. The lower bound is calculated by subtracting 1.5 times the IQR from Q1, and the upper bound is calculated by adding 1.5 times the IQR to Q3.

3. Identify any data points that fall below the lower bound or above the upper bound. These points are considered outliers.

4. Decide how to handle the outliers. You can either remove them from the dataset or apply a specific treatment, such as replacing them with a different value or imputing missing data.

By removing outliers, the IQR method helps to ensure that statistical analyses and models are more robust and reflective of the general patterns and trends in the data. However, it's important to note that outlier removal is a subjective decision and should be based on the specific context and goals of the analysis.

After removing the outliers the status of the dataset is shown on the table below.

Table 4.2: Dataset Description After Removing Outliers

Data	Before	After
Number of total issues	42574	30303
Number of issues with fixed status	25156	20159
Number of fixed bug issues	8936	5756
Number of fixed task issues	16220	14080

4.2.2 Filtering Bugs and Tasks

The bug data is filtered to focus specifically on the bug-related entries, removing any unrelated or irrelevant information. Similarly, the task data is filtered to isolate the task-related entries, discarding any non-relevant information.

4.2.3 Mean Time Calculation for bug-fixing and task solving

The filtered task data is used to calculate the mean time taken to solve tasks by individual developers. This calculation provides insights into the efficiency of developers

in resolving tasks. In the same way, the filtered bug data is used to calculate the mean time taken to fix bugs by individual developers. This calculation provides insights into the accuracy and efficiency of developers in addressing bugs.

The mean time to solve a task and mean time to fix a bug is computed for individual developers to compare their efficiency levels. the mean time is also used to calculate their proficiency.

The filtered bug dataset and the task dataset are divided into three spans of time periods which are 2013-2015, 2016-2018, and 2019-2023. Using this time-wise filtered dataset, the developer's efficiency increasing over time is checked.

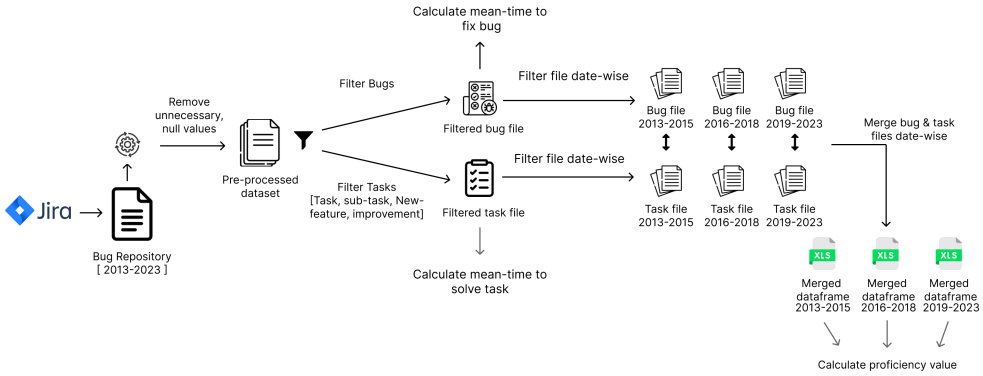

Figure 4.2: Data Processing

4.2.4 Dataset Quality

Understanding the goals and objectives of your dataset is crucial for assessing its quality and determining the appropriate analysis techniques. Since the dataset is collected from an open-source project and contains information about issues within a specific date range, we can derive several metrics to evaluate the developers' proficiency. For example, The total number of bugs provides an overall count of the bugs recorded within the dataset. It helps in understanding the magnitude of bug-related issues.

The total number of tasks solved calculates the total count of tasks that developers have successfully completed. It gives an indication of their productivity and ability

to accomplish assigned tasks. We also considered the severity of the task that the developers solved.

Mean time to fix a bug calculates the average time taken to resolve a bug. It helps in assessing the efficiency and speed of developers in addressing and fixing bugs.

Reopen bug and task ratio measures the frequency of bugs or tasks being reopened after they were initially considered resolved. It can highlight potential issues with the quality of work or the understanding of requirements.

By analyzing these metrics, you can gain insights into the proficiency and performance of developers in the open-source project.

4.3 Methodology

This section outlines the methodology employed in conducting an empirical study on the impact of developer proficiency on bug fixing efficiency and accuracy. The study aims to investigate how the proficiency of developers influences their ability to efficiently and accurately fix software bugs. The methodology encompasses the data analysis, calculating developer's proficiency, accuracy and familiarity check.

4.3.1 Calculation of Developer Proficiency

In the empirical study on the impact of developer proficiency on bug fixing efficiency and accuracy, developer proficiency will be calculated based on task proficiency and bug proficiency metrics.

$$DP_i = TSP_i + BFP_i \tag{4.1}$$

Here, i = Specific Developer
DP = Developer Proficiency
TS = Task Solving Proficiency
BF = Bug Fixing Proficiency

The following steps will be undertaken to calculate developer proficiency:

4.3.2 Calculating Task Solving Proficiency

Task proficiency will be calculated for each developer using the following metrics:

- **Number of Tasks Solved (TS)**
  In our study, the dataset is analyzed to determine the total number of tasks solved by each developer. These tasks were categorized based on their severity level, and we computed the total number of tasks in each category. However, we discarded the category of "Trivial tasks" as they do not have any impact on the

result of our analysis. To assign weights to each category of tasks, we assumed a weighting factor denoted by "W" in the formula mentioned below. The weighting factor ranged from 2 to 5, with the most severe category, "Blocker tasks," receiving the highest weighting value of 5, and the least severe category, "Minor tasks," receiving the lowest weighting value of 2.

By multiplying the total number of tasks in each category by their corresponding weighting factors, it is possible to incorporate the severity level into our calculation of a developer's proficiency. This approach allows us to assign higher importance to more critical tasks while evaluating a developer's performance. This metric reflects the developer's ability to successfully address and complete tasks.

$$TS_i = \sum BlockerTask_i \times W4 + \sum CriticalTask_i \times W3 + \sum MajorTask_i \times W2 + \sum MinorTask_i \times W1$$

Here, W1 = 2, W2 = 3, W3 = 4, W4 = 5

- **Task Reopen Ratio (TRR)**
  In our study, we examined the dataset to identify the ratio of the number of times tasks solved by each developer were reopened to the total number of tasks they solved. We calculated this reopen ratio for each type of task, considering the four categories you previously defined.

  To incorporate the impact of task reopen ratios into the overall measure of a developer's proficiency, we have multiplied each reopen ratio by its corresponding weighting factor. Each category of tasks was assigned a specific weighting factor to reflect its severity or importance in the analysis.

  By summing up the weighted reopen ratios for all task categories, we obtained the total reopen ratio of a particular developer. This approach allows us to capture the tendency of a developer's tasks to be reopened, providing insights into their accuracy and the quality of their initial fixes.

