Republic of Iraq
Ministry of Higher Education and Scientific Research
University of Babylon
College of Information Technology
Department of Software

Detecting Error in Software Engineering to Enhance Testing Task Based on Decision Tree Algorithm: A Case study of Android Applications

A Thesis

Submitted to the Council of the College of Information Technology for Postgraduate Studies of University of Babylon in Partial Fulfillment of the Requirements for the Degree of Master in Information Technology-Software

By
Raed Kazem Mohsen

Supervised by
Asst. Prof. Dr. Ahmed Saleem Abbas

2020 A.D. **1441 A.H.**

Supervisor Certification

I certify that the thesis entitled "Detecting Error in Software Engineering to Enhance Testing Task Based on Decision Tree Algorithm: A Case study of Android Applications" was prepared under my supervision at the department of Software/ College of Information Technology/ University of Babylon as partial fulfillment of the requirements of the degree of Master in Information Technology-Software.

Signature:

Supervisor Name: **Asst. Prof. Dr. Ahmed Saleem Abbas**

Date: /   /2020

The Head of the Department Certification

In view of the available recommendations, I forward the thesis entitled "**Detecting Error in Software Engineering to Enhance Testing Task Based on Decision Tree Algorithm: A Case study of Android Applications**" for debate by the examination committee.

Signature:
Asst. Prof. Dr. Ahmed Saleem Abbas
Title: Assistant Professor
Head of Software Department
Date: /   /2020

Certification of the Examination Committee

We, the undersigned, certify that (Raed Kazem Mohsen) candidate for the degree of Master in Information Technology-Software, has presented his thesis of the following title (**Detecting Error in Software Engineering to Enhance Testing Task Based on Decision Tree Algorithm: A Case study of Android Applications**) as it appears on the title page and front cover of the thesis that the said thesis is acceptable in form and content and displays a satisfactory knowledge of the field of study as demonstrated by the candidate through an oral examination held on March 17, 2020 .

Signature:
Name: Wafaa Mohammed Saeed Hamzah
Title: Assistant Professor
Date: / / 2020
(Chairman)

Signature:
Name: Ayad Rodhan Abbas
Title: Assistant Professor
Date: / / 2020
(Member)

Signature:
Name: Mohannad M.AL-Yasiry
Title: Senior lecturer
Date: / / 2020
(Member)

Signature:
Name: Ahmed Saleem Abbas
Title: Assistant Professor
Date: / / 2020
(Member) // Main supervisor

Signature:
Name: **Hussein Atiya Lafta**
Title: Professor **Prof. Dr.**
Date: / / 2020
(**Dean of Collage of Information Technology**)

Dedication

Praise be to God Almighty. Then my parents. My mother, who drank the empty cup to give me a drop of love, my father who harvested thorns from my path to pave me the path of knowledge, thank you for all their efforts from my birth to these moments. You are everything in my life, I love you. I also thank my brothers who have endured my absence from them in most of the beautiful moments, and I thank my sisters who helped me a lot to overcome the difficulties, and last but not least my wife who has always encouraged me and endured my busyness and my constant absences due to the study. Thank you from the heart my love. I am also pleased to extend my thanks to everyone who advised me, directed me, or contributed with me in preparing this research. In particular, I thank the supervisor, rescuer, brother and friend, Assistant Professor Dr. Ahmed Saleem Abbas Al-Saffar for supporting me in the darkest of circumstances and guiding me with advice and choosing the title and subject, and I thank the Dean of the College of Technology Information, and all the staff of the Faculty of Information Technology at the University of Babylon. thank you all

Acknowledgement

Thanks to the team who collected and created the data set for their hard work and published their data set for research and scientific studies. I also thank my teachers and colleagues at the University of Babylon - College of Information Technology for their guidance and continued assistance to me. I also thank Google for providing and facilitating access to Android app issues and facilitating the conclusion of this study.

Abstract

There are many challenges that affect the success of the application. Especially when the application is marketed, or after performing maintenance and development processes, because of the rapid development in the field of application services and mobile phones and their uses. It became necessary to speed up the creation of applications compatible with the new requirements. At present, most ministries and service institutions have applications in stores, and provide their services through these applications. This made software engineers quickly innovate and develop applications to satisfy users. On the other hand, the development and construction process may include many unintended errors that may lead to a defect or problem in the operation of the application, which is difficult to discover if the application is complex. Therefore, it is imperative to focus on some ways to accomplish software engineering tasks quickly.

The proposed system consists of four stages, a Pre-Processing stage of Dataset reviews, Pre-Processing stage of new reviews and star rating, Decision Tree classifier Model and Similarity measuring and building vector. By using decision tree classification to classifying user reviews about applications into simple and understandable categories, to identify Android applications that contain errors, diagnose the features that users want in applications, and diagnose the largest possible number of errors, to be presented to software engineers. This Thesis performs maintenance and add features required in future applications and Contributes to reducing the efforts of software engineers and speeding up building good and reliable applications in short time and less cost. The accuracy of the model has been verified and found to work with an accuracy of 92% to 93%

Publications

Some of the works presented in this thesis has published as listed below:-

- The First paper “Detect and diagnose Code Smell types by Using the Backpropagation Neural Network based on user feedback” was **published** using the Backpropagation of the neural network in the IOP Journal of Physics in April 2020.(Online ISSN: 1742-6596, Print ISSN: 1742-6588, **SCOPUS**).
- The second paper “A model for diagnosing the largest number of Android application problems, based on reviews in download stores by use of the decision tree” was **published** using the Decision tree algorithm in Journal of Advanced Research in Dynamical and Control Systems (JARDACS)-(ISSN: 1943-023X **SCOPUS**) in Vol. 12, No. 4, 2020

Declaration

I hereby declare that this dissertation " **Detecting Error in Software Engineering to Enhance Testing Task Based on Decision Tree Algorithm: A Case study of Android Applications** " submitted to University of Babylon in partial fulfilment of requirements for the degree of Master of Software in Information Technology \ Software, has not been submitted as an exercise for a similar degree at any other University. I also certify that this work described here is entirely my own except for experts and summaries whose source are appropriately cited in the references.

Signature: [illegible]

Name: Raed Kazem Mohsen

Date: 12/9/2020

Table of Contents

List of Tables

List of Figures

List of Appendices

List of Abbreviations

n	abbreviation	Full name
1.	APD	Access to Public Data
2.	ASM	Attribute Selection Measures
3.	BLOB	Blob Class
4.	BMAP	Bitmaps
5.	BSMC	Bad Smell Method Calls
6.	CBO	Coupling Between Objects
7.	CC	Complex Class
8.	CCy	Cyclamate Complexity
9.	Deep .L	Deep Learning
10.	DIT	Depth of Inheritance Tree
11.	GPS	Number of GPS Uses
12.	I/O	File I/O
13.	IGS	Internal Getter/Setter
14.	iOS	iPhone Operating system
15.	IPM	Instructions per Method
16.	LCOM	Lack of Cohesion in Methods
17.	LIC	Leaking Inner Class
18.	LM	Long Method
19.	LOCL	Number of Location Listeners
20.	Machine .L	Machine Learning
21.	MIM	Member "Ignoring Method
22.	NBI	Number of Byte- code Instructions
23.	NET	Networking
24.	NLMR	No Low Memory Resolver
25.	NLP	natural language processing
26.	NOC	Number of Classes
27.	NOCH	Number of Children
28.	NOM	Number of Methods
29.	NTO	Network Timeouts
30.	PPIV	Percent Public Instance Variables
31.	SAK	Swiss Army Knife
32.	Software E.M	Software engineering .methods
33.	SQL	SQLite
34.	SURF	Summarizer of User Reviews Feedback
35.	WKL	Wake Locks with no timeout
36.	WMC	Weighted Methods per Class
37.	XML	XML Parsers

Chapter one

Introduction

CHAPTER ONE
INTRODUCTION

1.1 Overview

Software engineers seek to build and development the applications that help provide maximum service to users. This includes all categories of different applications (health, news, games, movies, etc.). Because of the rapid development of information technology, mobile phones and increased user requirements, it has become necessary to create, develop and maintain applications to meet the growing demands of users with the size of development and keep abreast of rapid developments [1]. Where, the Mobile applications play a pivotal role in everyday life because a lot of customers use it every day for social networking, emergency and many other uses and at any time and any place [2],[3]. That mobile applications are software applications running on Well-known and certified systems such as Android, iOS (iPhone Operating system), etc., which are downloaded to the portable device or smartphone by the user by download store [4]. As is well known, download stores are growing rapidly in terms of the number and diversity of applications and the number of downloads for them [5]. Increasing and disparate users' requirements have made software engineers They are quick to build and develop the apps quickly and under pressure, to get an application that can perform many tasks and functions as well as include applications to work in systems of all kinds to cover their requirements and keep abreast of developments [6] . This may be accompanied by unintended errors that are difficult to detect at times, which may lead to a malfunction in the application quality and a decrease in the success rate, in addition to a breach in the level of revenue returns, so most applications need improvement and maintenance periodically [7],[8]. Not to mention the possibility of post-maintenance errors or development [9],[10]. Sometimes a software engineers misunderstanding of the problem in the application may lead to maintenance more than once. Where the maintenance process requires several studies to determine the standards of maintenance and diagnosis of the problem in the application to get the best performance and provide the maximum effort and time and cost . Therefore, it is necessary for Software Engineers to understand the required features and to add new features [11]. Maintenance is often difficult and costly, due to the lack of clear guidelines for maintenance [12]. So, it has become necessary to search for the best ways and means that contribute to the success of the application, planning and conducting studies in order to obtain the best application performance [13],[14]. The commitment to software engineering tasks helps to achieve the best results and leads to the provision of software engineers' efforts in building applications. On the other hand, software engineering processes may be very complex, costly and time-consuming. Such as the design, implementation, maintenance, testing and reuse of software applications. In addition, software engineering tasks require many skilled IT professionals to create millions of lines of code that must be installed, configured, and maintained. According to **Kephart** (2005), in the near future , it will be very difficult to manage IT environments, even for the most skilled IT engineers[15]. This requires a great effort by software engineers to understand the requirements and provide what users want in their applications [16]. Where the method of software engineer in building, development and maintenance of applications is a factor in the success or failure of the application. Therefore, the successful developer is looking for the best ways and means to contribute to the development and success of application evaluation. As well as finding factors that help to build, develop and maintain applications quickly and with the least errors. That the most successful apps are those that provide, above all, the features that users need [17]. In recent years, many studies have been conducted on the possibility of finding ways and tools to help software engineers build, develop and maintain applications. In addition, focus on the tools and standards that contribute to the chances of successful application and increase evaluation in application stores, and try to study and develop these standards to take advantage of them. Such as code quality standards, star ratings, number of download and usage, and many other criteria [18]. and Another studies on user reactions after maintenance [19]. as well as attempts to find quick ways to understand and provide user requirements [20]. It is therefore necessary to consider ways to speed up software engineering tasks and reduce the efforts of software engineers to understand what users want and take advantage of these methods in building and developing applications. Some studies have suggested that machine learning methods can be used to implement software engineering processes for the purpose of achieving goals and providing efforts [21],[22]. As well as making use of user feedback and categorization of intent and topics in application evaluation [23]. User reviews are an important source of valuable information about applications because they reflect real consumer certification [24]. This study deals with some of the main functions of software engineering with the possibility of implementing them through machine learning algorithms, as well as focusing on classified user reviews, using them to diagnose applications that have problems in their functions, and trying to benefit from them in evaluating applications and detecting the largest amount of performance problems in them and reducing recurrence Maintenance. Where user reviews classified as a main criterion will be dealt with in the evaluation of the application.

1.2 Problem Statements

There are many factors that affect an app's success rate, such as security, accuracy, speed of performance, cost, and number of downloads, and more. The success of the application depends greatly on the features available to the users. Due to the development in the field of applications and "mobile phones" and the ease of access and use around the world, this has led to the need to create applications that are "compatible" with the requirements. A design flaw or an application problem that is difficult to detect may accompany this. Especially if the application is complex, which makes maintenance difficult. Therefore, it is imperative to find quick ways to assist with maintenance, troubleshooting, and application development.

1.3 Literature Review

In this section, some of previous studies which are related to our study. These studies include the following theses and papers: -

- In (2016). Andrea Di Sorbo [25]. They used a new method called "SURF" to condense the massive amount of comments app developers must read. This "SURF" model based on a "conceptual model" to capture user needs that are beneficial to developers who perform maintenance and development tasks. It then uses sophisticated summarizing techniques to summarize thousands of reviews and create a structured and intense interactive agenda for recommended changes to the program. The results show the high accuracy of SURF in summarizing the Reviews, helping developers better understand user needs, and reducing the "time" required by "Developers" to manually analyze user requests and plan for future software changes.
- In (2016). Lorenzo Villarroel [26]. This study introduced the CLAP model, a tool that supports mobile app release planning activity by extracting information from user reviews. Such as reporting errors, grouping relevant reviews together (for example, all reviews reporting the same error), and automatically prioritizing review groups to be executed when planning the next app version. In fact, the proposed model achieved good results with accuracy of (86%).
- In (2016). Romain Rouvoy This study covers the individual and combined effects of the three Android performance code smell (Internal Getter / Setter, Ignore Members, and Hash Map Usage)[27].

This is done by using a tool called Paprika to detect the smell of the three icons in the applications and deriving four versions of these applications by correcting each detected odor independently and evaluating the performance of each version separately. Therefore, that is evaluate the UI and memory performance using metrics like frame time, number of delayed frames, memory usage, and number of garbage collection calls.

