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

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. A bug is a coding error that can lead to anomalous program behaviour. Within organizations, bug tracking allows for the assigning, monitoring and resolution of issues and bugs. When software testers find a bug, they will create a bug report to begin the process of fixing it. Software bugs must be dealt swiftly in large-scale software projects wherein erroneous bug report assignment to development teams can be quite costly. 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.

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.

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, the number of bug reports submitted daily grows, making manually recommending developers based on their expertise is difficult.

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 every day is a demanding effort.

## **Statement of the Problem**

Software bugs 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 or team. 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.

Even with issue trackers, it is still a time-consuming task to assign bug reports to programmers for fixing, since many bug reports have to be assigned (Anvik et al., 2006) and some projects have many team of programmers.

Some of the key processes that can be greatly improved on Bug Reporting System is to utilize a machine learning technique for auto assignment of the bug to the appropriate resolving team based on previously resolved bugs.

## **Objectives of the Study**

The main objective of this capstone project is to apply Machine Learning algorithm for auto-assignment of 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 auto assign it to the correct resolving team using NLP (Natural Language Processing) and RBF (Radial Basis Function) - SVM (Support Vector Machines) algorithm.
3. To evaluate the accuracy of bug auto-assignment using ISO 9126.
   1. **Scope and Limitations**

This study will focus on creating a prototype bug reporting tool with enhanced feature on bug auto assignment process by utilizing machine learning techniques. The primary goal of the application is on building classifiers by analyzing patterns of previously resolved bug reports and use it to automatically assign the bug to the appropriate resolving team.

Other attributes of a defect report such as Title, Description, Priority, Severity, Reported Date, Reported By, Status will be manually taken as an input to the bug reporting system.

The study will exclude other processes in a defect triage process and will only be concerned on identification of the bug assignee.

For instances wherein there is no existing keyword matches from the trained data model, manual intervention is necessary where the bug reporter shall select the appropriate assignment team.

* 1. **Significance of the Study**

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

**Bug Reporter.** The bug auto assignment feature will decrease the manual effort and time spent on identifying who is the appropriate team to resolve the reported software issue. Having an automated assignment feature will remove the manual error of assigning the bug an incorrect team.

**Customers/Clients.** By reducing the error on incorrect bug assignment, reported issues are fixed in a timely manner as bug tossing instances are reduced.

**Software Companies.** Efficient bug fix process can improve not only the appeal of an app to consumers, but also the reputation of the company that created it. The reduced time spent on bug assignment will ultimately contribute om reducing the cost of bug fixing.

* 1. **Definition of Terms**

Bug

Bug Tossing

Bug Report

Bug Reporting System

Bug Triage

Tokenize

Lemmatization

Vectorize

**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**

**Bug Triage Process**

Software defects are a problem that IT companies all around the world have to deal with software bugs. Managing programming problems consumes more than 45 percent of programming companies' budgets. Bug repositories, such as Bugzilla, a prominent and open-source bug repository, store software bugs. A bug reporting system is generally an essential element of a satisfactory software development infrastructure and regular use of a bug or issue reporting system consider one of the “sign of an honourable software team”. A large element of a bug reporting system is a database that tracks and records the information about bugs which is known (Jalbert and Weimer, 2008).

When a software bug is discovered, it is tested by a reporter, who could be a tester, developer, or deployer, and if it is determined to be a genuine bug, the occurrence is reported. A bug report is a document that contains information on a bug that can be used to reproduce it. Once a bug report is created, a human triager will allocate the bug to a developer. If the assigned developer is unable fix the bug, the bug is migrated to another developer. The process of assigning a bug report to an appropriate developer is called bug triage (Arudkar et al., 2017). Bug triage's purpose is to review, prioritize, and assign defect solutions. The triage team must confirm the severity of the fault, make necessary changes, conclude defect resolution, and assign resources. This method is primarily utilized in agile project management (Hamilton, 2021).

The bug reports in a bug repository are called bug data. In big software development projects, the Automated Bug Tracking System (BTS) monitors bug reports and a list of developers who work on correcting them.

Several works have been done to build automatic bug triagers using machine learning algorithms. It is not surprising that issue trackers constitute a central point of focus in current software engineering empirical research such as predicting the severity of reported bug in a study conducted by Lamkanfi et al. (2010). In the study of Jonsson et al. (2016), they presented a bug triager based on a machine learning ensemble and tested it on five industrial applications. They combined well-known machine learning algorithms to increase the performance of automatic bug triage. A study by Kumari snd Singh (2018) built classifiers based on machine learning techniques Naïve Bayes (NB) and Deep Learning (DL) using entropy-based measures for bug priority prediction while considering the severity, summary weight and entropy attribute to predict the bug priority.