  By considering the reopen ratios and applying appropriate weighting factors, we can effectively incorporate the accuracy aspect into the evaluation of a developer's proficiency.

$$TRR_i = \frac{\sum TasksReopened_i}{TotalTaskSolved_i} \tag{4.2}$$

- **Mean Time to Solve Tasks (MTST)**
  In our study, we have utilized the dataset to calculate the average time taken

by each developer to solve tasks. To determine the time required to complete a specific task, we subtracted the issue resolved time from the issue created time. This calculation provides the duration between the creation and resolution of an issue.

The average time to solve tasks for each developer reflects their efficiency and speed in resolving issues. This metric helps evaluating how quickly developers can address and resolve tasks. It allows for comparisons between different developers in terms of their efficiency in task solving and highlights potential areas for improvement.

By considering the average time to solve tasks, we can assess the speed and effectiveness of developers, which is crucial for reducing the bug life cycle and improving overall software system quality.

$$MTST_i = \overline{\sum_{i=1}^{n} Resolvedtime_i - Createdtime_i}$$

- **Task Solving Proficiency**

  In our study, we calculated the task solving proficiency using three factors: TS, TRR, MTST. To determine the overall impact of these factors on a developer's proficiency, we assigned impact factors to each of them.

  To establish the impact factors, we conducted a survey among industry professionals. This survey aimed to gather insights and opinions from experienced individuals in the field regarding the relative importance and impact of the TS, TRR, and MTST factors.

  By gathering the responses from the survey participants, we were able to determine the impact factors for TS, TRR, and MTST. These impact factors represent the weighting or importance given to each factor in calculating a developer's task solving proficiency. This approach also enhances the validity and relevance of our findings and provides a more comprehensive evaluation of a developer's proficiency in task solving.

  In the subsequent section,we will discuss in details about the survey, such as its methodology, participant demographics, and the final results observed from the survey responses.

$$TSP_i = \frac{(TS_i \times I1) - (TRR_i \times I2)}{MTST_i \times I3} \tag{4.3}$$

4.3.3 Calculating Bug Fixing Proficiency

Bug proficiency will be calculated for each developer using the following metrics:

- **Number of Bugs Fixed (BF)**

  In our study, we analyzed the filtered bug dataset to determine the total number of bugs solved by each developer. Similar to the tasks, these bugs were also categorized based on their severity level. However, we chose to exclude the category of "Trivial bugs" as they have no impact on the overall analysis.

  To incorporate the severity level into the calculation of a developer's proficiency, we introduced a weighting factor denoted by "W" in the formula. The weighting factor ranged from 2 to 5, with "Blocker bugs" assigned the highest weighting value of 5 and "Minor bugs" assigned the lowest weighting value of 2.

  By multiplying the total number of bugs in each category by their corresponding weighting factors, we effectively assigned higher importance to more severe bugs when evaluating a developer's performance. This approach recognizes that resolving critical bugs carries more significance in the bug fixing process.

  This also enables us to assess their ability to address and fix more severe issues, which can significantly impact the overall quality and stability of software systems.

$$BF_i = \sum BlockerBug_i \times W4 + \sum CriticalBug_i \times W3 + \sum MajorBug \times W2 + \sum MinorBug \times W1$$

Here, W1 = 2, W2 = 3, W3 = 4, W4 = 5

- **Bug Reopen Ratio (BRR)**

  In our study, we examined the dataset to determine the ratio of the number of times bugs solved by each developer were reopened to the total number of bugs they solved. This reopen ratio was calculated for each type of bug, taking into account the previously defined four categories.

  To incorporate the impact of bug reopen ratios into the overall measure of a developer's proficiency, we multiplied each reopen ratio by its corresponding weighting factor. Each category of bugs was assigned a specific weighting factor that reflects its severity or importance in the analysis.

  By summing up the weighted reopen ratios for all bug categories, we obtained the total reopen ratio of a particular developer. This metric allows us to assess the tendency of a developer's bugs to be reopened, providing insights into the accuracy and quality of their initial bug fixes.

  By evaluating both the proficiency and accuracy levels of developers, we can gain a better understanding of their overall bug fixing performance and identify areas for improvement in order to reduce the bug life cycle efficiently and accurately.

$$BRR_i = \frac{\sum BugsReopened_i}{TotalBugFixed_i} \tag{4.4}$$

- **Mean Time to Fix Bugs (MTFB)**

  By calculating the average time taken by each developer to solve bugs, we have obtained a metric that reflects their efficiency and speed in resolving issues. This metric can be valuable for evaluating and comparing the performance of different developers in terms of bug fixing.

  Analyzing the average time to solve bugs allows us to identify potential areas for improvement. This information can be used to allocate resources, provide targeted training or support, and optimize bug resolution processes.

  Assessing the speed and effectiveness of developers in bug fixing is essential for reducing the bug life cycle. By identifying areas where developers may be experiencing challenges or delays, we can implement strategies to address those issues promptly. It's important to note that while the average time to solve bugs provides useful insights, it should be considered alongside other relevant metrics and factors. Bug complexity, severity, and the specific circumstances surrounding each issue should also be taken into account for a comprehensive evaluation of developers' bug fixing capabilities.

$$MTFB_i = \overline{\sum_{i=1}^{n} Resolvedtime_i - Createdtime_i}$$

- **Bug Fixing Proficiency (BFP)**

  We calculated the bug fixing proficiency for each developer using the factors BF, BRR, and MTFB, along with their corresponding impact factors obtained from the survey. By multiplying each factor by its respective impact factor, we are weighting their contributions to the overall bug fixing proficiency score.

  This methodology allows us to incorporate the insights gathered from industry professionals and give more importance to factors that were deemed more influential in determining bug fixing proficiency. By considering the impact factors, we are providing a more nuanced evaluation that reflects the relative significance of each factor.

