**AN INTEGRATION OF MACHINE LEARNING TECHNIQUE ON SOFTWARE BUG REPORTING SYSTEM**

A Capstone Project Presented to the Graduate Program

College of Engineering and Technology

Pamantasan ng Lungsod ng Maynila

In Partial Fulfillment of the Requirements for the Degree

Master’s in Information Technology

­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

By

Joane Marie F. Llamera

Dr. Khatalyn E. Mata

Thesis Adviser

August 2021

# **TABLE OF CONTENTS**

[TABLE OF CONTENTS ii](#_Toc81841515)

[LIST OF FIGURES iv](#_Toc81841516)

[LIST OF TABLES v](#_Toc81841517)

[INTRODUCTION 1](#_Toc81841518)

[1.1 Background of the Study 1](#_Toc81841519)

[1.2 Statement of the Problem 2](#_Toc81841520)

[1.3 Objectives of the Study 2](#_Toc81841521)

[1.4 Scope and Limitations 3](#_Toc81841522)

[1.5 Significance of the Study 3](#_Toc81841523)

[1.6 Definition of Terms 3](#_Toc81841524)

[REVIEW OF RELATED LITERATURE 4](#_Toc81841525)

[2.1 Related Literature and Related Studies 4](#_Toc81841526)

[THEORETICAL FRAMEWORK 5](#_Toc81841527)

[3.1 Conceptual Framework 5](#_Toc81841528)

[METHODOLOGY 6](#_Toc81841529)

[4.1 Requirements Modeling 6](#_Toc81841530)

[4.2 Quick Design 7](#_Toc81841531)

[1.2.1 Context Diagram 7](#_Toc81841532)

[1.2.2 Data Flow Diagram 7](#_Toc81841533)

[1.2.3 Use Case Diagram 7](#_Toc81841534)

[4.2.4 System Flowcharts 8](#_Toc81841535)

[4.3 Building Prototype 9](#_Toc81841536)

[4.4 User Evaluation 9](#_Toc81841537)

[4.5 Refining Prototype 9](#_Toc81841538)

[4.6 Engineer Product 9](#_Toc81841539)

[LIST OF REFERENCES 10](#_Toc81841540)

# **LIST OF FIGURES**

[Figure 4.1 Prototype Model 6](#_Toc81841560)

[Figure 4.5 Context Diagram 7](#_Toc81841561)

[Figure 4.6 Data Flow Diagram 7](#_Toc81841562)

[Figure 4.7 Use Case Diagram 7](#_Toc81841563)

# **LIST OF TABLES**

**No table of figures entries found.**

**Chapter One**

# **INTRODUCTION**

## **Background of the Study**

In the lifecycle of a software project, bug fixing is an essential aspect of the development and maintenance phases. Software bugs must be dealt swiftly in large-scale software projects wherein erroneous bug report assignment to development teams can be quite costly. Within organizations, bug tracking allows for the assigning, monitoring and resolution of issues and bugs. A bug is a coding error that can lead to anomalous program behaviour. When software testers find a bug, they will create a bug report to begin the process of fixing it. Issue report assignment is a crucial phase in the process of locating and correcting a bug, since it is the skill of matching an open bug report to the most likely developer to handle it.

With ever-larger software development systems including more workers with varying skills, it's important to consider how bugs are assigned to technical groups rather than to a single developer. The classification of defects is a crucial phase in the bug correction process as it sends the errors to a key developer who can fix them (Chauhan et. Al, 2020). Manual bug triaging can be a time-consuming operation due to the enormous volume of bug reports sent every day. Additionally, assigning a bug to the incorrect team or developer increases the cost and time to remediate the bug.

The use of recommenders for bug report triage judgments is especially significant in large software development projects, where both the frequency of reported issues and the huge number of active engineers might make it difficult to identify the appropriate developer to work on a specific issue. For example, on a daily basis, 135 reported problems are submitted to Mozilla's open-source system (Liu et al., 2013). In these huge open systems, managing a high volume of new bugs submitted everyday by many reports is a demanding effort.

Bug assignment, or the process of assigning defects, is hampered by several factors: it is labor-intensive, time-consuming, and error-prone if done manually; also, it is difficult to maintain track of current engineers and their competence in large projects. Growth makes it even more difficult to find the right developer to fix a new bug. For example, as projects add more components, modules, developers, and testers (Increase, 2009), the number of bug reports submitted daily grows, making manually recommending developers based on their expertise is difficult.

According to a survey conducted by the National Institute of Standards and Technology (NIST), the annual cost of software vulnerabilities is estimated to be around $59.5 billion (NIST, 2002). According to some software maintenance studies, maintenance costs account for at least 50%, and in some cases more than 90%, of total costs associated with a software product (Koskinen, 2003; Seacord et al., 2003), while other estimates place maintenance costs at several times the cost of the initial software version (Koskinen, 2003; Seacord et al., 2003). (Sommerville, 2004). According to these studies, improving the bug-fixing process will minimize evolution effort and lower software development costs.

## **Statement of the Problem**

Software defects are an unavoidable part of the software development process. While software bugs must be fixed for the product to be of high quality, addressing them in a timely manner comes at a cost. Many organizations use issue tracking systems like Mozilla BugZilla and Atlassian JIRA to deal with them in a methodical way. While issue tracking systems have been proven to be effective in managing issues, many actions associated with bug resolution require a significant amount of time and effort.