- In (2016). Lili Wei [28]. In this study A technique called" **FicFinder**" was proposed to automatically detect issues. This technology analyzes Android application code, detects potential compatibility issues, and communicates developers. The model was tested and was able to obtain good results as well as detect unknown problems. This paper focused on the study of "191" problem collected from five Android applications only. This requires the introduction of as many applications as possible in the experiment. For getting good results.
- In (2017). Adelina Ciurumelea [29]. This paper focused on the importance of monitoring and analyzing user reviews about applications in download platforms. A total of '1566' user reviews were collected in download stores for 39 applications and analyzed the text of these reviews manually, in addition to determining a high- and low-level rating containing specific categories of mobile phone such as battery, performance, memory, privacy, etc. The study summarized how to build a prototype for classifying user reviews in download platforms using machine-learning methods, where the proposed model obtained good results.

- In (2017). Joydeep Mitra [30]. In these papers, they suggested a study was conducted on the possibility of developing criteria to evaluate the gap detection tools that help develop safe Android applications. Where the Mobile devices security affects the security of their users. Mobile applications have become central to our lives, as they have access to vast amounts of sensitive and personal data to enable different banking, shopping, social networking, etc. Thus, the security of mobile devices and applications is critical to ensure the security and safety of its users. But Android apps still have vulnerabilities [31],[32]. Where it is difficult to create applications without the existence of security holes resulting from lack of knowledge of the developer known gaps, even if tested by the tools gap detection. In these studies, repository containing some application gap standards was created and named "**Ghera**" , which has 25 known vulnerabilities. This has improved the assessment of tools and technologies for detecting gaps and helping secure Android applications. This study was limited to certain criteria only, there are many necessary criteria that have not been taken into consideration such as (security standards arising from APIs including system, storage, web, network, camera, etc.), and these additions will make the repository more comprehensive.

- In (2018) Ossi Taipale [33]. Study aimed at systematically collecting and analyzing a number of maintenance measures for applications, Therefore, study focused on the most important tools and standards used in the application maintenance process, in which the system architecture is designed and implemented that can be used to collect fixed metrics in addition to the application run time, for running a system to analyze and visualize maintenance measures for the application.
- In (2018) Hanyang Hu [34]. Is presented a study that dealt with a Star ratings and user feedback on download platforms so that star rating analysis and application reviews provided unique insight to app developers on the quality of their apps. Are studied with 34,258-star ratings from nineteen apps across different platforms collected and manually analyzed. Evaluating apps and programs based on star ratings and user reviews about apps is crucial and greatly contributes to app success. The study discussed the negative impact of the types of complaints about these applications, and at times found that users had higher expectations for methodologies, techniques, and platform.
- In (2018) Dario Di Nucci1 [35]. is presented a study that dealt with a method of using automated learning models in detecting errors of the type "Smile code", where in spite of the good performance shown by code filters and error detection measures, there are some limitations that threaten the adoption of these measures in the
- detection of errors [36]. The study included thinking and finding ways to help software engineers and developers discover application errors and problems. This is done by processing a dataset of part of its

elements containing errors of type "Smile code". The study summarized the need to conduct an assessment on the impact of a number of factors on the accuracy of the results, such as (1) the size of the data set, (2) the choice of features and (3) verification methodology.

- In (2018) V. Guna [37]. Presented a study that dealt with issue of cloning the application code during the application development process, where application developers have the opportunity to copy parts of the main code and paste into the code of the application to be developed. This process is call code cloning. Using this method may result in duplicate copies of the code in the application , and will greatly affect if a part of the code contains errors. Thus, the detection of code cloning (duplicate code and error codes) is important to reduce maintenance costs. It also helps to identify the structure and design of the application in case of future development. The researchers suggested using a token-based comparison technique to detect code cloning . It can also be used to detect code theft in computer lab programs for students.

A table containing previous and similar works for project according to Table 1.1

Table 1.1 Previous work and similar to the project

n	Paper number in Reference	The Problem	Technologies used	Data Set	Results
1	[25]	This paper focuses on the need to understand user feedback about apps in the Apple Store and Google Play, and use them to capture the useful user needs of developers who carry out maintenance and development tasks.	NLP classifier to automatically assign a sentence in a review to one or more concepts. In addition, stemmed each sentence using the Snowball Stemmer Algorithm, to reduce words to their root form. Thus, built the classifier on top of the concept-related dictionaries to summarize thousands of reviews and create a structured and intense interactive agenda for recommended changes to the program.	Experiment_I_ results and Experiment_ II_ results	(93.75%)
2	[26]	The study discussed how to process, understand and classify requests submitted in user reviews, based on the information contained therein and use it to increase the app's evaluation.	the CLAP model, a tool that supports the mobile app version planning activity by extracting information from user reviews. and automatically prioritizing review groups to be executed when planning the next app release ,it use the **DBSCAN clustering algorithm and Random Forest machine learning algorithm**	WSDM_2015	86%.

3	[27]	a study that would help include the experience of evaluating the performance of Android applications after maintenance and development processes, and calculating accuracy with a set of metrics for the purpose of evaluating the accuracy of the application's performance	It used some of UI metrics like the frame time and the number of delayed and memory usage and the number of garbage collection calls to determine the impact on the memory, because they are known to be related to the user experience.	paprika_csv_merge_complete	MIM performs very well concerning the UI metrics in one of the studied app with 12.4% less delayed frames,
4	[28]	Understand Compatibility Issues At Code Level and Detect Compatibility Issues, to find methods and tools to help developers diagnose compatibility issues that are caused by fragmentation in the Android system.	In this study, a technique called Fragmentation-Induced Compatibility Issue Finder "**FicFinder**" proposed to automatically detect problems. This technology analyzes Android application code, detects potential compatibility issues, and contacts developers.	empirical_dataset_2.0	(90, 2%)

5	[29]	Monitoring and analyzing user opinions about apps on download platforms	User Request Referencer (**URR**) has been used and it is a approach that is able to organize reviews according to maintenance and development tasks	The dataset is a number of Android applications	(79%)
6	[30]	Evaluating vulnerability detection tools in Android applications.	A repository has been created named it **Ghera**, which has 25 known vulnerabilities. This improved the evaluation of tools and techniques to discover vulnerabilities and help secure Android applications.	The dataset is a number of Android applications	---

7	[33]	Facilitate the systematic collection and analysis of maintenance metrics.	System architecture is collect both fixed metrics (maintenance metrics and analysis system) and runtime.	DOC-REQUIREMENTS	---
8	[34]	star rating analysis and app reviews about apps	No technique was used, but 34,258 reviews were manually checked	snapshot_s1_android_tagged	68%

9	[35]	detecting errors of the type "Code Smile "	Machine-learning Techniques (Grid-search algorithm)	MLD_Results. csv	90%
10	[37]	Addressing errors associated with the issue of cloning application code during the application development process.	back propagation and multi objective genetic algorithm based on neural network	The dataset is a number of Android applications	---

1.4 The Aim of Thesis

The main objective to Design a form that deals with user feedback on the performance of Android applications to contribute to detecting errors and diagnosing the largest number of problems in the performance of applications. To help software engineers to understand the problem in the application in addition to contributing to improving the performance of the application by implementing some software engineering tasks to reduce the effort, time and cost in building, developing and maintaining applications.

1.5 Thesis Organization

This Thesis Consists Four Chapters in the addition Chapter one as the following: -

Chapter Two: - It contain Theoretical Background about important topics addressed in this thesis, such as Concepts of **Software Engineering**, **Machine learning**, **Decision tree**, **Code Smells** and natural language processing (**NLP**).

Chapter Three: - This chapter contains an overview of the proposed model and how to build it through the use of various artificial intelligence techniques and algorithms.

Chapter Four: - includes the results obtained by the algorithms and techniques.

Chapter Five: - Conclusion and future work for the purpose of developing work and obtaining more accurate and rapid results.

Chapter two

Theoretical Background

CHAPTER TWO
THEORETICAL BACKGROUND

2-1 Overview

Theoretical background is presented of all the concepts and techniques used in this thesis, which included six main topics: Software engineering, Machine learning algorithms, Decision Tree, Android, and the Code Smell, in addition to Text processing techniques. In addition, all parts of the data set used in our project will be explained

2-2 Software Engineering

Software engineering plays an important role in the process of building programs, as it provides the fundamental concepts, methods and tools for producing reliable and high-quality software [38]. The main tasks of software engineering are requirements, coding, and communication, Development, Design, Modeling, Planning and Testing. Software engineering is not limited to focusing on about to create simple programs, but in fact represent the complex program life cycle. Some efforts have been made to define software engineering as a profession and to define the boundaries of this emerging field of research. Where several definitions of software engineering have emerged since its first mention at the NATO Software Engineering Conference in 1968 and there is a lot of literature that has described software engineering and its functions, a good example is this, it is the practical application of scientific knowledge in the design and construction of computer programs and associated documents necessary for their development and maintenance. Another definition is the application of a disciplined and measurable approach to software development, operation and maintenance [39]. In addition to study all these approaches. Moreover, the application of engineering to software, where software development is a multifaceted task and often contains new challenges. However, software engineering work and the product that it produces remain on the edge of chaos [40]. Chaos Engineering is the field of distributed system testing in order to build confidence in the system's ability to withstand all turbulent production conditions [41] .The edge of chaos can be visualized as an unstable, partially structured state. All know that unexpected events must happen, as if we had an operating system distributed in production, and distributed systems contain many interacting components so that the number of things that can happen is large. Where hard disks can fail, the network can be disrupted, sudden increase in customer traffic can overload a functional component ... etc. These events often provoke undesirable behaviors. We will never be able to prevent all possible failure modes, but we can identify many system weaknesses and avoid and fix errors, prevent future problems and make the system more flexible. Chaos engineering is a way to experiment with infrastructure that highlights systemic weaknesses. This practical verification of the testing process leads to build flexible systems, and more confidence in the operating behavior of these systems. The challenges of educating software engineers are more than just programming [42]. In addition to this, the best software engineers measure their program characteristics to ensure that the requirements are consistent and integrated, whether the design is of high quality, and whether the code is ready for release [43]. In addition, the effective project managers measure the attributes of processes and products so they can know when tools will be ready for delivery and whether the budget will be exceeded. They include attention to detail, such as quality, schedule and economic goals. In addition, focusing on ways to practice software engineering helps professionals respond to non-technical problems, including management, communication and all other fields. As a result, software engineers must engage in the experience of professional software engineering practices in order to understand useful practices and techniques in different situations [44]. Moreover, find a good balance between theory and practice. Therefore, it is necessary researchers is making efforts to develop and implement new ways to support a comprehensive software engineering approach [45] ,[46] ,[47].

2-2-1 Software Engineering Models: Plan-driven and Agile Method

Conventional IT system development organizations selectively utilize two techniques [48]. The plan-driven method in which requirements are defined and base-lined in the initial phase of the project [49]. And the agile method which is based on iterative, incremental development of the project scope [50],[51]. Figure2.1. the plan-driven methods, additionally referred to as the Waterfall method, is AN IT system development method that consists of sequent phases. The stages in the plan driven approach include some steps such as requirements analysis, design, implementation, testing and commissioning. Where the basic requirements are defined before design and implementation to start using the integrated change control process. Thus, after completing the requirements in the initial stage, customer participation is limited. Whereas, with agile methods, are re-planned frequently even during the implementation phase. With the possibility of collecting requirements early in the project. At the end of that, the program building team should repeatedly prioritize detailed requirements. And conduct studies on the potential benefit of using and balancing the above two approaches, and the rapid response to change in plans [52] ,[53] .

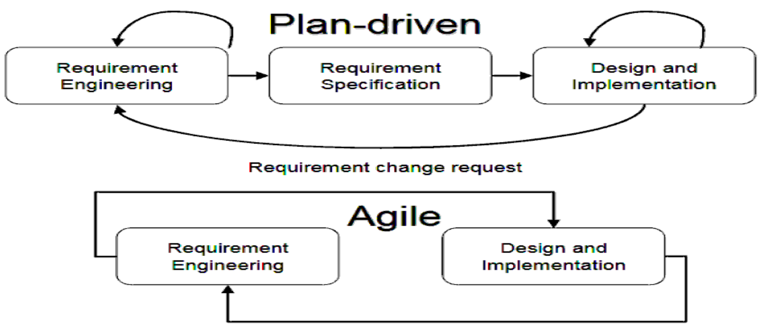

Plan-driven
Requirement Engineering
Requirement Specification
Design and Implementation
Requirement change request
Agile
Requirement Engineering
Design and Implementation

Figure 2.1: Plan-driven and Agile Method

2-2-2 Software Engineering Models

- **The Waterfall Model**: - is the oldest and the most well-known model [54]. Figure 2.1 this model is widely used in government projects and in many major companies, in which the requirements of the problem is usually well understood. The model allows for well-defined modifications or improvements to an existing system (e.g., an adaptation to new software careers that has been mandated because of changes to regulations), these increase development efforts. Where, one of the major disadvantages of this model is that major changes to the requirements cannot be adopted in the middle of the project [55]. Where if the application moves to the testing stage and one of the requirements is changed, it becomes difficult to return and change it. Another problem could be the prolonged delivery of the final product, but when requirements are well defined and reasonably stable all of this will be under control this model is characterized by its sequential steps. Which means that the waterfall model begins and ends one stage before starting the next one. Moreover, the important point is that the evolution stage of the waterfall model is the last stage. In addition, continuous program improvement was not a decision to engineer software. but was the result of increased user requirements [56]. It is better to use a different model if the project includes many changes, because the waterfall model does not address the change well, and is not cost-effective.