  In the next section, the survey for calculating the impact factors will be described.

$$BFP_i = \frac{(BF_i \times I4) - (BRR_i \times I5}{MTFB_i \times I6} \tag{4.5}$$

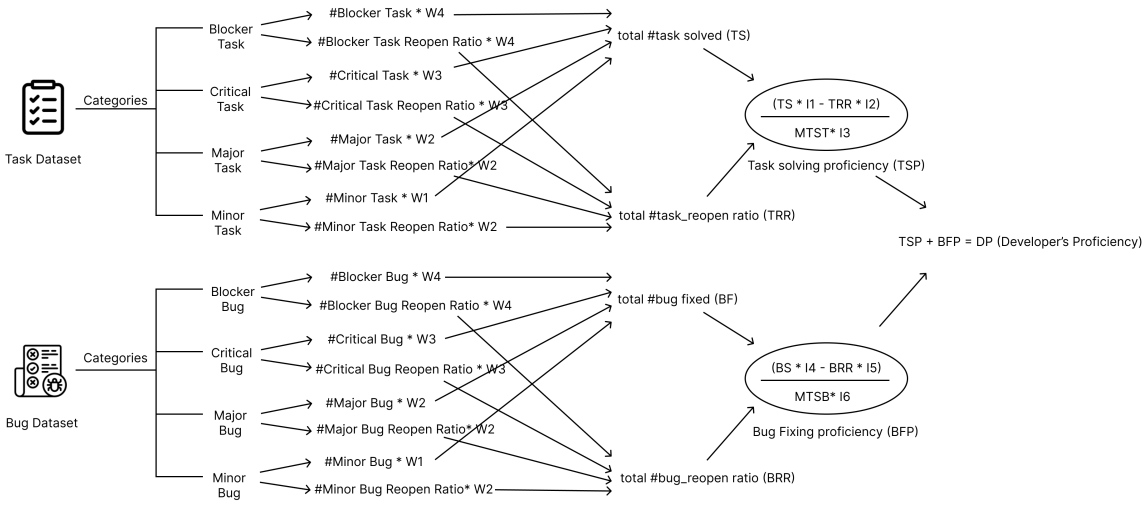

Figure 4.3: Calculation of Developer's Proficiency

Now in the next section we are going to discuss about the survey with professionals in the field. The survey aims to gather expert opinions and insights on the importance and relevance of the factors considered in the proficiency calculations.

4.3.4 Survey

We conducted survey to compute the impact facotrs for our metric. By approaching 30 industry professionals through LinkedIn and Facebook and receiving 15 responses, we have obtained valuable insights from a diverse set of participants. Considering professionals with at least 2 years of experience, and with most respondents having over 3 years of experience, ensures that the opinions gathered are from individuals with relevant expertise in the field of software development.

The fact that we received responses from professionals working in 11 different renowned software companies adds to the diversity of perspectives and further enhances the credibility of our findings. This suggests that the impact factors identified in our survey may have broader applicability and relevance within the industry.

We have used likert scales with a range from 1 to 5 which allows respondents to express their opinions on the impact of each factor in a graded manner. This enables us to capture the relative importance assigned to each factor by the participants, and subsequently calculate the impact factors for our proficiency metric.

Table 4.3: Description of the Survey Respondents

Respondent	Company Name	Designation	Years of Experience
1	DreamOnline Ltd.	Sr. Manager	10 years+
2	Samsung RnD Bangladesh	Chief Engineer	5-7 years
3	CHEQ Inc.	Sr. Technical Project Manager	10 years+
4	Renaissance Group	System Administrator	10 years+
5	Samsung	Software Developer	2-3 years+
6	Microsoft Corporation	Sr. Software Engineer	10 years+
7	MathWorks	Software Engineer	5-7 years
8	Amazon Web Services	Sr. Software Development Engineer	5-7 years
9	Kaz software	Sr. Software Engineer	5-7 years
10	Kona Software Lab Ltd.	Lead Software Engineer	10 years+

From the result of the survey we got the following values for our selected factors. So the values of the impact factors are mentioned in the table

Table 4.4: Impact Factor Values

Factors	Denoted as	Impact Value
Number of tasks solved by a developer	*I1*	3
Mean time to solve a task	*I2*	5
Task reopen ratio	*I3*	5
Number of bugs solved by a developer	*I4*	2
Mean time to solve a bug	*I5*	3
Bug Reopen Ration	*I6*	5

The professionals' perspectives and feedback are considered in determining the relative importance of each factor. The weighting factors are adjusted or refined based on the insights gained from the survey.

Chapter 5

Results and Discussion

In the empirical study on the impact of developer proficiency on bug fixing efficiency and accuracy, the analysis focused on identifying the metrics that significantly influence developer proficiency. There are three questions of our study and in our study we focused on these three.

5.1 Metrics for Developer Proficiency

Developer proficiency can be viewed to influenced by Bug Fixing Proficiency and Task Solving Proficiency. For calculating these factors, we propose a metrics. Using this we calculate developer proficiency.

5.1.1 Task Solving Proficiency

- **Number of Tasks Solved:** The analysis revealed that a higher number of tasks solved by developers positively correlated with their proficiency. Developers who consistently solved a larger number of tasks demonstrated higher proficiency levels.
- **Reopen Ratio of Tasks:** It was observed that a lower reopen count of tasks indicated better proficiency. Developers who had fewer instances of tasks being reopened after initial fixes demonstrated higher proficiency.
- **Mean Time to Solve Tasks:** The analysis indicated that a shorter mean time to solve tasks correlated with higher proficiency levels. Developers who efficiently resolved tasks in less time exhibited better proficiency.

5.1.2 Bug Fixing Proficiency

- **Number of Bugs Fixed:** The analysis found that developers who fixed a greater number of bugs exhibited higher proficiency levels. A higher bug-fixing capability reflected improved proficiency in addressing and resolving bugs.
- **Reopen Ratio of Bugs:** It was observed that a lower reopen count of bugs indicated better proficiency. Developers who had fewer instances of bugs being reopened after initial fixes demonstrated higher proficiency.
- **Mean Time to Fix Bugs:** The analysis showed that a shorter mean time to fix bugs correlated with higher proficiency levels. Developers who efficiently resolved bugs in less time showcased better proficiency.

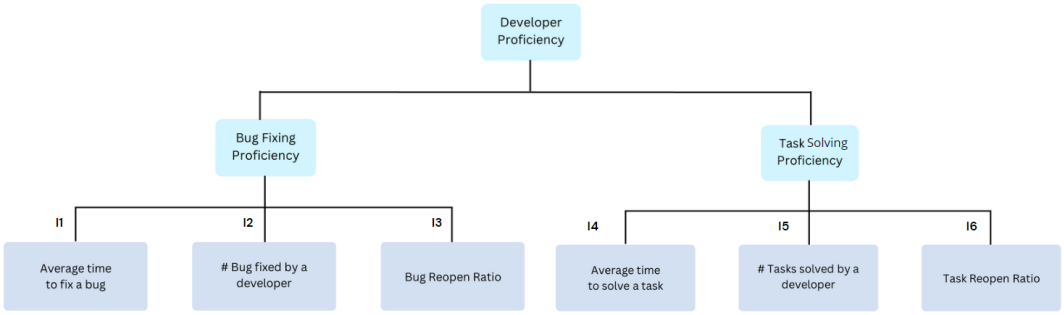

Figure 5.1: Factors affecting Developer Proficiency

Considering these factors we calculate the Developer Proficiency (DP). We use the following equation -

$$DP_i = TSP_i + BFP_i \tag{5.1}$$

$$DP_i = \frac{(TS_i \times I1) - (TRR_i \times I2)}{MTST_i \times I3} + \frac{(BF_i \times I4) - (BRR_i \times I5}{MTFB_i \times I6} \tag{5.2}$$

Table 5.1 shows some calculated values of some randomly chosen developers. This calculation has been done by using our formula from equations 5.1 and 5.2. We considered both bug fixing and task solving proficiencies.