**Support Vector Machine (SVM)**

Support vector machine, according to Hearst et al. (1998), provides a classification learning model and algorithm rather than a regression model and algorithm. It manipulates the simple mathematical model y = wx + to allow for linear domain division. There are two types of support vector machines: linear and nonlinear (Hastie et al. 2009). If the data domain can be divided linearly (e.g., straight line or hyperplane) to separate the classes in the original domain, it is called a linear support vector machine. Nonlinear support vector machines are used when the data domain cannot be divided linearly but can be changed to a space called the feature space where the data domain may be divided linearly to separate the classes.

The mapping of the data domain into a response set and the division of the data domain are the steps in the linear support vector machine. The mapping of the data domain to a feature space using a kernel function (Scholkopf et al. 1999), the mapping of the feature space domain into the response set, and finally the division of the data domain are the steps in nonlinear support vector machines.

Kanwal et al. (2010) suggested a bug priority recommender that is based on classification techniques such as SVM. The bug priority recommender assigns a priority rating to newly arrived problems automatically. The eclipse dataset platform product was validated by the authors. Kanwal et al. (2012) expanded on this research by comparing two classifier algorithms, namely Support Vector Machine and Nave Bayes. The results reveal that Support Vector Machine outperforms Nave Bayes for textual features while Nave Bayes outperforms Support Vector Machine for categorical data.

**Supervised Machine Learning Techniques in Bug Reporting**

Yuan Tian et al. (2013) made an attempt using a new framework called DRONE (PreDicting PRiority via Multi-Faceted FactOr ANalysEs). GRAY is a new classification engine developed by the authors (ThresholdinG and Linear Regression to ClAssifY). The experiment was carried out using Eclipse project training and testing data sets. The authors compared SeverisPrio and SeverisPrio+ to his planned work DRONE. Menzies et al. proposed the SEVERIS approach as a foundation for bug severity evaluation (2008). Tian et al. (2015) expanded on the research and used an automated approach known as DRONE.

It uses machine learning and information from a reported bug to forecast the priority level. The experimental findings were validated using Eclipse project bug reports, and the results suggest that DRONE can outperform a baseline method. Sharma et al. used summary attributes to assess the efficacy of various machine learning techniques, including SVM, NB, KNN, and NNet, in predicting bug priority. Except for the NB technique, the accuracy of different classifier algorithms in predicting the priority of reported problems inside and across projects is found to be above 70%.

Mani et al. (2018) introduced a deep learning system that learns a paragraph level representation while retaining word ordering and semantic relationships over a longer context. Multinomial naive Bayes, cosine distance, support vector machines, and softmax classifier were among the classifiers utilized by the authors. Bug reports from three famous open-source bug repositories - Google Chromium (383,104), Mozilla Core (314,388), and Mozilla Firefox (314,388) - were used to validate the experimental result (162,307). In all three datasets, DBRNN-A combined with the softmax classifier outperforms the bag-of-words model, improving the rank-10 average accuracy.

**Natural Language Processing (NLP)**

Natural language processing (NLP) is an area of AI that aids computers in interpreting and manipulating language. The goal of NLP researchers is to learn how people perceive and use language so that appropriate tools and techniques may be developed to help computers understand and manipulate natural languages to execute tasks (Chowdhury, 2003).

In a given context or domain, a word or sentence may have a specific meaning or connotation, and it may be related to a large number of other words and sentences. According to Liddy (1998) and Feldman (1999), it is critical to be able to discern between the seven interdependent levels that humans utilize to derive meaning from written or spoken languages in order to understand natural languages:

Phonetic or phonological level - deals with pronunciation

Morphological level - deals with the smallest parts of words that carry meaning, and suffixes and prefixes

* Lexical level - deals with lexical meaning of words and parts of speech analyses
* Syntactic level - deals with grammar and structure of sentences
* Semantic level - deals with the meaning of words and sentences
* Discourse level - deals with the structure of different hands of text using document structures
* Pragmatic level - deals with the knowledge that comes from the outside world, i.e., from outside the content of the document

A natural language processing system may involve all or some of these levels of analysis.

**Chapter Three**

# **THEORETICAL FRAMEWORK**

**Bug Triaging Process**

**Tokenization**

In Natural Language Processing, tokenization is a typical activity (NLP). Both classic NLP approaches like Count Vectorizer and Advanced Deep Learning-based architectures like Transformers rely on this phase. Tokenization is the process of breaking down a large chunk of text into smaller tokens. Tokens can be words, characters, or sub words in this case. Tokenization can thus be divided into three categories: word, character, and sub word (n-gram characters) tokenization.

The most popular method for creating tokens is to use space. The tokenization of the statement yields three tokens – Never-give-up, assuming space as a delimiter. Because each token is a word, it is a Word tokenization example.