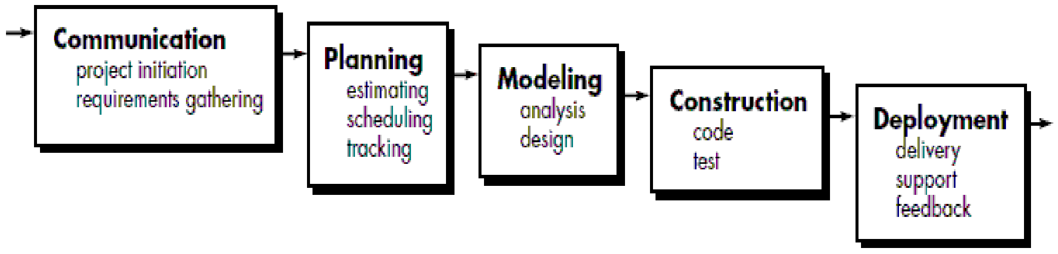


Figure 2.1 The Waterfall Model

- **Incremental Process Models**: - It is characterized by the achievement of the basic requirements and the possibility of adding complementary features that will be requested by the beneficiary during the design stages. Figure 2.3. Where the underlying product is used by the customer or subject to a detailed evaluation. In case of changing or adding features, a plan will be prepared for that purpose to better meet customer needs and provide additional features and functions. This process is repeated after each increase is delivered, until the full product is produced. An important aspect to consider is the relationship between progressive improvement and design changes, where a slight but important difference can be found in the objectives and degrees of freedom available to the design team compared to the progressive improvement team. The concept of progressive improvement carries the meaning that the design of the process has been developed and that standards have been developed for its operation. Where, optimizing the process indicates tracking the reasons for departing from the specific operating standards or investigating how the process works for better results that are close to achieving the original design goals. And provide integration to repair interactive and additional processes in the system [57].

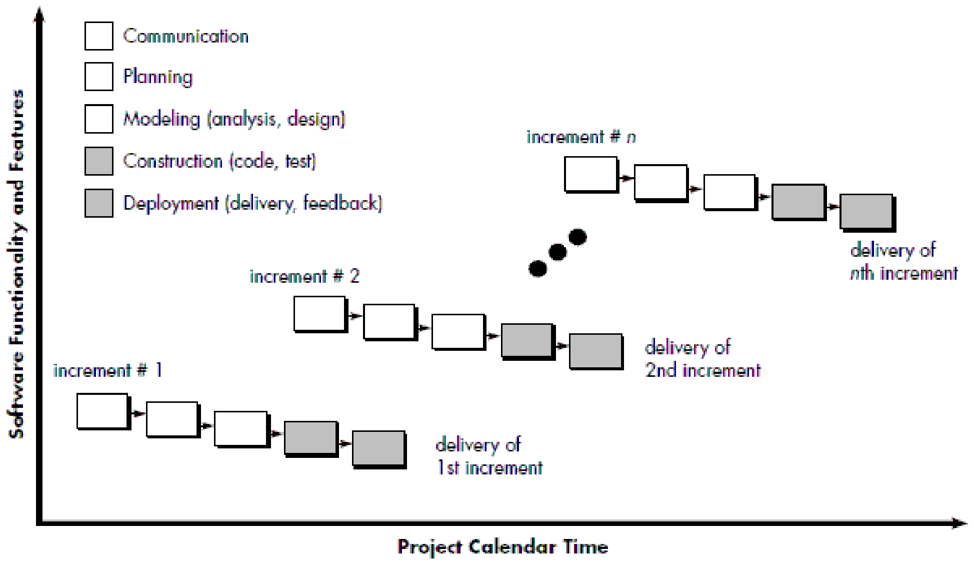

Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

Figure 2. 3 The Incremental Process Model

- **Evolutionary Process Models**: - Like all complex systems, software evolves over a period of time [58]. As time progresses, business and product requirements change, making the fixed-line path to the final product unknown. It makes time to complete the product difficult, but a limited version must be provided to counter the pressures of work or competition. Most projects require a model that accommodates the continuous development of the product. Therefore, a product that evolves over time must be introduced, as repeated models have the potential to develop more complete versions of the program [59].

There are two common models of the evolutionary process: -

1- The Spiral Model: - This model combines both the design and prototypes of the previous stages and the collection of concepts from top to bottom. And focus on assessing the risks in the project [60]. Figure 2.2

2- The Prototyping Model: - Often determined by general requirements, without explanation of functions and features. Figure 2.3

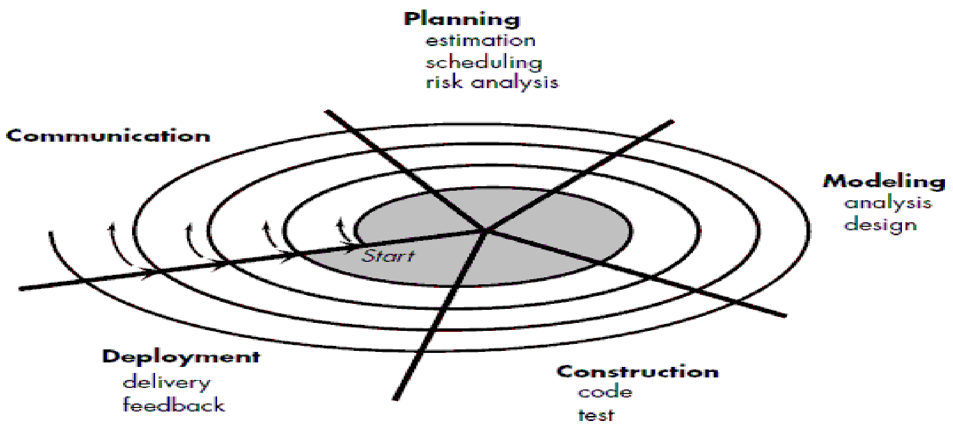


Figure 2.2 the Spiral Model

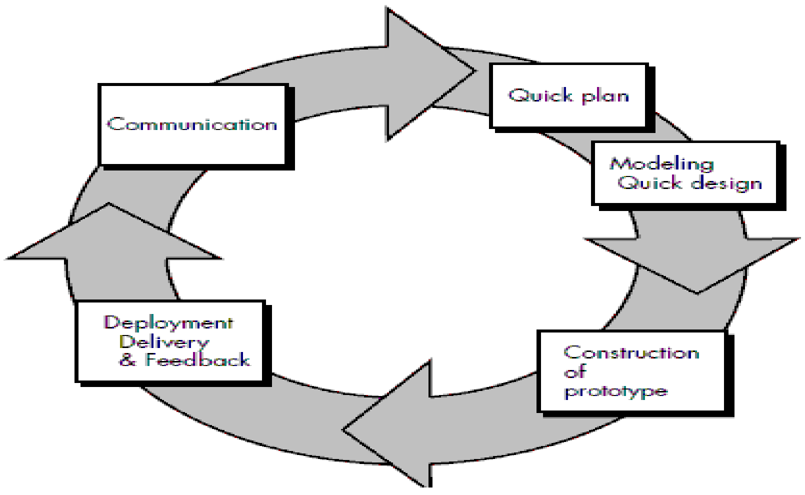


Figure 2.3 the Prototyping Model

- **Concurrent Models**: - The concurrent development model, sometimes called concurrent engineering, allows the software team to represent the simultaneous and simultaneous elements of any of the operations models by calling one or more of the software engineering procedures: prototyping, analysis, and design. To providing an accurate picture of the status of the project [59]. Figure 2.4

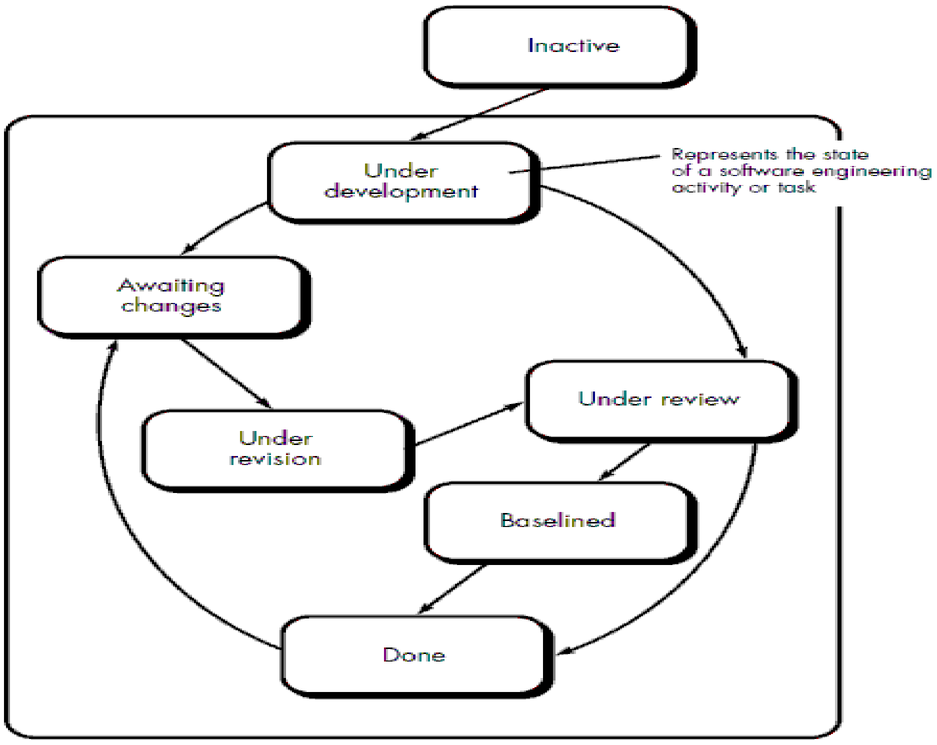


Figure 2.4 the Concurrent Model

2-2-3 Requirements Engineering

It is the process of understanding and the required services and determining the restrictions imposed on these services and create a solid foundation for design and construction. [59]. Figure 2.5. Engineering processes for the requirements ensure that the program will meet the expectations of the user and end with a high-quality program. Requirements engineering is a very important activity that can affect the entire software development process. At the beginning of the design, the requirements may not be clearly defined, it may be bound by a number of factors beyond its control, or it may be influenced by other goals. It also affects the development process. It is like embarking on a cruise without any idea of fate and without any navigation planner, the requirements provide the means to go to the specified destination and requirements allow risk management from the earliest possible point in development and impact evaluated. Thus, Requirements therefore form the basis for a project planning [61].

There are four activities for engineering requirements [62]. It is as follows: -

- **Feasibility study:** Determined using budgeted software technologies, etc. The feasibility study should be cheap and fast and must decide whether or not to proceed with the project.

- **Requirements elicitation and analysis**: It is a method of calculating system needs through monitoring, and this may include one event or many prototypes that will facilitate understanding of the system to be presented.

- **Requirements specification:** It is an activity that can include information such as user and system requirements.

- **Requirements validation:** It is a process of checking the requirements of consistency and completeness, where during this process errors are detected in order to be modified and get rid of problems.

Activities in the process of requirements are not simply carried out, but are carried out in strict sequence. Where the focus must be on knowledge of the needs in order for the system developer to obtain a clear view of the requirements of the evolving system [٦٣] .

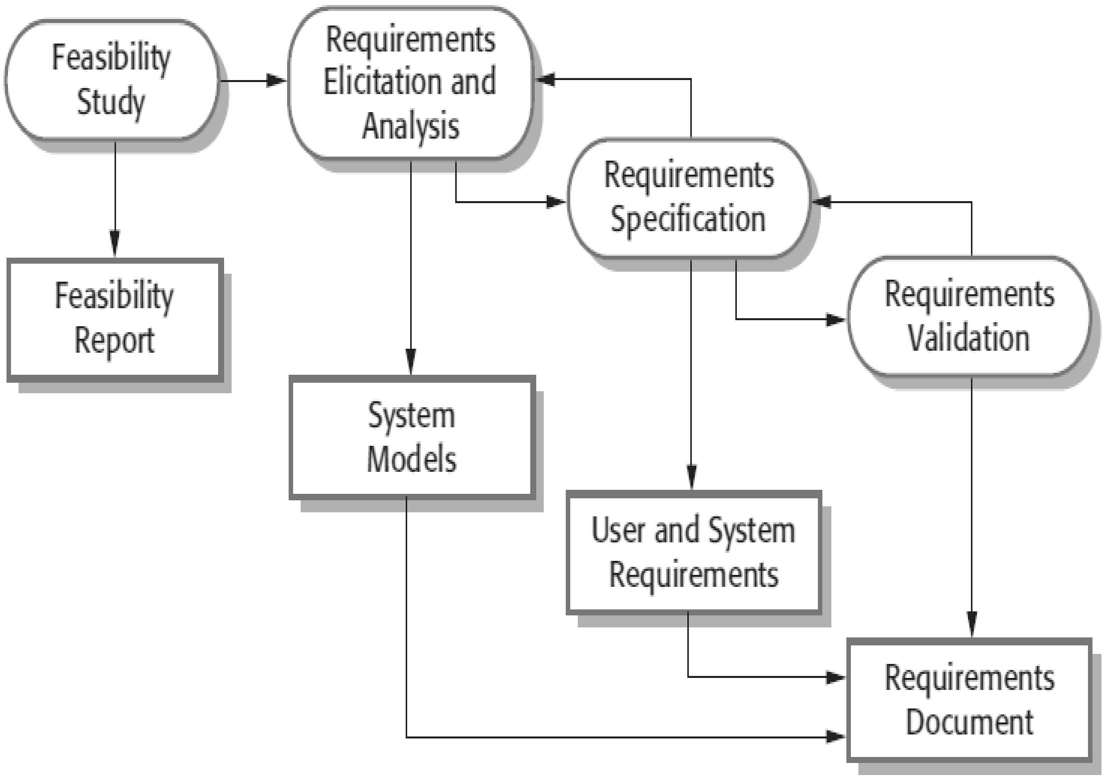


Figure 2.5 The requirements engineering process

Requirements Engineering is the process of finding out/discover, analyzing, documenting and checking the services and constraints [59], [64] .