In our study, we get the value of developer proficiency from range 10 - 120 approximately. We labelled the developers as highly proficient if they have proficiency range between 60-120, medium proficient if the proficiency range is between 30-60 and if the range is between 10-30 then low proficient.

Table 5.1: Developer Proficiency Calculated for some developers

Developer	TS	MTST	TRR	BF	MBFT	BRR	Proficiency
viirya	217	22.126	11	110	24.327	11	23.6846
cloud_fan	419	16.668	9	120	31.91667	6	50.5691
sarutak	135	15.673	4	94	17.553	5	21.2654
yumwang	216	22.221	10	36	20.5556	3	19.5896
XinrongM	112	11.25056424	3	2	3	0	18.9324
davies	201	25.8099	12	89	40.382	9	18.7801

Table 5.2: Developers and number of component/s they have worked in

Developer	No. of Components worked on
zsxwing	16
mengxr	14
holdenkarau	4
joshrosen	19
srowen	21
marmbrus	7
codingcat	12

Table 5.2 shows the names of some developers and the number of component/s he has worked on. Working in a number of components gives the developer more knowledge about the project. His bug fixes are less likely to trigger additional bugs in other components.

Figure 5.2 shows that with the increase in proficiency, the mean time to fix a bug decreases. This matches with our calculation.

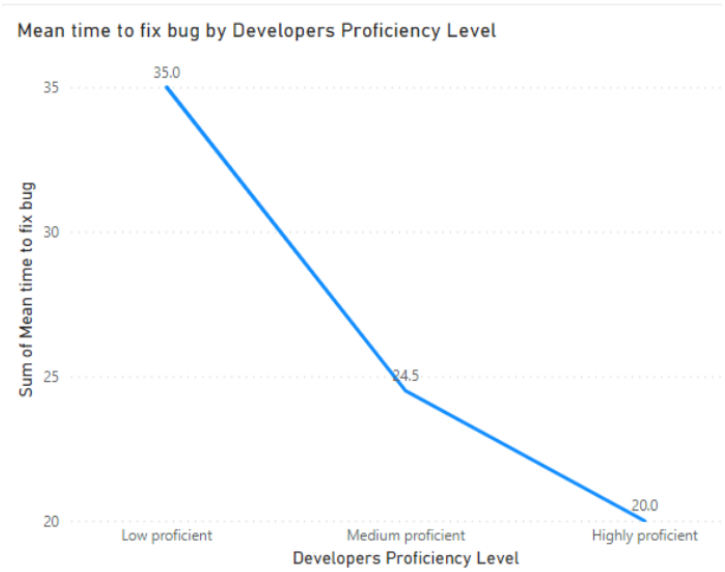

Figure 5.2: Proficiency Level Vs Mean time to solve bug

To validate our formula and the assumptions of the weight of the impact factors that we got from the survey, we use SEM (Structural Equation Modelling) to cross check the level of impact factors of the factors affecting developer proficiency.

Structural Equation Modelling (SEM)

Structural Equation Models explain relationships between measured and latent variables and relationships between latent variables. Latent variables: which can't be measured directly. For example, "Intelligence" is a latent variable that can be measured by measured variables such as exam scores, IQ scores, psych test scores, etc. SEM is used to convert the measured variables into latent variable.

Why SEM?

- It will lets us analyze the influence of predictor variables on numerous dependent variables simultaneously.
- It will allow us to account for measurement error and even addresses error in predicting relationships
- It is capable of testing an entire model instead of just focusing on individual relationships. This is in direct contrast to similar techniques like regression that can test only one dependent variable at a time, does not account for measurement error, and focuses on singular relationships instead of the collective whole.

Confirmatory Factor Analysis

We perform Confirmatory Factor Analysis for the factor to assess the unidimensionality of the measurement items. The results of the confirmatory factor analysis indicate that eacg item loaded on its respective underlying concept and all loadings were significant.

Construct reliabilities were also assessed for every construct. The model fit indices indicate that the measurement model was a good fit to the data.

Some indicators of fitness of the model-

- **CMIN/df**
  CMIN/df refers to the ratio of the chi-square statistic (CMIN) to the degrees of freedom (df). The chi-square statistic is a measure of the discrepancy between the observed data and the model-implied covariance matrix. The degrees of freedom represent the number of independent pieces of information available for model estimation.

- **Degree of freedom**
  The degrees of freedom represent the number of independent pieces of information available for model estimation. The degrees of freedom are often used in the calculation of fit indices, such as the chi-square statistic (CMIN) and its ratio to degrees of freedom (CMIN/df).

- **CFI**
  CFI is a relative fit index that compares the fit of the proposed model to a baseline model. The CFI ranges from 0 to 1, with values closer to 1 indicating better fit. A CFI value of 1 represents a perfect fit, while values above 0.90 are generally considered as indicative of a good fit. However, the exact threshold for an acceptable fit can vary depending on the specific research context and the complexity of the model being tested.

- **RMSEA**
  The RMSEA is a population-based fit index that measures the average discrepancy between the implied model and the observed data, adjusted for model complexity and the number of degrees of freedom. It is typically interpreted as a measure of lack of fit or error in the model. The RMSEA ranges from 0 to infinity, with lower values indicating better fit.

Table 5.3 and Table 5.4 shows the measurement indices for both Task Solving and Bug Fixing Proficiency Models. They indicate a good model fit.

Table 5.3: Measurement indices for Bug Fixing Proficiency Model

Indices	Values
CMIN/df	0.09
CFI	0.963
RMSEA	0.132

Table 5.4: Measurement indices for Task Solving Proficiency Model

Indices	Values
CMIN/df	0.072
CFI	0.9832
RMSEA	0.256

Structural Model Analysis

The hypothesis here in this model is that number of bug count and mean bug fixing time have a positive effect on Bug Fixing Proficiency, while number of may bug reopen ratio has a weaker or indirect effect on Bug Fixing Proficiency.