Similarly, tokens can be either characters or sub words. For example, let us consider “smarter”:

Character tokens: s-m-a-r-t-e-r

Sub word tokens: smart-er

The most often used tokenization algorithm is word tokenization. It divides a chunk of text into distinct words using a delimiter. Different word-level tokens are created depending on the delimiters. Word tokenization includes pre-trained word embeddings like Word2Vec and GloVe.

**Lemmatization**

Lemmatization is the process of combining a word's several inflected forms into a single item for analysis. Similar to stemming, lemmatization adds context to the words. As a result, it joins together words that have comparable meanings. Stemming and lemmatization are both part of text preprocessing. Many individuals are perplexed by these two terms. Some people confuse the two. Because lemmatization performs morphological examination of the words, it is preferable over stemming.

Applications of lemmatization are:

* Used in comprehensive retrieval systems like search engines.
* Used in compact indexing

Lemmatization is a term that relates to performing things correctly using a vocabulary and morphological analysis of words, with the goal of removing inflectional endings solely and returning the base or dictionary form of a word, also known as the "lemma." When faced with the token "saw," stemming might only yield s, whereas lemmatization might try to return "see" or "saw," depending on whether the token was used as a verb or a noun. Stemming and lemmatization may also differ in that stemming usually collapses derivationally related words, but lemmatization usually just collapses a lemma's different inflectional forms. Linguistic processing for stemming or lemmatization is generally done via a separate plug-in component to the indexing process, and there are a number of commercial and open-source options available.

A picture containing text, wall, screenshot

Description automatically generated

**Stop Words**

Stopwords are English words that don't add much to a sentence's meaning. They can be safely ignored without jeopardizing the sentence's meaning. Words like the, he, and have, for example. Both while indexing entries for searching and retrieving them as the result of a search query, search engines have been configured to ignore them.

Input: “This is a sample sentence, showing off the stop words filtration.”

Output: ['This', 'sample', 'sentence', 'showing', 'stop', 'words', 'filtration']

**Word Vectorization**

It's a method for converting a group of text documents into numerical feature vectors. There are other approaches for converting text data into vectors that the model can comprehend, but the TF-IDF method is by far the most prevalent. The term "Term Frequency — Inverse Document Frequency" is an abbreviation that stands for "Term Frequency — Inverse Document Frequency."

The TF-IDF is widely employed in machine learning algorithms for a variety of purposes, including stop-word elimination. These are terms like "a, the, an, it" that are commonly used yet provide little information. Term frequency and inverse document frequency are the two components of the TF-IDF.

In a simple language, TF-IDF can be defined as follows:

In the TF-IDF, a high term frequency (in the provided document) and a low document frequency of the term in the entire collection of documents result in a high weight.

The TF-IDF algorithm is a combination of two algorithms.

* Term Frequency
* Term frequency (TF) is how often a word appears in a document, divided by how many words there are.
* TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)
* Inverse document frequency
* Term frequency is how common a word is, inverse document frequency (IDF) is how unique or rare a word is.
* IDF(t) = log\_e(Total number of documents / Number of documents with term t in it)

**Support Vector Machines (SVM)**

SVM stands for Support Vector Machine and is a supervised machine learning technique that can be used for classification and regression. SVMs are based on the concept of determining the optimum hyperplane for dividing a dataset into two classes.

**Types of SVM**

* Linear SVM: Linear SVM is used to categorize data that is linearly separable, i.e. a dataset that can be divided into two groups using a single straight line. These data points are referred to as linearly separable data, and the classifier is known as a Linear SVM classifier.
* Non-linear SVM: Non-linear SVM is used to classify data that cannot be classified using a straight line. We do this by employing a kernel approach, which places data points in a higher dimension from which they may be separated using planes or other mathematical functions. Non-linear data is referred to as such, and the classifier utilized is known as a Non-linear SVM classifier.

To begin, a set of points from each class is plotted and visualized as shown below. We can efficiently separate these two classes in a 2-d space by simply applying a straight line. However, various lines can be used to classify these classes. You can choose from a variety of lines or hyperplanes (green lines). The inquiry will be: which of these lines is appropriate for classification?

Chart, scatter chart

Description automatically generated

Essentially, choose the hyper-plane that best separates the two groups. This is accomplished by increasing the distance between the closest data point and the hyper-plane. The better the hyperplane and the better the categorization results, the bigger the distance. The hyperplane chosen has the greatest distance from the nearest point from each of those classes, as shown in the diagram below.

The support vectors of the hyperplane are the two dotted lines that run parallel to the hyperplane and cross the nearest points of each of the classes. The distance between the supporting vectors and the hyperplane is now referred to as a margin. And the SVM algorithm's goal is to maximize this margin. The hyperplane with the greatest margin is the best hyperplane.