There are two types of requirements: -

1- **User requirements**: Natural language data as well as graphs of services provided by the system and operational constraints.
2- **System requirements**: It is a document that outlines detailed descriptions of system functions, services and operational constraints in an orderly manner. Determines what needs to be done to be part of a contract between the client and the design and build administrator.

2-2-4 Software Design and Implementation

It is the process of converting a finished system specification into an executable system. Where a stepwise approach is used, it may also include an improvement in the program specifications [65]. The program design is a description of the structure of the program to be implemented, data models, and interfaces between system components. Where software designers frequently develop software design and add details during the development of the system.

Figure 2.6 is a sample design process that shows all the inputs, activities and documents that will be produced as output.

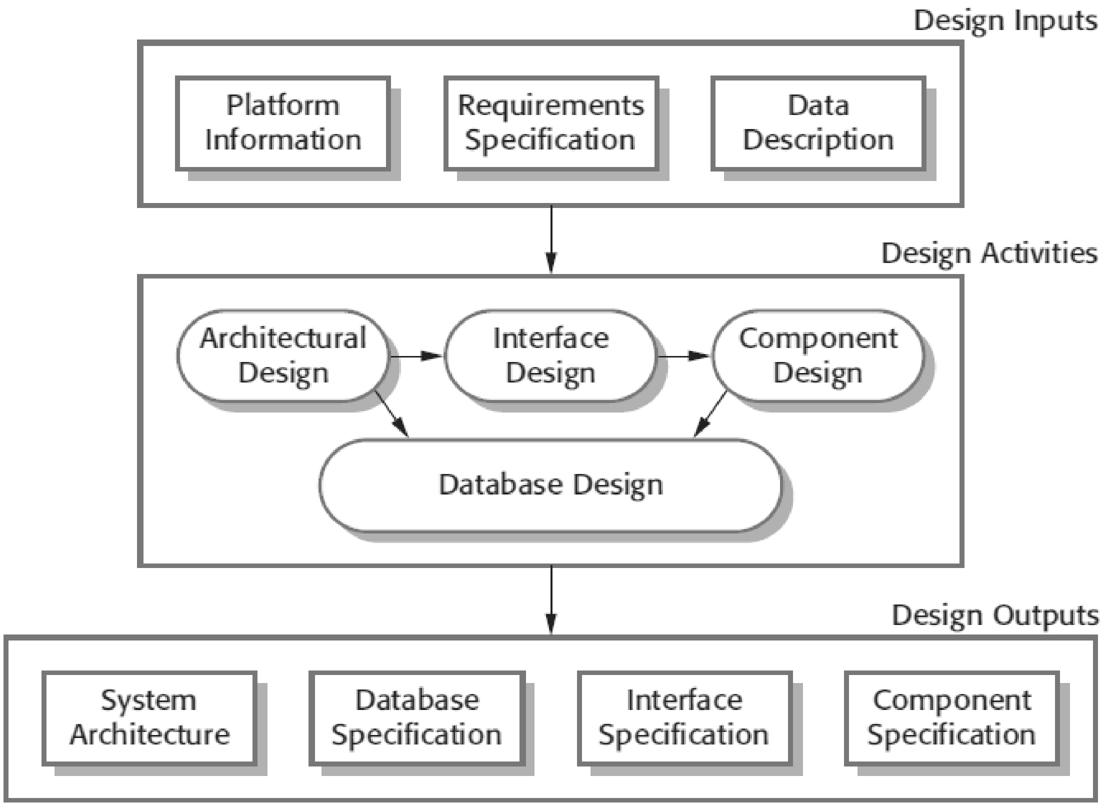


Figure 2.6 the software design process

There are four main activities in the design of the information systems parts Figure 2.7 [59].

As follows: -

- **Architectural design:** It is the general structure of the system, the main components of the system, and the relationships of components among them.
- **Interface design:** The interface specifications must be clear and components can be designed and developed simultaneously.
- **Component design**: Each component is taken with its own design and how to operate it and leaves the specific design of the programmer.
- **Database design:** This depends on whether to reuse an existing database or create a new database. So that the structure of the system data is determined and how it is represented in the system database

These activities lead to a number of different designs, with details varying depending on how the system is developed. Design documents must provide an accurate description of the system.

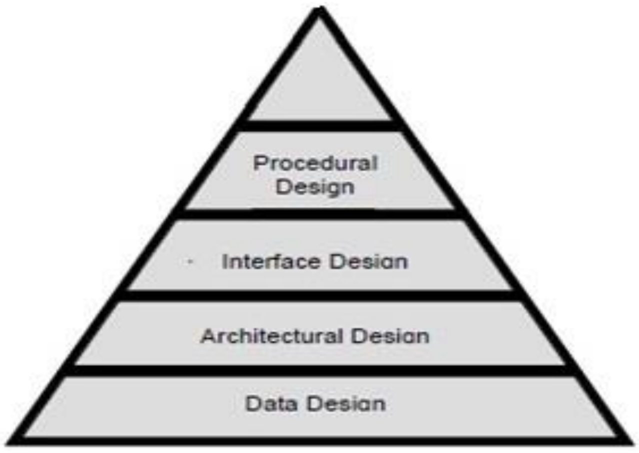


Figure 2.7 Design Model

2-2-5 Software Verification and Validation

The objective of validating the program is to ensure that the system conforms to its specifications and that it meets the clients expectations [59],[66].Figure 2.8 Validation can also involve every stage of code. Software testing strategies provide a way to test program design in a series of well-planned steps; can lead to a successful program. And provides a road map for testing [67].

There are four different types of software testing strategies: -

- **Unit testing**: - It means testing all system components independently, without other system components. Where, Unit is the smallest testable part.
- **Integration testing**: - At this stage, system components are integrated and search for errors resulting from the integration of parts and components of the program, as well as making sure of the system's readiness. There are two methods of processing, namely top-down integration and bottom-up integration
- **Acceptance / Validation Test**: - In this approach, the test is taken to check if the product has been developed according to detailed standards and criteria and meets all requirements set by the user. The acceptance test approach is also known as validation test, quality assurance test, final test, factory test acceptance test, etc. in software engineering. Sometimes the overlap between development and software testing is affected, so programmers tend to gradually test application code during development. Whereas, in plan-based software operations a system specification test is performed and tests may be performed at the same time as the

requirements writing stage before development begins. Figure 2.9 helps testers and developers understand and disclose requirements.

- **System testing**: - Here an integrated system test is conducted to assess the compatibility of the system with its specific requirements. The purpose of the system test is to assess the compatibility of the system with the specific requirements.

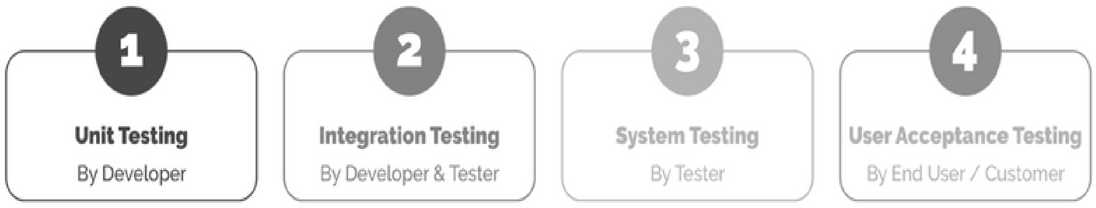


Figure 2.8 Levels of Testing

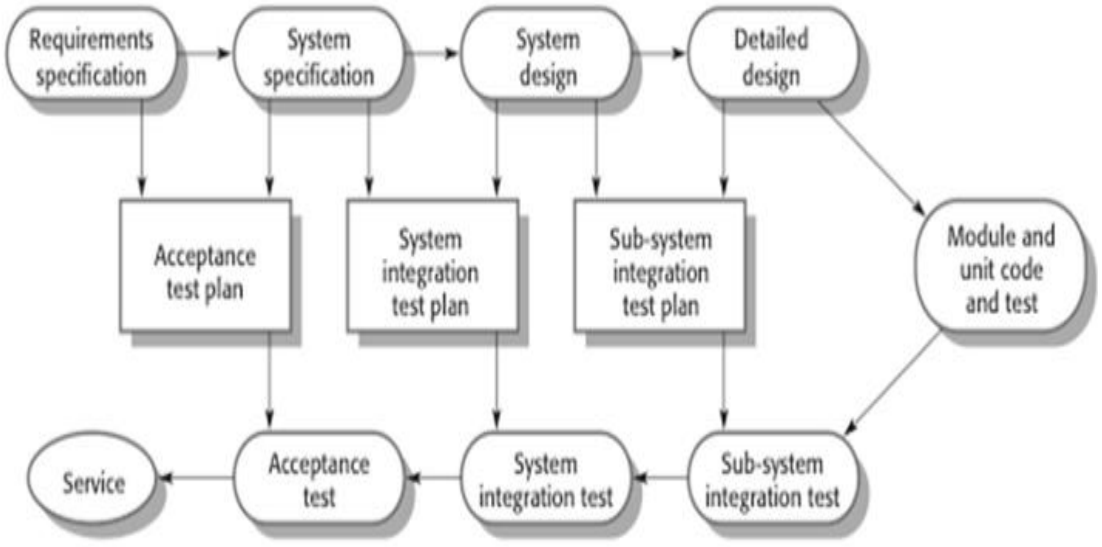

Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

Figure 2.9 Testing phases in a plan-driven software process

2-2-6 Software Maintenance

The requirements of users always change even after the completion of its system operating environment [68] . This is attributed to the reason for the continued evolution and the increased use of programs. The flexibility of software systems is therefore one of the main reasons why software is used in large, complex systems. it is logical to see evolution and maintenance as a continuum instead of two separate processes, it is also more realistic to think of software engineering as an evolutionary process.

2-3 Machine Learning

There are many books and studies that have explained the concepts of algorithms of machine learning types [69],[70]. the concept of machine learning is made is about making computers modify or adapt their actions whether these actions are making predictions, or controlling a robot to get results more accurate, and accuracy is measured by number of the chosen the correct actions, where This is achieved through the use of machine learning algorithms. Imagine playing a certain game against your computer. You can beat it every time at the beginning, but after you continue playing, the level starts to rise. The higher the level, the harder the game becomes, the game will continue to use the same strategies against other players. Scratch with each new player, this is a form of circularization. The intention is computer will learn how to play this game and how to defeat you and defeat players of the same level. Machine learning is distinguished by incorporating diverse ideas and sciences such as neuroscience, biology, statistics, mathematics and physics, as it uses algorithms to make computers learn. While algorithms are chains of instructions used to solve the problem, where Computer algorithms organize enormous amounts of data into information and services, based on specific instructions and rules, the work of learning algorithms is to create rules, not computer programmers. This approach allows learning from data rather than computer programming in every step. This means that smart computer applications can be used to perform complex tasks that cannot be programmed manually such as image recognition applications for the visually impaired. The thing in machine learning is data mining these are the methods and algorithms that look to extract useful information from big data sets. This is particularly important for the desire of software engineers to use automated learning methods in processing huge data. Therefore, highly complex algorithms used to train a large data set will be a problem. In addition, the complexity is often divided into two parts, the complexity of the training and the complexity of the experience of the results of the algorithm used. The basic process of machine learning is to give training data to a learning algorithm. The “learning algorithm” algorithm creates groups of rules based on the conclusions you got from the data, where this is referred to as the machine-learning model. Various training data is used. The same learning algorithm can be used to create different models and rules. For example, a computer can be taught how to translate languages or forecast the stock market using the same type of learning algorithm. Thus, deducing new instructions from the data is the primary force of machine learning. It highlights the role of influencing data. The more data available to train the algorithm, the more knowledge of the algorithm. In fact, there have not been many recent developments in artificial intelligence due to the radical innovations in learning algorithms, but rather because of the vast amount of data provided by the Internet. Figure 2.10 Therefore , the concept of machine learning is one of the most promising fields of research in the field of artificial intelligence, It has become a powerful tool and It has the ability to develop rapidly and in a Wide and scattered range applications.[71],[72].