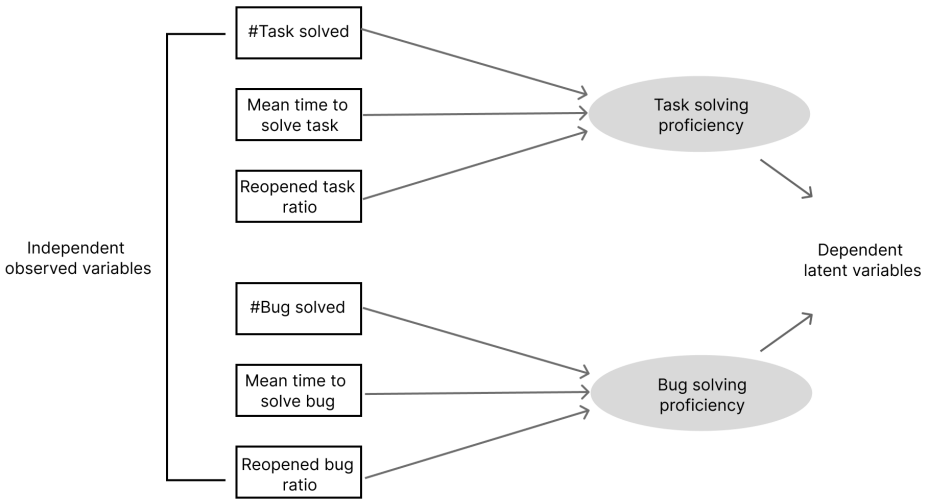

Figure 5.3: Visual Representation of our SEM Model

Similarly the hypothesis here in the other model is that number of task count and mean task solving time have a positive effect on Task Solving Proficiency, while number of may Task reopen ratio has a weaker or indirect effect on Bug Fixing Proficiency.

After implementation we get the covariance and correlation between the observed and unobsereved variables.

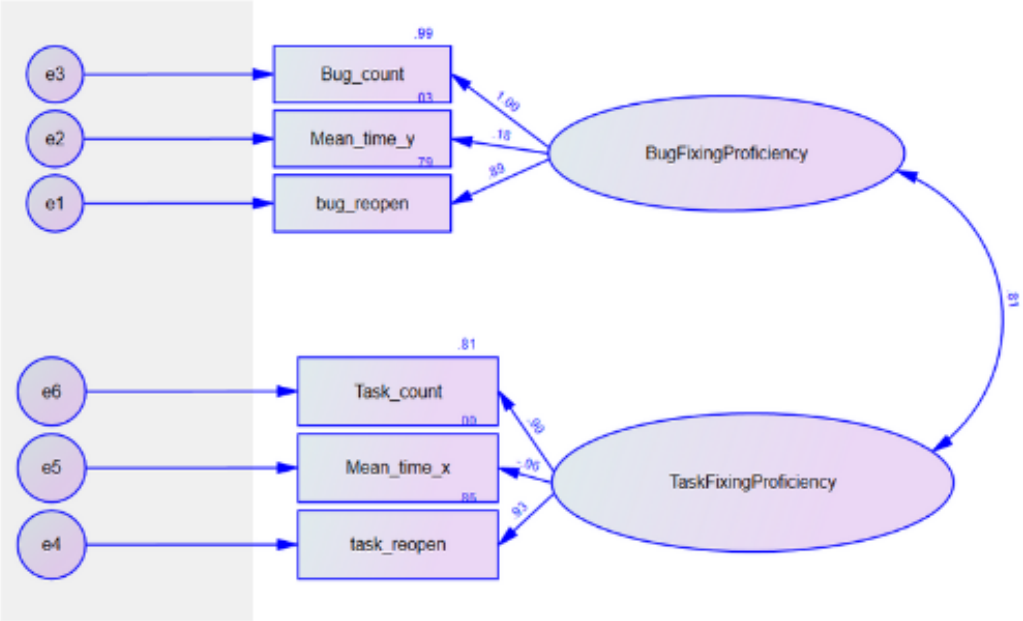

Figure 5.4: Structural Model

5.1.3 Validity of Developer Proficiency

From the SEM Model, we get the new importance weight of the factors affecting the Bug Fixing Proficiency and Task Solving Proficiency. The new weights are:

Table 5.5: Impact Factor Values by SEM

Factors	Denoted as	SEM Impact Value
Number of tasks solved by a developer	*SEM_I1*	0.9
Mean time to solve a task	*SEM_I2*	1
Task reopen ratio	*SEM_I3*	0.6
Number of bugs solved by a developer	*SEM_I4*	0.18
Mean time to solve a bug	*SEM_I5*	0.93
Bug Reopen Ration	*SEM_I6*	0.89

Using these new importance weights we calculate the proficiency of the developers again to validate these new values with our calculated ones. A comparison between these proficiency value is shown in Table 5.6.

From Table 5.6 we can say that the ratio of each row is more or less similar. For de-

Table 5.6: Poficiency comparison (Importance weight from Practitioners vs Importance weight from SEM)

Developer	Proficiency	Proficiency (SEM)
rxin	31.16789009	84.25353221
davies	13.66634942	46.94027764
mengxr	13.49231294	35.63962171
cloud_fan	12.09294203	39.12516773
joshrosen	8.732933354	35.06608786
yumwang	8.692820225	22.53917845
zsxwing	8.356762238	36.13587568

veloper named rxin, proficiency calculated with out impact weight is 31.16789, while with that of SEM, it is 84.253553. Again, for a developer named davies, with proficiency 13.66634 has proficiency of 46.9402 in terms of SEM. So these two calculations are consistent with one another. So our importance weight complies with the SEM.

Answer to the RQ1: *Developer proficiency depends on bug fixing and task solving proficiency. Each has three impact factors, mean time to fix/solve, their total number and their reopen ratio.*

5.2 Proficiency and Bug Fixing Accuracy

As we mentioned earlier in our study, we labelled the developers as highly proficient if they have proficiency range between 60-120, medium proficient if the proficiency range is between 30-60 and if the range is between 10-30 then low proficient.

Research Question 2 aimed to investigate the relationship between a developer's proficiency and bug-fixing efficiency and accuracy.

5.2.1 Bug Fixing Accuracy

The analysis also indicated a strong association between a developer's proficiency and bug-fixing accuracy. Developers with higher proficiency levels demonstrated a higher level of accuracy in identifying and resolving bugs. Their fixes were more precise and reliable, resulting in fewer instances of reopened bugs. The study found that developers with higher proficiency tended to exhibit better bug-fixing accuracy.

We calculate Bug Fixing Accuracy using the following equation -

$$Accuracy_i = \frac{\sum(Bug\_Fixed_i) - \sum(Bug\_Reopen_i)}{\sum Bug\_Fixed\_by\_that\_Developer_i} \tag{5.3}$$

Table 5.7: Proficiency of Developers with Bug Fixing Accuracy

Developer	Proficiency	Bug Fixing Accuracy
dongjoon	112.1009	0.8125
william	63.9719	0.8591
cloud_fan	50.5691	0.875
maxgekk	45.2505	0.7142
ala.luszczak	15.1341	0.5333
joshrosen	14.5759	0.6111

Table 5.7 represents our calculation and we found out that with proficiency, bug fixing accuracy increases.