Consider the distinction between good and harmful cells. xi is an n-dimensional feature vector that can be plotted in n-dimensional space. The class yi is assigned to each of these feature vectors. The class yi can be either a +ve or a -ve (for example, good=1 or good=-1). The hyperplane's equation is y=w.x + b = 0. The line parameters W and b are used here. The previous equation returns a value of 1 for examples in the +ve class and a value of -1 for examples in the -ve class.

The hyperplane is defined by determining the best values for w (weights) and b (intercept). The cost function is minimized to find these best values. The SVM model or the line function f(x) efficiently distinguishes the two classes once the algorithm accumulates these optimal values.

In a nutshell, the optimal hyperplane has equation w.x+b = 0. The left support vector has equation w.x+b=-1 and the right support vector has w.x+b=1.

Thus, the distance d between two parallel liens Ay = Bx + c1 and Ay = Bx + c2 is given by d = |C1–C2|/√A^2 + B^2. With this formula in place, we have the distance between the two support vectors as 2/||w||.

The cost function for SVM looks the like the equation below:

A picture containing text

Description automatically generated

The parameter in the cost function equation above denotes that a larger yields a wider margin, while a smaller yields a narrower margin. In addition, the cost function's gradient is determined, and the weights are adjusted in the direction of lowering the lost function.

**Algorithm for Non-linear SVM**

It is simple to create a linear hyper-plane between these two classes in the SVM classifier. However, if the data is not linearly separable, the SVM algorithm employs a technique known as the kernel trick.

The SVM kernel function translates a lower-dimensional input space to a higher-dimensional space. To put it another way, it turns a not separable problem into a separable problem. Complex data transformations are performed based on the labels or outputs that specify them.

To better understand data transformation, look at the diagram below. The left-hand set of data points is clearly not linearly separable. When a function is applied to a set of data points, however, we receive changed data points in a higher dimension that may be separated using a plane.

Chart

Description automatically generated

We need to add an extra dimension to separate non-linearly separable data items. Two dimensions, x and y, have been employed for linear data. We add a third dimension, say z, to these data points. Let z=x2 +y2 in the example below.

Chart, scatter chart

Description automatically generated

The sample space is transformed by the z function, or increased dimensionality, and the above image becomes as follows:

A picture containing scatter chart

Description automatically generated

Close examination reveals that the data points above can be split using a straight-line function that is either parallel to the x axis or inclined at an angle. Linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid kernel functions are all included.

**RBF Kernel**

Due to its resemblance to the Gaussian distribution, RBF kernels are the most generic form of kernelization and one of the most extensively used kernels. For two points X1 and X2, the RBF kernel function computes their similarity, or how near they are to one other. This kernel can be expressed mathematically as follows:

Text, letter

Description automatically generated

where,

1. ‘σ’ is the variance and our hyperparameter

2. ||X₁ - X₂|| is the Euclidean (L₂-norm) Distance between two points X₁ and X₂

Let d₁₂ be the distance between the two points X₁ and X₂, we can now represent d₁₂ as follows:

Chart, scatter chart

Description automatically generated

The kernel equation can be re-written as follows:

Text

Description automatically generated

The maximum value that the RBF kernel can be is 1 and occurs when d₁₂ is 0 which is when the points are the same, i.e. X₁ = X₂.

* When the points are the same, there is no distance between them and therefore they are extremely similar
* When the points are separated by a large distance, then the kernel value is less than 1 and close to 0 which would mean that the points are dissimilar

Because we can see that as the distance between the points increases, they become less similar, distance can be regarded of as an analogue to dissimilarity.

Text

Description automatically generated with medium confidence

It is critical to determine the appropriate value of ‘' in order to determine which points should be regarded comparable, and this may be proven case by case.

**a] σ = 1**

When σ = 1, σ² = 1 and the RBF kernel’s mathematical equation will be as follows:

Diagram, text

Description automatically generated

The curve for this equation is shown below, and we can see that the RBF Kernel reduces exponentially as the distance rises, and is 0 for distances larger than 4.

Chart, histogram

Description automatically generated

* We can notice that when d₁₂ = 0, the similarity is 1 and as d₁₂ increases beyond 4 units, the similarity is 0
* From the graph, we see that if the distance is below 4, the points can be considered similar and if the distance is greater than 4 then the points are dissimilar

**b] σ = 0.1**

When σ = 0.1, σ² = 0.01 and the RBF kernel’s mathematical equation will be as follows:

Diagram

Description automatically generated with medium confidence

The width of the Region of Similarity is minimal for σ = 0.1 and hence, only if points are extremely close they are considered similar.

Chart, line chart

Description automatically generated

* The width of the curve is large
* The points are considered similar for distances up to 10 units and beyond 10 units they are dissimilar

## **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**

Diagram

Description automatically generated

**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**

Diagram

Description automatically generated

**Figure 4.7 Use Case Diagram**

### **4.2.4 System Flowcharts**

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

**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**