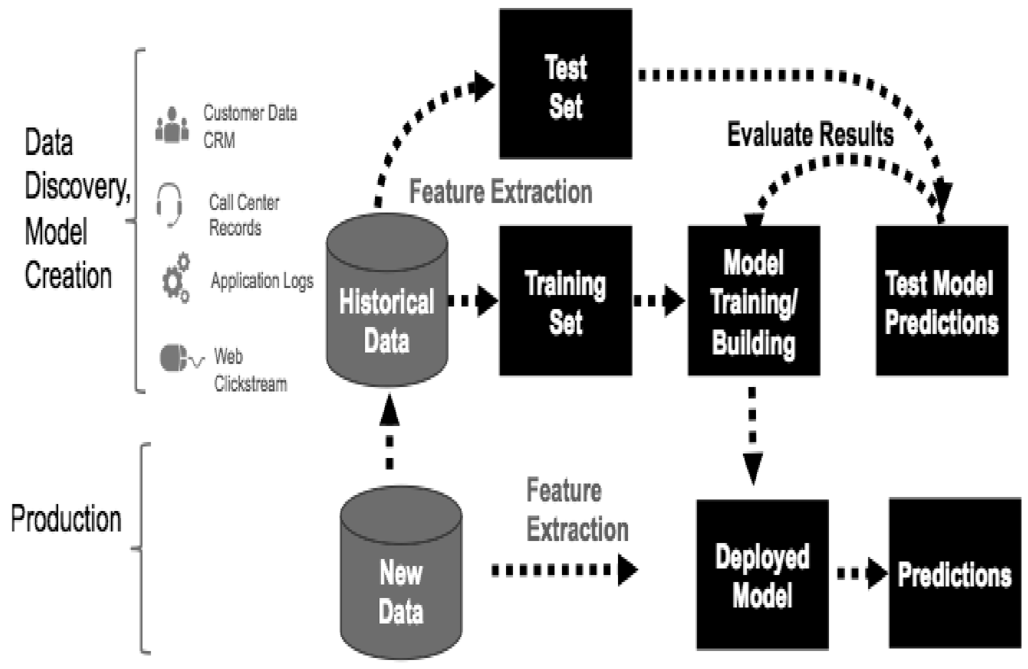


Figure 2.10 Machine Learning Training and testing

2-3-1 Machine .L and Deep .L

Machine learning is about designing algorithms that allow a computer to learn Computational analysis and performance of machine learning algorithms are a major and influential branch of statistics known as computational learning. Learning does not necessarily involve awareness, but learning is about creating statistical regularity or finding other patterns of data. Thus, many machine-learning algorithms will hardly be similar to humans in terms of handling the learning task. In addition, learning algorithms can encounter difficulty when experimenting with different environments [73]. An example is deep learning Deep learning refers to mathematical models consisting of multiple layers, where data representation is taught at multiple levels of abstraction. These tools and methods have been greatly improved with the latest technologies including speech recognition, visual recognition, discover drugs and genomics, as well as discover objects and many other fields [74],[75]. Deep learning detects the complex structure of large datasets, to indicate how the device changes its internal parameters that are used it for calculate the representation in each layer of representation layer (for previous layers). Figure 2.11. Deep torsion networks have brought about significant changes in image, video, speech and audio processing, while repetitive networks have highlighted sequential data like text and speech [76].

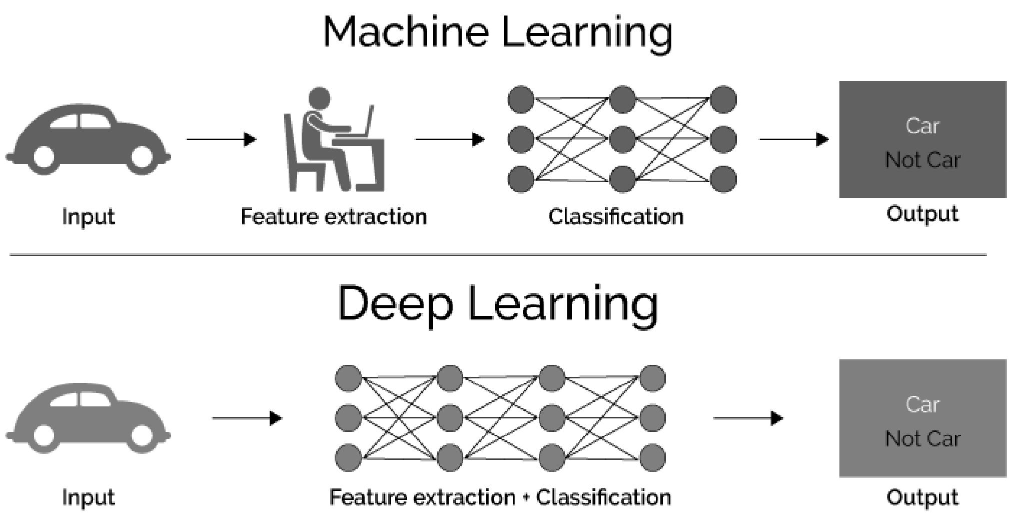


Figure 2.11 Machine Learning and Deep learning

2-3-2 Machine learning methods

Machine learning algorithms are organized into taxonomy based on the desired outcome of the algorithm [70],[77]. Figure 2.12 Common algorithm types include -

- **Supervised learning: -** Here, the algorithm creates a single function or set of functions for the purpose of planning and determining the required inputs and outputs. Since one standard version of the supervised learning task is the problem of classification, the learner must learn. Learning is a function that plans a vector in one of several chapters by looking at several examples of input and output of the function and training the algorithm on it.
- **Unsupervised learning:** - This type is characterized by helping to find diverse patterns and previously unknown in the data set.

- **Semi-supervised learning**: - This type aims to help find different and previously unknown patterns in the data set.
- **Reinforcement learning**: - Here the algorithm learns what to do (the algorithm learns how to behave). So that each procedure has some impact on the environment, the environment provides parameters to guide the learning algorithm.
- **Transduction**: - Does not create tasks explicitly. It tries to predict new outputs based on training inputs, training outcomes and new inputs. So, it's somewhat like supervised learning.
- **Learning to learn**: - The inductive bias algorithm is trained based on previous experiments.

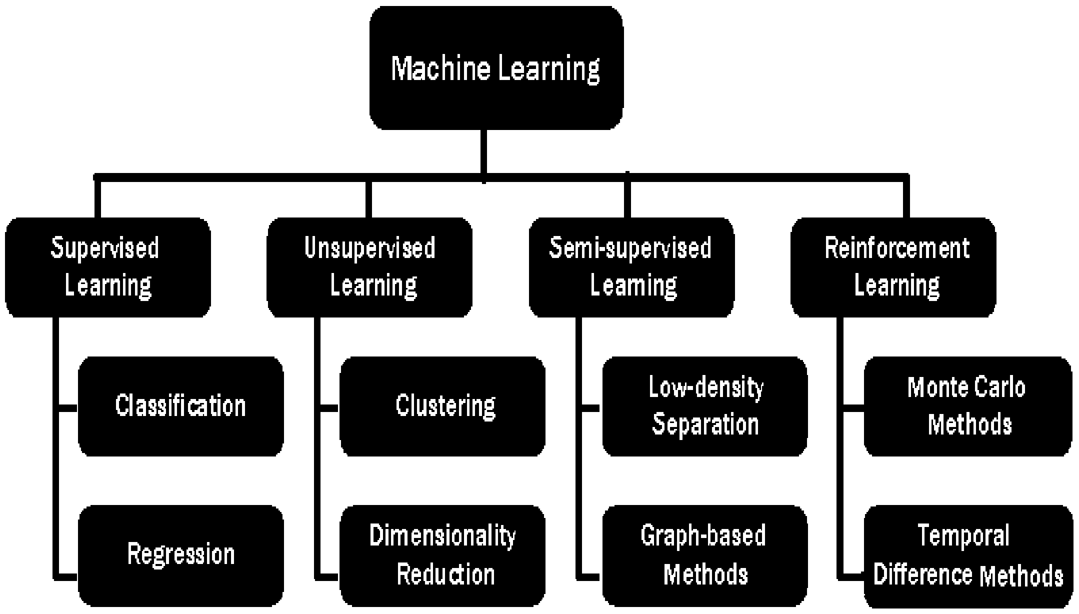


Figure 2.12 Machine Learning methods

2-4 Decision Tree

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (**terminal node**) holds a class label [78],[79]

2-4-1 Construction of Decision Tree:-

The Tree is Construction by splitting the source set into subsets based on an attribute value test. This process are repeated on each derived subset in a recursive manner called recursive partitioning, and it complete when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. See Algorithm 1

2-4-2 Decision Tree Representation:

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value. This process is then repeated for the sub tree rooted at the new node.

2-4-3 Challenge in Decision Tree:-

In Decision Tree, the major challenge is to identification of the attribute for the root node in each level. This process is known as attribute selection. We have several common attribute selection measures to determining the root node Information Gain, Gini Index and Entropy.

1- Information Gain

The information gain is based on the decrease in **Entropy** after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches) [78].

2- Entropy

a decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous, the entropy is zero and if the sample is an equally divided, it has entropy of one.

3. Gini Index

Is a metric for classification tasks, It stores sum of squared probabilities of each class [78].

Algorithm 1: Decision tree learning

Input: A set of training instances, attribute **Ai**, where I is index of the attribute, value **Vj** where j is the index of the value

Output: A decision tree T

```
1   if stooping criterion is satisfied then
2    |  create a leaf that corresponds to all remaining training instances
3    end
4    else
5       | choose the best (according to same heuristics) attribute Ai
6       |  label the current node with Ai
7          for each value vj of the attribute Ai do
8           | label an outgoing edge with value vj
9           |  recursively build a sub tree based on a subset of training instances
            | that meet the condition "Ai = vj"
10         end
11  end
```

2-5 Android Operating system

Is a free and open source operating system designed primarily for touch screen devices such as smartphones and tablets, where the Android user interface is mainly based on direct processing, using touch gestures that are largely compatible with realistic movements, such as clicking, scanning, and fingerprinting, from In order to deal with things on the screen panel, in addition to the default keyboard for entering text, in addition to this this system features a set of applications and tools to perform various functions and services, such as the application of e-mail, SMS and calendar Maps, browser, contacts and other applications [80] . In previous days, mobile phones were just small phones and used to make calls only, and with the passage of time, mobile devices have become more advanced and available for use by all groups of society and it is easy to get them anywhere and at any time [81]. In addition, mobile services have become more advanced and in line with what users want, as they are able to capture data and watch movies. Internet services have also developed very quickly and there are applications that deal with the Internet and phones at the same time, such as YouTube, social networking programs, iCloud, etc. Where Google has developed touch devices as well as Android TV devices for TVs, Android Auto cars, and Android Wear watches. As each developed with a special user interface. Types of Android systems are also used on laptop computers, game consoles, digital cameras, and other electronic devices. In addition, the Android system is easy to work with low-power devices that run on battery and are full of devices such as GPS receivers, cameras, light and direction sensors, and WiFi connection. Also, the Android system, unlike other mobile operating systems, where Android applications are characterized by being built using the Java programming language [82]. Therefore, all of the above has contributed to the Android operating system being one of the most common systems, and Android applications can be easily developed, as the Java language provides maintenance and development libraries for Android functions, as well as the basic tools needed to create some mobile applications, as Android development tools make Operating, debugging and testing applications is fast, as well as contributing to the success of this system and its applications used. Recently, several studies have been conducted on factors that may be linked to the success of system applications to contribute to increasing the chances of its success and providing the best services to users [83]. Table 2.1

Table 2.1 Factors affecting the rating of an app.

Dimension	Factor (Name)	Explanation	Rationale
Size of App	Install	Binary size of the APK file (measuredin KB)	The size of the installation and the number of project categories may indicate more features and lead to better functions.
	Total classes (num-class).	Total number of classes (includinglibrary code).	
	App classes(num-project-Class).	Total number of app specific classes.	
	Number of activities(num-activity).	Total number of activities defined in the AndroidManifest.xml file.	
	Number of services(numservice)	The total number of services selected	
Complexity of Code	Weighted methods perclass (wmcmean)	In each category we find the average number of methods	Chidamber and Kemerer are metrics for measuring the complexity of coding.
	Depth of inheritance tree(ditmean).	For each category we find the average length of the features chain	
	Number of children(nocmean).	number of sub-classes of each class.	