To analyse the data, we plotted accuracy and proficiency of the developers from table 5.7 and have seen that developers with high proficiency has more accuracy 70% of the time and if proficiency is less then with accuracy decreases.

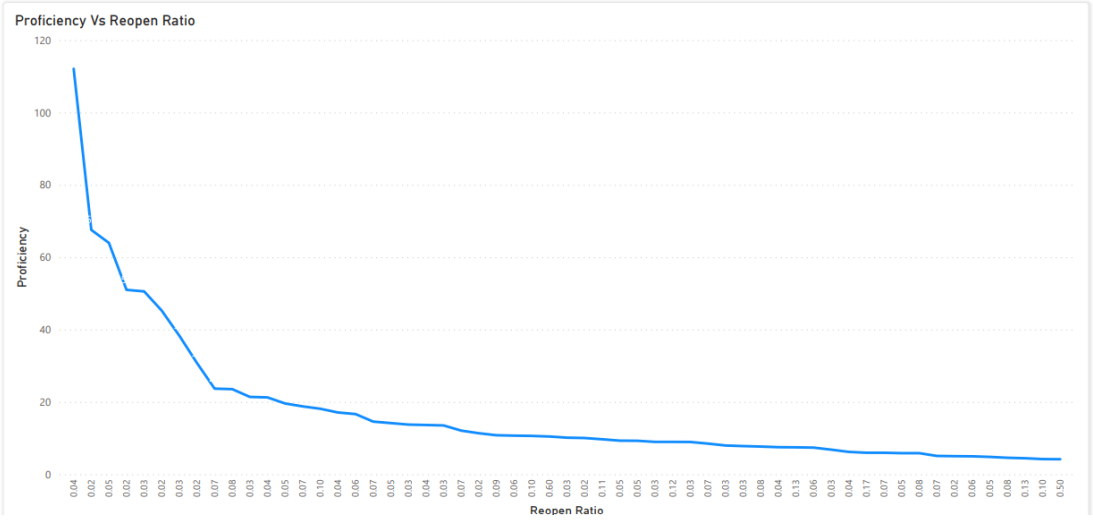

Figure 5.5: Bug Fixing Accuracy Vs Developer Proficiency

Table 5.8 shows the average bug fixing accuracy for the three levels of proficiency. So with proficiency level bug fixing accuracy increases in most cases, around 70% of the time.

Table 5.8: Level of Proficiency of Developers with average Bug Fixing Accuracy

Level of Proficiency	Average Bug Fixing Accuracy
High Proficiency Level	0.8015
Medium Proficiency Level	0.6142
Low Proficiency Level	0.5233

5.2.2 Bug Fixing Efficiency

The analysis demonstrated a significant impact of a developer's proficiency on bug-fixing efficiency. Developers with higher proficiency levels exhibited improved efficiency in fixing bugs. They were able to identify and address bugs more swiftly and effectively compared to developers with lower proficiency levels. The study found a positive correlation between developer proficiency and bug-fixing efficiency, suggesting that developers with higher proficiency tend to resolve bugs in a more efficient manner.

In our study we plotted for each developer their bug fixing time comparing with the component's mean time to fix a bug and have seen that efficiency of developer for fixing bugs in a familiar component is more and they fix it in less time then average bug fixing time of a component.

The analysis demonstrated a significant impact of a developer's proficiency on bug-fixing efficiency. Developers with higher proficiency levels exhibited improved efficiency in fixing bugs 50% of the time. They were able to identify and address bugs more swiftly and effectively compared to developers with lower proficiency levels. The study found a positive correlation between developer proficiency and bug-fixing efficiency, suggesting that developers with higher proficiency tend to resolve bugs in a more efficient manner.

In our study we plotted for each developer their bug fixing time comparing with the component's mean time to fix a bug and have seen that efficiency of developer for fixing bugs in a familiar component is more and they fix it in less time then average bug fixing time of a component.

In our study we plotted for each developer their bug fixing time comparing with the component's mean time to fix a bug and have seen that efficiency of developer for fixing bugs in a unfamiliar component that is the component where they have only fixed bug but haven't solved any task is less and they fix it in more time then average bug fixing time of a component.

So from this our result shows that efficiency of a developer increases when he solves bug in a familiar component and decreases if the component is unfamiliar.

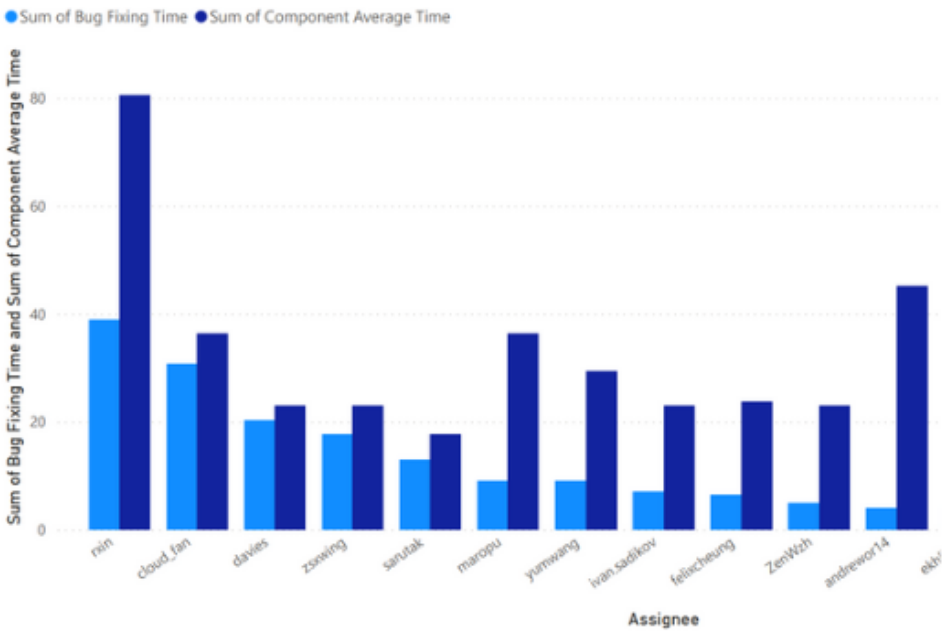

Figure 5.6: Bug Fixing Time of Familiar Component

Overall, the empirical study provides compelling evidence that a developer's proficiency significantly affects bug-fixing efficiency and accuracy. Developers with higher proficiency levels tend to exhibit improved efficiency and accuracy in fixing bugs. These findings suggest that organizations should focus on enhancing developer proficiency through training, skill development initiatives, and knowledge sharing platforms to optimize bug-fixing processes.