According to Jeong et al.’s research, 44% of problems are assigned to the wrong developer. Manual defect assignment is time-consuming and error-prone, according to multiple studies (Baysal et al, 2009; Jeong et al, 2009; Bhattacharya et al, 2012), resulting in "bug tossing" (i.e., reassigning problem reports to another developer) and delayed bug remedies.

Some of the key processes that can be greatly improved on is to utilize a machine learning technique in order to auto assign the bug to the appropriate resolving team based on previously resolved bugs and display related bugs that had been already fixed.

## **Objectives of the Study**

The main objective of this capstone project is to apply Machine Learning algorithm for auto-assignment and extraction of past fixed bugs for Software Bug Reporting.

Specifically, this capstone project seeks to achieve the following objectives:

1. To extract previously fixed bug based on keyword extraction using TF-IDF (Term Frequency — Inverse Document Frequency) Algorithm.
2. To automatically categorize newly reported bugs and recommend a developer to resolve the bug using NLP (Natural Language Processing) RBF (Radial Basis Function) - SVM (Support Vector Machines) algorithm.
3. To evaluate the accuracy of the auto-assignment feature using ISO 9126.
   1. **Scope and Limitations**

This study will focus on creating a prototype bug reporting tool with enhanced feature on bug assignment process and related bug extraction by utilizing machine learning and NLP techniques.

The study will focus on determining bug team assignee based on previously resolved bugs in the system and extract related bugs for efficiency of use.

The data to be used will be from a proprietary software repository.

For instances wherein there is no existing keyword match, manual intervention is necessary where the bug reporter shall select the appropriate assignment group.

* 1. **Significance of the Study**

Results obtained from this capstone project will benefit the following stakeholders:

* 1. **Definition of Terms**

**Chapter Two**

# **REVIEW OF RELATED LITERATURE**

This chapter presents the different research and other literatures from both foreign and local researchers, which have significant bearings on the variables included in the research. It focuses on several aspects that will help in the development of this study. The literatures of this study come from books, journals, articles, electronic materials such as PDF or E-Book, and other existing thesis and dissertations, foreign and local which are believed to be useful in the advancement of awareness concerning the study.

## **2.1 Related Literature and Related Studies**

**Chapter Three**

# **THEORETICAL FRAMEWORK**

## **3.1 Conceptual Framework**

This section aims to demonstrate the overview of the final product of this capstone project. An I-P-O (Input-Process-Output) model will be used as the conceptual schema of the system It identifies relevant variables, inputs, mappings, and other components and how they will interact with each other. This includes all the underlying concepts and their associated mappings based on the system’s use.

**Chapter Four**

# **METHODOLOGY**

The proponent of this capstone project used prototype method in delivering the objectives of this project.

Diagram

Description automatically generated

**Figure 4.1 Prototype Model**

The phases of the prototype model involve the following steps:

## **4.1 Requirements Modeling**

## **4.2 Quick Design**

At this stage the initial prototype is developed, where the very basic requirements are showcased, and user interfaces are provided. This stage would provide a high-level view of the application to the client.

### **Context Diagram**

Diagram

Description automatically generated

**Figure 4.5 Context Diagram**

### **Data Flow Diagram**

**Figure 4.6 Data Flow Diagram**

The researcher used the Data Flow Diagram, which is a dramatic representation of the information flow within a system that shows how information enters the system and leaves the system, what changes the information and where it is stored (Kendall, 2005).

### **Use Case Diagram**

**Figure 4.7 Use Case Diagram**

### **4.2.4 System Flowcharts**

**1. Recommend Developer**

1.1. Generate model from Fixed Bugs

a. Take 'Description' and 'Summary' fields

b. Pre-process fields (NLP Preprocessing)

- tokenize

- word stemming/lemmatization

c. Prepare train and test data sets

d. Vectorize Words

e. Apply ML algorithm

1.2. Use data from 1.1 to provide developer name to new tickets

**2. Get Similar Bugs via Keywords**

2.1. Create keyword datastore from existing bugs

a. Take 'Description' and 'Summary' fields

b. Pre-process fields (NLP Preprocessing)

- tokenize

- word stemming/lemmatization

c. Calculate TF-IDF

d. Apply ranking

2.2. Incoming Bugs

a. Apply automatic keyword extraction to new Bug

b. Match keyword extracted from existing bugs' keywords datastore

## **4.3 Building Prototype**

At this stage, system requirements and other components necessary to develop the proposed application will be identified.

## **4.4 User Evaluation**

This is the stage where the application users would evaluate the application based on its required features. The capstone project will utilize a checklist type of survey questionnaire in which the respondents will be able to answer faster and easier at their convenience.

## **4.5 Refining Prototype**

In this stage, any dissatisfaction with the prototype at this level will result to a revision based on the given requirements. The new prototype will be re-evaluated, and the process continued until such time that the requirements identified by the end-user were met. Revisions will be done based on the user’s comments and suggestions during the evaluation of the developed application.

## **4.6 Engineer Product**

The last stage of this approach will conclude with the confirmation and approval of the application by the end-users. This will also be referred to as the user acceptance phase. It is also in this phase that the proponent will be able to appraise the overall performance of the final system, using the predetermined indices or indicators such as functionality, efficiency, reliability, usability, and portability.

# **LIST OF REFERENCES**