	Coupling between objectclasses (cbomean).	The number of categories that are associated with each other	
	Response for a class(rfcmean).	The number of different methods that can be implemented	
	Lack of cohesion in methods (lcommean)	Average number of road category	
	Afferent coupling(camean).	Average number of categories based on each table category	
	Number of public methods (npmmean).	General methods.	
Dependence on Library	Total dependency on Android libraries(depandroid).	Total number of libraries	Too much dependence on the APIs might impact the quality of the code.
	Total dependency onthird-party libraries(depthird).	Total number of calls to third partylibraries	
	Percentage dependencyon Android libraries(depperandroid).	The percentage of calls	Similar with depandroid and depthird, butthese two

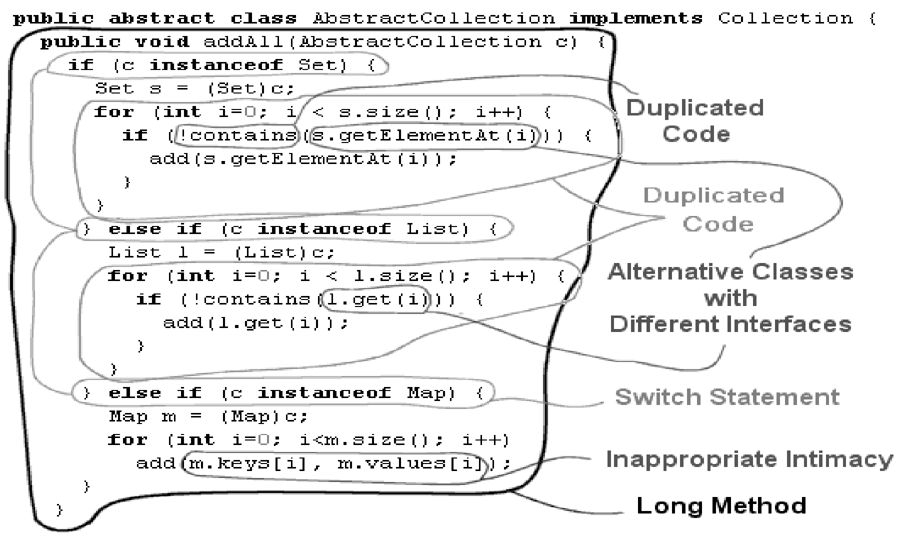
	Percentage dependency on third-party libraries(depperthird).	Percentageof callsto third partylibraries	factors consider proportion
Quality of Library Code	Change of used Android	number of methods are use in the Android.	Change and defect-proneness of adopted Android APIs might represent quality of the used Android APIs.
	Faultiness of usedAndroid API (apibug).	number of bugs in Android.	
Complexity of UI	Input elements per layout(uiinput).	The number of input elements.	The complexity of the user interface affects the success of the Android app
	Output elements perlayout (uiouput)	The number of output elements.	
Requirement s	Minimum SDK version (min SDK).	he minimum SDK version to run the app	The SDK version indicates that apps are updated frequently to add the latest available features

	Target SDK version(targetSDK).	The version of the SDK that the application targets.	The high SDK version indicates that developers have tested their application on SDKs.
	Required device features(devicefeature).	Required features from the user's device.	Some required device features may further complicate the application.
	Required user permission(userpermission).	Number of permissions needed fromuser.	Due to privacy concerns, some users may be sensitive to this
Marketing Effort	Length of description	The number of words for the app description on the store page	It would be preferable to offer clear job and more features.
	Promotional images(numimg).	The number of images on the app store page	

Category of App	App category	Category	Users are sure to have different expectations for the app based on their understandin g and requirements

2-6 Code Smell

Code smells are manifestations of design flaws that can degrade code maintainability [84]. Figure 2.13. These symptoms may arise because of activities performed by developers during design or development or by making bad decisions or using so-called countermeasures [85],[86]. Which increases the complexity of the code and Difficulty of maintenance, and considered is a true and ideal indicator of maintenance evaluation. Where, Code smells indicate that there are problems with the application code, and some problems such as difficulty in understanding and modification, which can lead to faults. To overcome this problem code smells must be identified and dealt with, the errors type (Code Smells) are identified by the structural information extracted from the source code [87], [88]. In general, the smell of code has a negative connotation about the design that affects program performance for the task. It also has great potential to ensure the quality of the program. There are many studies has been for the of aiming at detecting and removing bad Code Smells [89],[36].



```
public abstract class AbstractCollection implements Collection {
  public void addAll(AbstractCollection c) {
    if (c instanceof Set) {
      Set s = (Set)c;
      for (int i=0; i < s.size(); i++) {
        if (!contains(s.getElementAt(i))) {
          add(s.getElementAt(i));
        }
      }
    } else if (c instanceof List) {
      List l = (List)c;
      for (int i=0; i < l.size(); i++) {
        if (!contains(l.get(i))) {
          add(l.get(i));
        }
      }
    } else if (c instanceof Map) {
      Map m = (Map)c;
      for (int i=0; i<m.size(); i++)
        add(m.keys[i], m.values[i]);
    }
  }
```

Figure 2.13 Code Smell errors

2-6-1 Some Types of Code Smell

In the table below, Eight types of Code Smell with a simplified explanation for each of them [90],[91] . Which will be addressed in this Thesis (See Table 2.2).

Table 2.2 Types Code Smell errors

Smell	Description
Swiss Army Knife (SAK)	A *Swiss army knife* is a category with various interface signatures , leading to a really complicated category interface designed to handle a good diversity of abstractions.
Long Method (LM)	*Long methods* are enforced with rather more lines of code than alternative strategies. theyre usually terribly complicated .
Member "Ignoring Method (MIM)	In Android , "when a method" does not access" an object" attribute , it is recommended" to use" a static" method because "they are" about 15%–20% "faster than a dynamic invocation

No Low Memory Resolver (NLMR)	When the system is running low on memory , the system calls the strategy on Low Memory() of running activities , that ar presupposed to trim their memory usage. If this technique isnt enforced by the activity , the system kills the method so as to free memory , and may cause AN abnormal termination of programs.
Blob Class (BLOB)	A Blob class , is a category with an outsized variety of attributes and/or operations.
Internal Getter/Setter (IGS)	On system , fields ought to be accessed directly among a category to extend performance. The usage of an inside getter or a setter converts into a virtual invoke , that makes the operation thrice slower than a right away access.
Leaking Inner Class (LIC)	In Java , non-static inner and anonymous categories area unit holding a relevancy the outer category , whereas static inner categories dont seem to be. this might provoke a memory leak in robot systems.
Complex Class (CC)	A *complex class* is a category containing advanced strategies. Again , these categories are laborious to know and maintain and wish to be refectories.

2-6-2 Standards Metrics of Code quality used to detect Code Smell

- **Dimensional Metrics**

  This type of measure aims to provide a quantitative measure of the program in terms of code sizes and modules. Despite a large code size can be a symptom of a lot of provided features, it might led to an higher probability to contain bugs [92],[93]. Table 2.3

Table 2.3-Dimensional Metrics to detect Code Smell

Metrics	Description	Measuring Method
Number of Byte- code Instructions (NBI)	This metric counts the total number of small byte-code instructions, ignoring comment lines and blank lines (i.e., NCLOC)	Counting all byte-code instructions not beginning with the characters “#” and “.”
Number of Classes (NOC)	Calculates the number of categories within the application package	Counting the occurrences of the class directive
Number of Methods (NOM)	This metric calculates the number of methods within an application package	Counting the occurrences of the method directive
Instructions per Method (IPM)	Measurement is made by calculating the ratio between the total number of instructions and the total number of methods	Computed by the formula (IPM) = (NBI/NOM)

- **Complexity Metrics**

  The complexity of applications is estimated and measured by special measures Where the application understanding process is strongly correlated the complexity of the application, the results of these measures help us to understand the cost of understanding the product when conducting some activities (including testing activities). In addition, the greater the complexity of the application, the greater the tracks and results of the implementation of the metrics, and therefore it is difficult to test the application in full. Thus it is difficult to understand complex applications [94],[93].Table 2.4

Table 2.4 Complexity Metrics to detect Code Smell

Metrics	Description	Measuring Method
Cyclamate Complexity (CCy)	The number of independent and incoming tracks is measured when controlling the program	The mean cyclic complexity is calculated for the application categories
Weighted Methods per Class (WMC)	the sum of the complexities of all class methods	Average of this standard was calculated according to the formula WMC = (NOM / NOC).

- **Object-Oriented Metrics**

Because mobile applications are built using object oriented programming languages (like Java, C ++, C #, etc.), it is possible to evaluate the quality of an application's code using a set of metrics that measure the complexity, consistency, and conjugation of code [95],[93]. Object Oriented Metrics play an important role in the development of the software product , and the Estimating object-oriented metrics is essential in software engineering, including measuring software code complexity, and estimating product size and quality [96]. Table 2.5

Table 2.5 Object-Oriented Metrics to detect Code Smell

Metrics	Description	Measuring Method
Number of Children (NOCH)	This metric indicates the number of direct subcategories for a category in the category hierarchy.	Class origin is determined by hyper-route. To reduce data to one value per application
Depth of Inheritance Tree (DIT)	This scale indicates depth, and deeper trees pose more complexity in design	We took into account the depth of each inherited tree found with superior orientation, with the scale being the deepest tree of any category in the application
Coupling Between Objects (CBO)	This scale indicates the degree of dependency between categories	The scale value is obtained by calculating external methods.
Percent Public Instance Variables (PPIV)	This scale indicates the percentage of variables	This type of variable is calculated in all categories
Access to Public Data (APD)	This metric indicates the number of times the general attributes of each category are accessed	We computed the average of this value among all app's classes

Lack of Cohesion in Methods (LCOM)	This scale indicates the level of cohesion between separation methods and features.	Lower percentages indicate greater cohesion of data and methods. In the smile byte-code the class attributes are marked by the keyword.

- **Android-oriented Metrics**

  Mobile applications have become complex software systems that must be developed quickly and constantly to suit new user requirements and implementation contexts [8] . Addressing these requirements can lead to poor design choices. This can lead to malfunctions, problems, resource consumption, and poor response. This kind of metrics aims to evaluate the application after maintenance and development, as these metrics help guide developers to reconfigure their applications and thus improve their quality. We describe the metrics in this category in Table 2.6. [97],[98] .

Table 2.6 Android-oriented Metrics to detect Code Smell

Metrics	Description	Measuring Method
Bad Smell Method Calls (BSMC)	Making exceptions can cause application crashes	The number of times these methods are called is calculated.
Wake Locks with no timeout (WKL)	To allow the device to be kept in an active state, the acquisition methods and the release method can be called to allow screen shutdown	The number of times the acquisition method is invoked is calculated.
Number of Location Listeners (LOCL)	The Android app can track the position of users. But it reduces energy	This metric computes the number of times a Location Listener object is instantiated

Number of GPS Uses (GPS)	Android apps can access the user's location, but it consumes battery power	Gaining instructions for codes containing the GPS series
XML Parsers (XML)	Preferring Android apps can save battery power	Calculating the byte code instructions from the fixed call type.
Network Timeouts (NTO)	Here, application developers are allowed to set time limits for establishing a TCP connection via network timeouts	All Smellbyte instructions are calculated here
Networking (NET)	In Android applications, some problems can cause the app to not respond to user requests	This type of metric computes all Smell byte instructions
File I/O (I/O)	I / O operations can cause application response delays, and simple operations may require large and unexpected response times	Recurrences are counted between the invocation: smali byte code instructions for the Ljava / io / File types

SQLite (SQL)	The database can generate a large amount of exorbitant operations to the portable storage unit, which can lead to problems	The frequency is calculated between the calling byte code instructions for Landroid / database / sqlite / SQLiteDatabase
Bitmaps (BMAP)	Large image processing can be computationally expensive	For this metric we count the occurrences of the method invocation Bitmap Factory.

2-7 Text processing technique

The amount of texts that are created every day increases dramatically, as there are many practices that create texts in different languages and formats such as social networking applications, application download sites, etc. The rapid development of techniques for obtaining text and digital data has also contributed to the increase in the size and magnitude of data [99]. In recent years, the growth of digital data has increased dramatically. It is expected that the volume of data will grow to forty zettabytes by 2020, fifty times more than in 2010 [100]. Consequently, it is difficult to process this huge volume of unstructured data and its perception by computers. Therefore, it is necessary to find effective techniques and algorithms to discover and extract text styles. Text data is one of the simplest data and a simple example of unstructured data, as humans can deal with unstructured text easily, but this seems difficult on a computer.[99]. Understanding and processing texts extracted from a large amount of texts contributes to the success of applications, as in applications of market sales analysis and business management [101]. In many of these applications, textual information is stored in the application's database, so exploration of the text is one of the most difficult procedures. In addition, extracting the required information from the user is the difficult issue. In addition to the need to bring these data closer to useful information, studies indicate that more than 80 percent of the data is disorganized or semi-structured data [102]. As a result, there is an urgent need to conduct research, find, and design methods and algorithms that help in the processing and organization of texts. The results of word processing, especially the large ones, have a significant impact on the application success rate. Whereas, research on knowledge discovery and data mining has received great attention, this allowed a lot of attention to be given to this field due to the huge amount of textual. Where researchers search and investigate the best methods of exploration and analysis of texts. Where the right and appropriate method contributes to improving speed and reducing the time effort required to extract valuable information. Researchers usually use some linguistic concepts to analyze texts, such as a portion of speech (name, verb, adjective, etc.), grammatical structure and use of many cognitive iterations, such as dictionary of words, their meanings and characteristics, and a set of grammatical rules and other resources such as entities and procedures, or a glossary Synonyms or abbreviations [103]. Sometimes text analysis techniques are repeated until the information is extracted. The resulting information can also be placed in a special system, which results in a great deal of knowledge for the user of this system in the process of extracting the text. Figure 2.14. In addition, there are other methods such as storing most of the information related to commercial, industrial, governmental, and other institutions in a text database and organizing them with a number of structured fields such as title, name of authors, publication date, category, etc. Where researchers are very interested in developing methods of exploration and analysis of texts, as well as in exploring new ways to integrate different technologies, as well as some other technologies that are a new field for text processing and benefit from them in increasing the accuracy of text extraction and deriving high-quality information [104]. In addition to that, text analysis techniques and making comparisons to find matching between texts. See Algorithm 2