Answer to the RQ2: *With the increase in Developer's Proficiency, his Bug-fixing Efficiency and Accuracy increases 50% of the time.*

5.3 Developer's Efficiency over time

The empirical study aimed to explore the impact of developer proficiency on bug fixing efficiency and accuracy over time. The result analysis focused on understanding how a developer's efficiency in fixing bugs evolved and whether proficiency had a discernible influence.

In our data a pattern was visible that with proficiency efficiency increases over time. We divided the time range in 2013 to 2015, 2016 to 2018 and 2019 to 2023.

Table 5.9 shows a positive correlation between developer proficiency and bug fixing efficiency over time. Developers who demonstrated higher proficiency levels tended

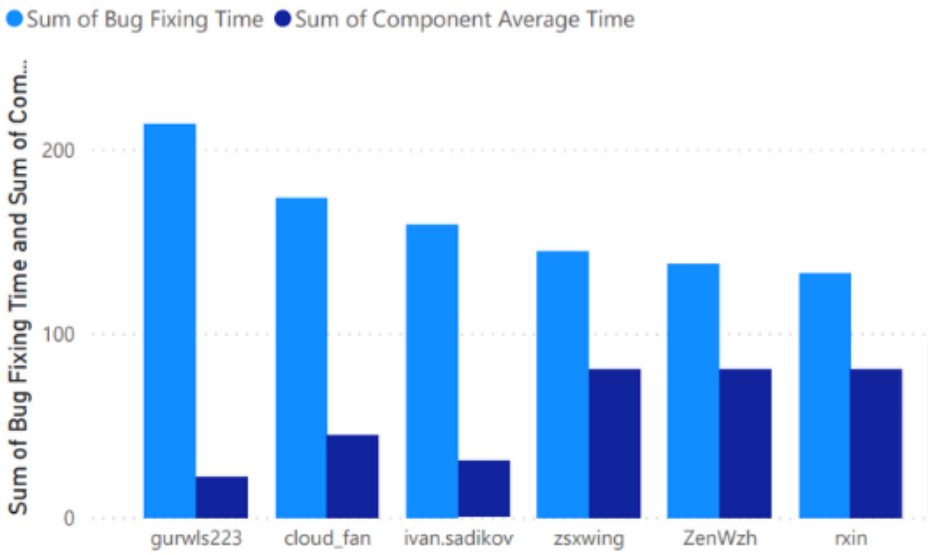

Figure 5.7: Bug Fixing Time of Unfamiliar Component

Table 5.9: Mean time to solve a bug and Developer's Proficiency over time

Developers	Time Span 1 (2013-2015)		Time Span 2 (2016-2018)		Time Span 3 (2019-2023)	
	MTSB	Proficiency	MTSB	Proficiency	MTSB	Proficiency
techaddict	110.25	0.11	87	0.52	1	5.03
ueshin	85.4	2.72	20	4.00	2.11	15.53
maropu	68.67	1.02	27.3	2.53	21.875	6.21
srowen	54.76	2.81	38.12	3.04	11.26	5.25

to exhibit more significant improvements in efficiency. Their enhanced technical skills and domain knowledge translated into quicker bug resolution, reduced rework, and improved accuracy in fixing bugs.

Now plotting the value in a graph that is Developer with mean time to fix a bug we have seen that over the time period for each developer their efficiency increases and if efficiency increases proficiency also increase.

Overall, the empirical study provides compelling evidence that developer proficiency has a significant impact on bug fixing efficiency over time. As developers' proficiency levels increase through targeted interventions, their efficiency in fixing bugs improves and sustains over the course of the study. These findings underscore the importance of investing in continuous proficiency development to optimize developer performance in bug fixing activities.

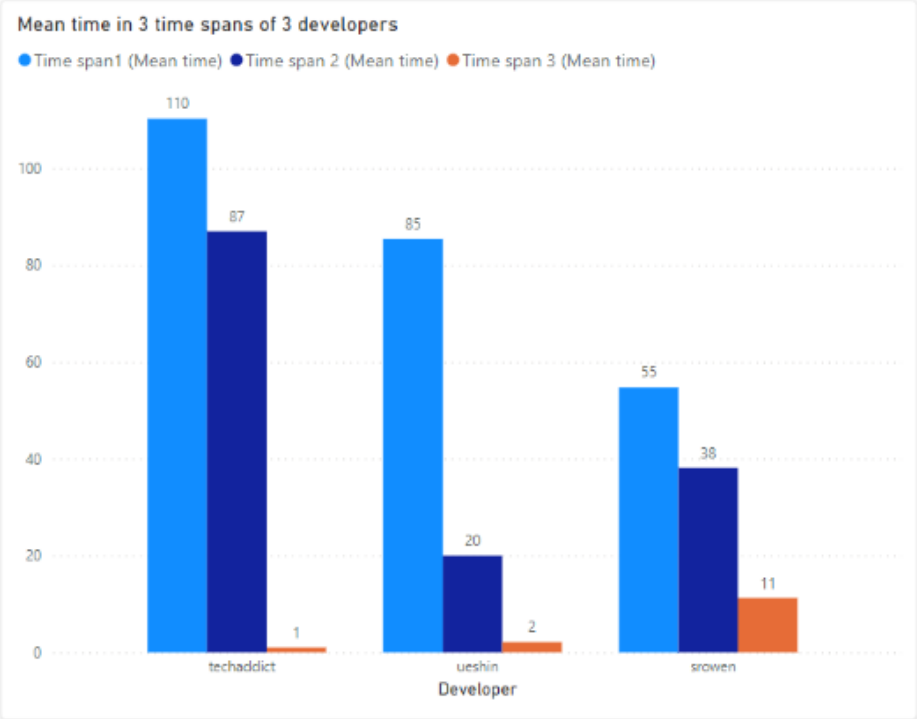

Figure 5.8: Developer's Efficiency Over Time

Answer to the RQ3: *Developer's Bug Fixing Efficiency increases with time.*

5.4 General Discussion

The findings of our empirical investigation on the effect of developer skill on bug fixing efficiency and accuracy offer strong support for the idea that performance in bug fixing is greatly influenced by proficiency. The research results provide profound understandings of the value of developer proficiency and its implications for improving bug-fixing procedures.

First off, the significant positive association between developer skill and bug-fixing effectiveness emphasizes the critical role that technical expertise, subject-matter expertise, and task proficiency play in reducing bug-solving times. Our research unequivocally shows that developers with greater competency levels are better able to locate and fix errors in a timely and efficient manner. Shorter development cycles and increased output are the results of this improved efficiency. These findings highlight how crucial it is to give proficiency-related factor development inside organizations top priority because these variables directly influence how effectively bugs are fixed.