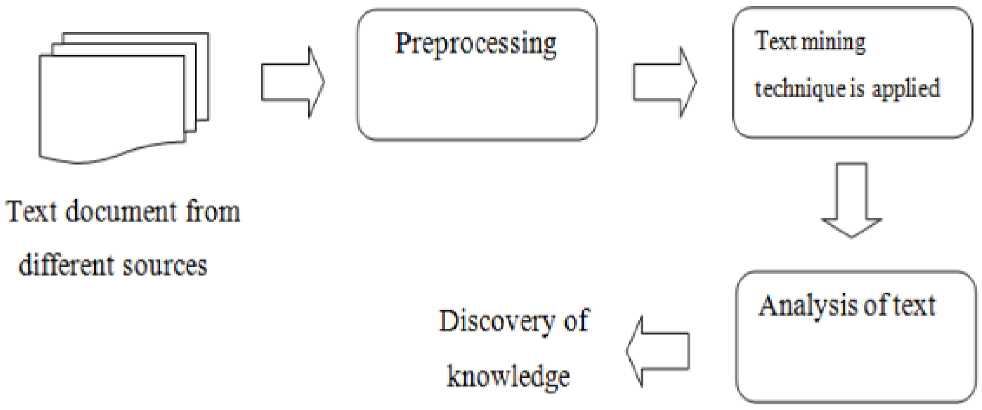

Text document from different sources
Preprocessing
Text mining technique is applied
Analysis of text
Discovery of knowledge

Figure 2.14 Types Text mining process

Algorithm 2: Matching algorithm

```
  Input: Floats x and y, which are a pattern file and a test file
  Output: True (malicious) or False (A, B)
  Procedure similarity-match (A, B)
    A=|A|
    B=|B|
    O = 0.05
    C = A * C
    If |A B| < C then return True;
    Else, return false
  End procedure
Procedure BEGIN(x, y)
   Test Attribute Value <----- y
   n <----- 0
   While not at end of pattern or unmatched do
      Pattern Attribute Value <----- read a line form x
      For n < number of attributes do
         n += 1
         Test n <----- test Attribute Value [n]
         $Pattern_n$ <----- Pattern Attribute Value [n]
         $M_n$ <----- similarity-match ($pattern_n$ , $test_n$)
      IF all $M_n$ == True then break;
      Else, move to next pattern;
   End procedure
```

2-8 Dataset Description

This chapter includes a comprehensive description of all concepts and techniques that we will need in the system design process. The chapter includes a variety of important topics such

The data set was collected by a dedicated team and published on GitHub, a well-known site hosting open source software projects. Includes 288, 065 reviews extracted from Google Play, and relates to 629 apps (395 open source apps with new versions) and (23) different categories of apps, extracted from F-Droid. The dataset consists of six parts (individually as a CSV file). Dataset is available here [23].

https://github.com/sealuzh/user_quality/tree/master/csv_files.

Each part contains certain information as follows: -

- Part one (**Versions**): -

  This part contains information about 629 versions of 395 Android apps from 23 different categories (games, news, magazines, sports, health, communications, education, etc.) where each row is included (ID / foreign key, package name, category, version code, release date, Num-Comments). Table 2.7, Table 2.9

Table 2.7 the Versions Table

PK/FK	Field Name	Description
F- Key	id	The unique id for the apk
	package_name	The package name for the apk on the Google Play Store
	category	The category of the apk
	version_code	The version code of the apk
	release_date	The release date of the apk
	count	The number of reviews mined for the apk

- Part two (**Reviews**): -

Contains over 288,065 comments on these apps, which include (ID / Primary Key, Package Name, Comment, Comment Date, Star Rating, ID / Foreign Key). According to Table 2.8

Table 2.8 the Reviews Table

PK/FK	Field Name	Description
P-Key	review_id	The review unique id
	package_name	The package name for the apk on the Google Play Store
	review_content	The text of the review
	review_Date	The posting date of the review
	start_rating	The rating of the review expressed in number of starts
F-Key	app_version	The id of the correspondent apk version

- Part three (**Sentences**): -

This section contains 451,294 comments on Android apps that are categorized according to one intent and for one or more topics, including (ID / Primary Key, Review, Intent, Total Topic). Table 2.10. There is also a description of the users' comments according to intent and subject, as in Table 2.10 and Table 2.11

Table 2.9 Categories Android applications and number of versions

App Category	Apps	Total of apps	Total reviews
Books & Reference	19	35	15,892
Business	1	1	1,172
Comics	4	5	2287
Communication	33	64	31,219
Education	12	14	1,291
Entertainment	5	5	2,584
Finance	7	10	621
Games	10	44	20,378
Health & Fitness	3	4	1,149
Libraries & Demo	3	5	990
Lifestyle	4	6	246
Maps & Navigation	10	15	1,411
Music & Audio	17	32	3,025
News & Magazines	6	9	1,988
Personalization	18	26	12,037
Photography	7	10	4,275
Productivity	45	80	8,361
Shopping	2	2	2,647
Social	7	11	6,146
Tools	139	214	151,509
Travel & Local	9	12	984
Video Players & Editors	12	22	15,352
Weather	2	3	2,501
TOTAL	395	629	288,065

Table 2.10 the Sentences Table

PK/FK	Field Name	Description
F-Key	id	The idof thereview towhich the sentence belongs
	review	The text of the sentence
	category	The intention of the sentence
	topic	The topic discussed in the sentence

Table 2.11 Intention Categories Definition

Information Giving	Reviews that explain to users some aspects of the application
Information Seeking	Reviews of users requesting information about the app.
Feature Request	User reviews express ideas or suggestions to improve the application.
Problem Discovery	Reviews that report unexpected problems.
Other	Reviews that do not represent the above categories

Table 2.12 Topic Categories Definition

Cluster	Description
App	Reviews related to the entire application such as general notes and assessments
GUI	Reviews related to the graphical user interface or application format.
Contents	Reviews of application content.
Pricing	Reviews related to the price of the application
Feature or Functionality	Reviews related to application functionality.
Improvement	Reviews for the purpose of improving the application.
Updates/ Versions	Reviews of application releases.
Resources	Reviews related to device resources such as battery consumption, storage, and the like
Security	Reviews related to app security or privacy
Download	Reviews related to app download.
Model	Reviews related to notifications about operating system versions.
Company	Reviews about the company or organization responsible for developing the application.
Other	Reviews do not represent any of the previous topics.

- Part four **(Code-Smell): -**
  This section contains a complete description of each application in terms of (ID / foreign key, Package_name, types of “Code Smell “error, number of error). It has 629 rows. See Table 2.13

Table 2.13 the Code-Smell Table

PK/FK	Field Name	Description
F-Key	id	The unique id for the apk
	BLOB	Value of Blob Class smell
	LM	Value of the Long Method smell
	SAK	Value of the Swiss Army Knife smell
	CC	Value of the Complex Class smell
	IGS	Value of the Internal Getter/Setter smell
	MIM	Value of the Member Ignoring Method smell
	NLMR	Value of the No Low Memory Resolver smell
	LIC	Value of the Leaking Inner Class smell

- Part five (**Code Metrics**): -
  This section contains the results of 22 metrics used to detect “Code Smell “errors in android apps and analyze app code quality, and also Contains 629 rows including (ID / Primary Key, Package_name, Types of Errors Detected, Number of Errors Detected). Several tools were used, including a tool to decompress Android apps, access internal files, and check the quality of the written code. Table 2.14

Table 2.14 Code Metrics Table

PK/FK	Field Name	Description
Primary Key	id	A unique id for an apk
	NBI	Number of Byte- Code Instructions
	NOC	Number of Classes
	NOM	Number of Methods
	IPM	Instructions per Method
	CC	Cyclomatic Complexity
	WMC	Weighted Methods per Class
	NOCH	Number of Children
	DIT	Depth of Inheritance Tree
	LCOM	Lack of Cohesion in Methods
	CBO	Coupling Between Objects
	PPIV	Percent Public Instance Variables
	APD	Access to Public Data
	BSMC	Bad Smell Method Calls
	WKL	Wake Locks with no timeout
	LOCL	Number of Location Listener
	GPS	Number of GPS Uses
	NTO	Network Timeouts
	NET	Networking
	I/O	File I/O
	SQL	SQLite
	BMAP	Bitmaps

- Part six (**User Metrics):**

This section includes the results of the ratio between the main categories "Intention and Topic" and the number of comments categorized by Intention and Topic. In addition, mathematical ratios for categories. It contains 629 rows including (ID/Primary Key, Package_name, star rating, total number of reviews, etc.). Table 2.15

Table 2.15 Users Metrics Table

PK/FK	Field Name	Description
P- Key	id	A unique id for an apk
	total_reviews	total of reviews for each apk
	total_sentences	total of sentences for each apk
	average_rating	The average rating collected reviews
	feature_requests	Number of sentences which have been classified as Feature Requests
	%feature_request	Percentage of sentences which have been classified as Feature Request over the total for an apk
	problem_discoveries	Number of sentences which have been classified as Problem Discoveries
	%_problem_discoveries	Percentage of sentences which have been classified as Problem Discoveries over the total for an apk
	information_seeking	Number of sentences which have been classified as Information Seeking
	%_information_seeking	Percentage of sentences which have been classified as Information Seeking
	information_giving	Number of sentences which have been classified as Information Giving
	%information_giving	Percentage of sentences which have been classified as Information Giving
	other intention	Number of sentences without any intention associated
	App	Number of sentences which have been classified as talking about App
	GUI	Number of sentences which have been classified as talking about GUI
	Contents	Number of sentences which have been classified as talking about Contents
	Download	Number of sentences which have been classified as talking about Downloads
	Company	Number of sentences which have been classified as talking about Company
	Feature/Functionality	Number of sentences which have been classified as Feature /Functionality
	Improvement	Number of sentences which have been classified as talking about Improvement

Pricing	Number of sentences which have been classified as talking about Pricing
Resources	Number of sentences which have been classified as talking about Resources
Update/Version	Number of sentences talking about Update /Version
Model	Number of sentences which have been classified as talking about Model
Security	Number of sentences which have been classified as talking about Security
Other_topic	Number of sentences not assigned to any topic.

And here the schema of the dataset of reviews and metrics, with how each the six parts linking with the other Figure 2.15

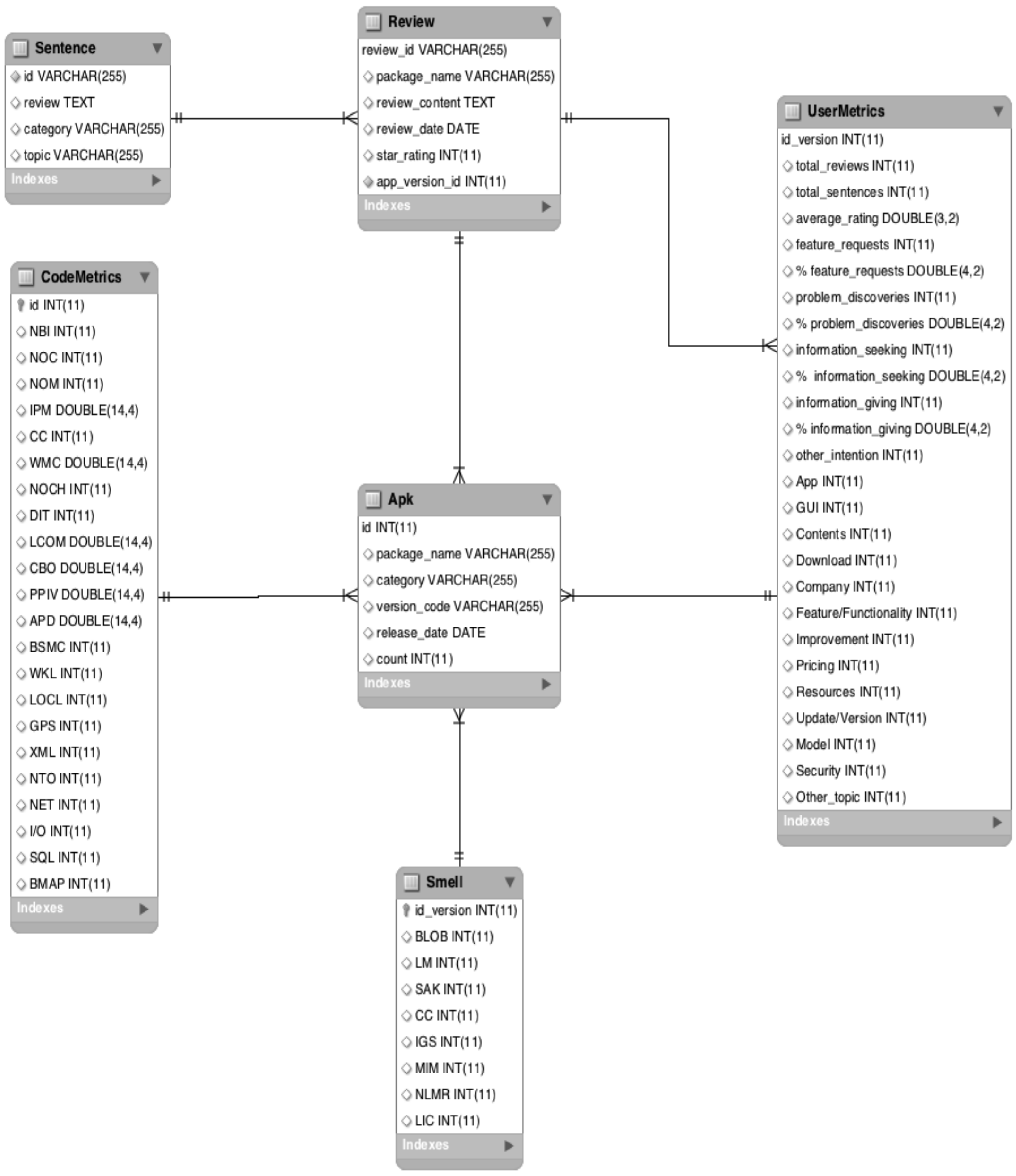


Figure 2.15 the schema of the dataset

2-9 Accuracy calculation

The accuracy of the results was measured by the well-known "**Confusion Matrix**" which is used for the purpose of calculating the accuracy and performance of the model, the details of which are as follows:

	Predicted No	**Predicted Yes**
Actual: no	**TN**	**FP**
Actual: yes	**FN**	**TP**

True positives (**TP**): These are cases in which we predicted yes (they have the disease), and they do have the disease.