Second, a basic characteristic of bug fixing performance is shown by the relation-

ship between developer skill and problem repair accuracy. According to our analysis, developers with higher competency levels are more accurate at locating and fixing defects, which leads to fewer cases of reopened bugs. The importance of proficiency in assuring the caliber of bug solutions and reducing rework is shown by this finding. Organizations may greatly improve the accuracy of bug patches, which will increase software stability and customer happiness, by investing in the skill development of developers.

Our study also looked at the long-term effects of developer skill on the effectiveness and precision of bug fixes. The findings offer strong proof that developers' bug-fixing ability increases as their proficiency increases thanks to targeted training programs and skill-enhancement activities. This research emphasizes how successful continuous proficiency development operations are in bringing about long-lasting and sustainable improvements in bug fixes. To take advantage of these advantages, businesses should continuously offer developers the chance to improve their skills while promoting a culture of lifelong learning.

Organizations must give the evaluation and improvement of developer proficiency in bug repair activities top priority in light of these findings. Organizations may make educated judgments about investments in training, mentoring, and knowledge-sharing platforms by recognizing the enormous influence that competency has on bug fixing efficiency and accuracy. In addition, encouraging a culture of ongoing learning and development within software development teams will allow competency to build over time, resulting in continuously better bug-fixing performance and overall software quality.

As a result, our study emphasizes how crucial developer expertise is to the effectiveness and efficiency of issue fixing. Organizations can streamline bug-fixing procedures, cut down on development cycle time, increase software reliability, and eventually improve customer pleasure by comprehending and using the power of proficiency.

Chapter 6

Conclusion and Future Work

In this research work, we have done a comparative analysis and found that the Doc2Vec model works the best for difficulty based question classification. The motivation behind this work was to recommend questions to suitable users based on the question difficulty level. So, for that, we needed to categorize the questions into different difficulty levels, which we have already done. In our work, we mostly considered Java-related questions. As Java is the most popular programming language with a well-built developers' community, we could gather diverse types of questions in Stack Overflow to support our research work.

So, our future works will explore the performance of our model on other programming languages like C#, Python, Pearl, Ruby, etc. Other than that, to build the desired recommendation system, the user base is also needed to be categorized for getting the question answered. The users can be categorized based on their activities on Stack Overflow(asking and answering questions), expertise level, and activeness. To measure the users' expertise level, further works can be done, like proposing some framework or model. The temporal data related to users' activities are also essential for recognizing the active users because recommending questions to an inactive user will be futile.

And finally, we need a recommendation system that would recognize the hidden relations between the question types and the users and recommend the questions to users with enough expertise to answer the questions. So, there are a lot of scopes for future works from this research work.

REFERENCES

[1] S. Kalvala and R. Warburton, "A formal approach to fixing bugs," 09 2011, pp. 172–187.

[2] C. Wang, Y. Li, L. Chen, W. Huang, Y. Zhou, and B. Xu, "Examining the effects of developer familiarity on bug fixing," *Journal of Systems and Software*, vol. 169, p. 110667, 2020.

[3] A. Yadav, S. K. Singh, and J. S. Suri, "Ranking of software developers based on expertise score for bug triaging," *Information and Software Technology*, vol. 112, pp. 1–17, 2019.

[4] V. R. Basili and B. T. Perricone, "Software errors and complexity: an empirical investigation0," *Communications of the ACM*, vol. 27, no. 1, pp. 42–52, 1984.

[5] A. Radu and S. Nadi, "A dataset of non-functional bugs," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 399–403.

[6] D. J. Dean, H. Nguyen, X. Gu, H. Zhang, J. Rhee, N. Arora, and G. Jiang, "Perfscope: Practical online server performance bug inference in production cloud computing infrastructures," in *Proceedings of the ACM Symposium on Cloud Computing*, 2014, pp. 1–13.

[7] A. B. Sánchez, P. Delgado-Pérez, I. Medina-Bulo, and S. Segura, "Tandem: A taxonomy and a dataset of real-world performance bugs," *IEEE Access*, vol. 8, pp. 107 214–107 228, 2020.

[8] A. Khatun and K. Sakib, "A bug assignment approach combining expertise and recency of both bug fixing and source commits." in *ENASE*, 2018, pp. 351–358.

[9] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 1–1.

[10] S. A. Licorish and S. G. MacDonell, "Exploring software developers' work practices: Task differences, participation, engagement, and speed of task resolution," *Information & Management*, vol. 54, no. 3, pp. 364–382, 2017.

[11] H. Shafiq and z. Arshad, "Automated debugging and bug fixing solutions: A systematic literature review and classification," Ph.D. dissertation, 11 2014.

[12] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code! examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 4–14.

[13] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, "Identification of coordination requirements: Implications for the design of collaboration and awareness tools," in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, 2006, pp. 353–362.

[14] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: an empirical case study," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 521–530.

[15] F. P. Brooks Jr, *The mythical man-month: essays on software engineering*. Pearson Education, 1995.

[16] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, no. 11, pp. 1268–1287, 1988.

[17] A. Espinosa, R. Kraut, J. Lerch, S. Slaughter, J. Herbsleb, and A. Mockus, "Shared mental models and coordination in large-scale, distributed software development," 2001.

[18] S. Zaman, B. Adams, and A. E. Hassan, "Security versus performance bugs: a case study on firefox," in *Proceedings of the 8th working conference on mining software repositories*, 2011, pp. 93–102.

[19] J. Imseis, C. Nachuma, S. Arifuzzaman, M. Zibran, and Z. A. Bhuiyan, "On the assessment of security and performance bugs in chromium open-source project," in *International Conference on Dependability in Sensor, Cloud, and Big Data Systems and Applications*. Springer, 2019, pp. 145–157.

[20] L. D. Panjer, "Predicting eclipse bug lifetimes," in *Fourth international workshop on mining software repositories (MSR'07: ICSE workshops 2007)*. IEEE, 2007, pp. 29–29.

[21] H. Zeng and D. Rine, "Estimation of software defects fix effort using neural networks," in *Proceedings of the 28th Annual International Computer Software and*

Applications Conference, 2004. COMPSAC 2004., vol. 2. IEEE, 2004, pp. 20–21.

[22] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, 2011, pp. 1–8.

[23] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software defect association mining and defect correction effort prediction," *IEEE Transactions on Software Engineering*, vol. 32, no. 2, pp. 69–82, 2006.

[24] J. Han, M. Kamber, and J. Pei, "2 - getting to know your data," in *Data Mining (Third Edition)*, third edition ed., ser. The Morgan Kaufmann Series in Data Management Systems, J. Han, M. Kamber, and J. Pei, Eds. Boston: Morgan Kaufmann, 2012, pp. 39–82. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123814791000022