True negatives (**TN**): We predicted no, and they don't have the disease.

False positives (**FP**): We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")

False negatives (**FN**): We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

This is a list of rates that are often compute from a confusion matrix for a binary classifier:

Accuracy: Overall, how often is the classifier correct?

(TP+TN) / Total

Error Rate: Overall, how often is it wrong?

(FP+FN)/Total

Recall: When it is actually yes, how often does it predict yes?

(TP)/ TP+ FN

Precision: When it predicts yes, how often is it correct?

(TP)/ TP+ FP

2-10 Summary

This chapter includes a comprehensive description of all concepts and techniques that we will need in the model design process such as various software engineering tasks, Machine learning algorithms and Code smell errors, in addition to text processing techniques and description android system. In addition, this chapter includes a comprehensive description of all six parts of the dataset that the system will handle. This chapter it describes in detail the main elements above that are common to the project construction and design process. A number of tables and graphs have also been added to benefit from them to reduce the time and effort involved in the research process.

Chapter Three

The Proposed System

CHAPTER THREE
THE PROPOSED SYSTEM

3.1 Overview

This chapter includes a complete description of the proposed system. Where user comments about applications in download stores that were previously categorized according to intentions and topics will be dealt with (Table 2.11 and Table 2.12) and take advantage of these comments to design a model to detect errors and performance problems in Android applications in addition to diagnosing problems. The model was constructed using a decision tree algorithm. Decision tree are well-known machine learning technique for solving complex classification problems. Software engineering tasks were also strictly adhered to for saving effort, time and cost.

The model also aims to facilitate the understanding of the large and increasing number of user reviews about applications and discover the largest possible number of performance problems, and benefit from them in avoiding frequent maintenance and focus on tools that help in creating and developing applications while providing the required features by users quickly, with a view to raising application success rates and achieve the goal of software engineers.

3.2 The Proposed System

In this project, we will focus on some software engineering tasks such as testing, maintenance and development, with the purpose of building a model that contributes to the rapid completion of these tasks, Where the decision tree classification was used to design a model to detect errors and performance problems in Android applications, in addition to diagnosing problems in this type of application. Figure 3.1

The model consists of Four stages: -

1- Pre-Processing stage of Dataset reviews
2- Pre-Processing stage of new reviews and star rating.
3- Decision Tree classifier Model
4- Similarity measuring and building vector

Each of the above-mentioned stages plays an important role in diagnosing the largest possible number of problems in the applications (malfunction in the application or in one of the functions), with the aim of expediting maintenance and adding what the users want in the applications. In each stage of the model, one or more of the six parts of the Data Set are dealt with. See Algorithm 3

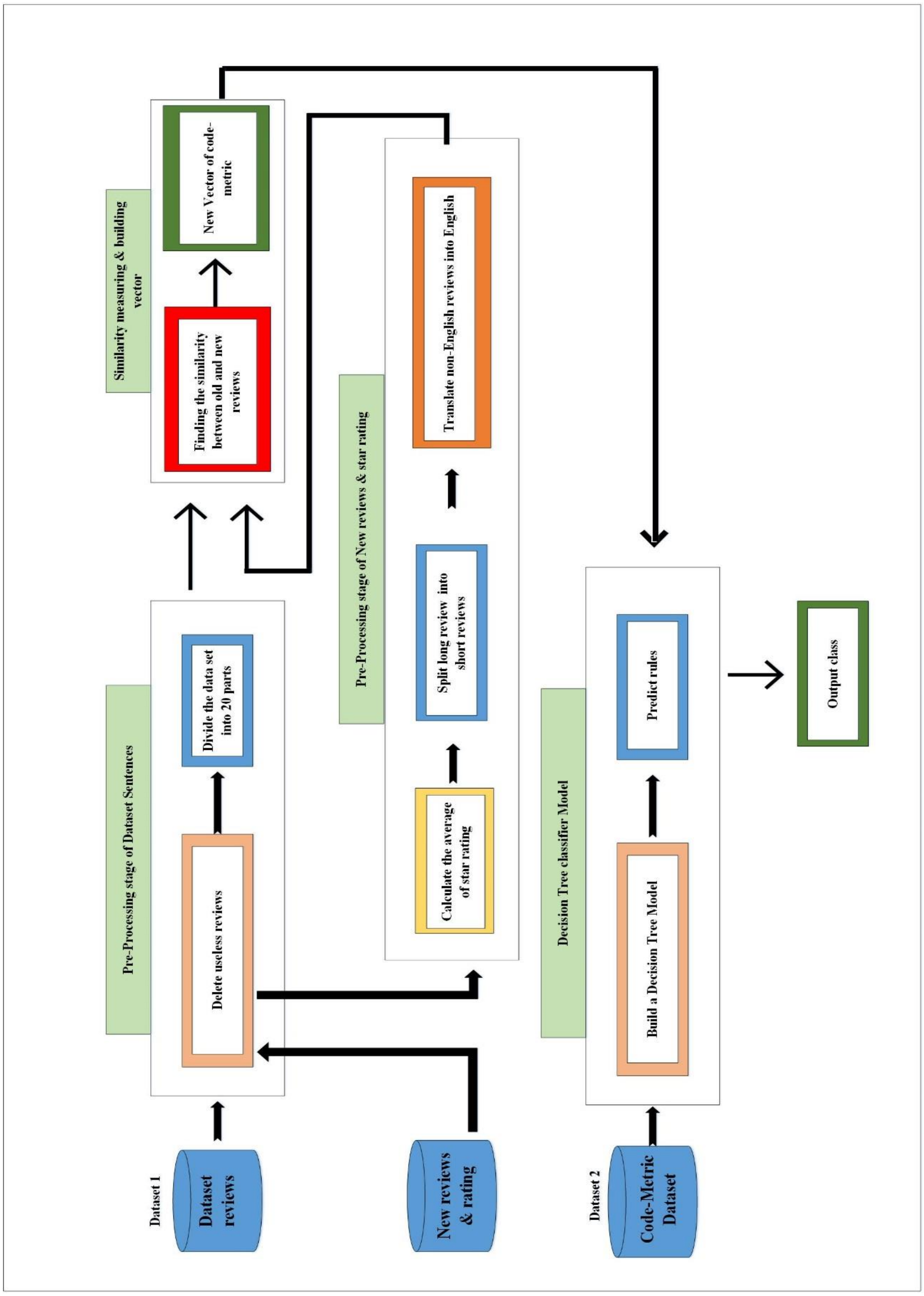

Dataset 1
Dataset reviews
New reviews & rating
Dataset 2
Code-Metric Dataset
Pre-Processing stage of Dataset Sentences
Delete useless reviews
Divide the data set into 20 parts
Pre-Processing stage of New reviews & star rating
Calculate the average of star rating
Split long review into short reviews
Translate non-English reviews into English
Similarity measuring & building vector
Finding the similarity between old and new reviews
New Vector of code-metric
Decision Tree classifier Model
Build a Decision Tree Model
Predict rules
Output class

Figure 3.1 Diagram of the proposed system

Algorithm 3: Proposed System Algorithm

Input: Excel file includes two columns u and b **A** = Comments, **B** = Star rating

Output: "**0**" means that the application is good, "**1**" It means that the application contains problems

```
1  BEGIN
2  Processing stage of data set reviews
3    | delete useless reviews
4    |divided the dataset into 20 parts
5  Pre-Processing stage of A and B
6    | Calculate the average of star rating
7    | Split long review into
8    | Translate non-English reviews into English
9  Build a Decision Tree Model
10   | Call Algorithm 1
11 Similarity measuring and building vector
12   | Call Algorithm 2
13   | building New Vector of code-metric
14    if New Vector  == "0" then print "the app is good"
      Else print, "the app is have error" and print A
15 End
```

Below is a detailed explanation of the process for building each of the four stages of the model: -

- **Pre-Processing stage of Dataset reviews: -**

  At this point, the third part of the data set is dealt with. Where this part was prepared to work within the requirements of the model, this is done in two operations. The first operation is that the data set is filtered for the purpose of eliminating empty and duplicate fields and unwanted fields (numbers, repeated letters, etc.). Where it is filter (451,294) from users' comments. After completing this process, get (330,581) comments arranged according to intention and Topic. Figure 3.2

In the second process of this stage, after relying on the results of the classification of the data collection task force, the results of classification accuracy were for the categories of intentions (ranging between 84% and 89%) and for the Topic categories (76%).

The process included how to sort all the comments in the database and divide them into 20 parts, according to length. This was done after taking into account the linguistic principles of the English language sentences.

Each part represents one of the lengths of the comment. Mathematical equation has been made to avoid errors, as it starts Dividing data from one letter to 105 or more characters, each of these parts deals with a specific length of comments, for example a comment that is between 27 to 31 characters long will deal with one of the twenty parts that includes the length of the comments between (25 to 35 characters), where a special standard has been set for dividing these parts, which is to add 5 characters as an increase rate , and erase 5 characters as an shortage rate for the purpose of helping to avoid the mistakes of the user when writing his comment. Table 3.1

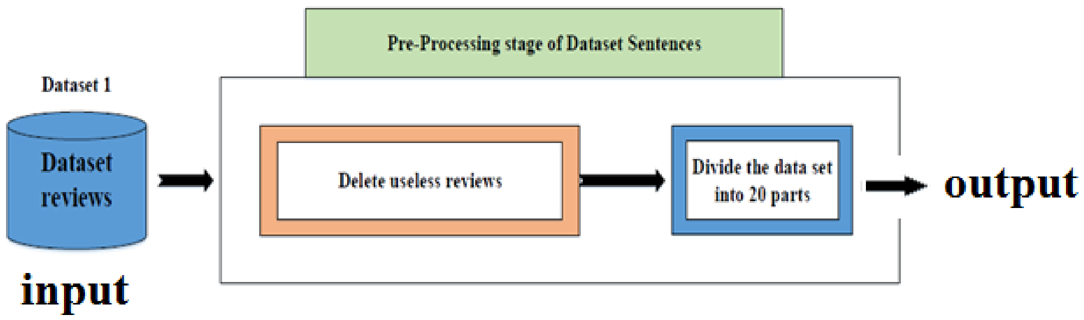

Pre-Processing stage of Dataset Sentences
Dataset 1
Dataset reviews
input
Delete useless reviews
Divide the data set into 20 parts
output

Figure 3.2 Pre-Processing stage of Dataset reviews

Table 3.1 the length of the comments stored in the 20-dataset portions

The length of the new Comment	The length of the Comment in the data set	Part Number	Number of Comments
Commend < 10	**Command < 10**	**Part 1**	**٢٢٠٩٤**
Commend = 10 or 11	**5<= Command <= 15**	**Part 2**	**٥٣٤٨٥**
11 < Command <= 16	**10 <= Command <= 20**	**Part 3**	**٦٤٩١١**
16 < Command <= 21	**15 <= Command <= 25**	**Part 4**	**٦٣٦٨٤**
21 < Command <= 26	**20 <= Command <= 30**	**Part 5**	**٥٧٧٦٣**
26 < Command <= 31	**25 <= Command <= 35**	**Part 6**	**٥١٢٥٣**
31 < Command <= 36	**30 <= Command <= 40**	**Part 7**	**٤٤٧٥٢**
36 < Command <= 41	**35 <= Command <= 45**	**Part 8**	**٣٩٣٠٨**
41 < Command <= 46	**40 <= Command <= 50**	**Part 9**	**٣٥٠٥٩**
46 < Command <= 51	**45 <= Command <= 55**	**Part 10**	**٣١٥٢٠**
51 < Command <= 56	**50 <= Command <= 60**	**Part 11**	**٢٨٢٠٥**
56 < Command <= 61	**55 <= Command <= 65**	**Part 12**	**٢٥٣٤٠**
61 < Command <= 66	**60 <= Command <= 70**	**Part 13**	**٢٢٨٥١**
66 < Command <= 71	**65 <= Command <= 75**	**Part 14**	**٢٠٣٥٤**
71 < Command <= 76	**70 <= Command <= 80**	**Part 15**	**١٧٩٧٣**
76 < Command <= 81	**75 <= Command <= 85**	**Part 16**	**١٥٨٩٦**
81 < Command <= 86	**80 <= Command <= 90**	**Part 17**	**١٤٢١5**
86 < Command <= 91	**85 <= Command <= 95**	**Part 18**	**١٢٦٦٠**
91 < Command <= 96	**90 <= Command <= 100**	**Part 19**	**٩٧٤٢**
96 < Command	**95 <= Command**	**Part 20**	**٤٠٨69**

- **Pre-Processing stage of new reviews** and **star rating: -**

  At this point in the form, reviews will be handled in download stores that were previously collected in the second part' of the master data set. These comments are user reactions about the performance of the Android application. The model prepares these reviews for testing the application. In this stage, three tasks are accomplished, namely: -

  - The form will delete unwanted reviews (duplicate, number-only reviews, two-letter reviews, etc.) and also calculate the average' star rating. Where it is calculated mathematically, by dividing the sum of all-star ratings (1 to 5 stars) by the number of reviews. To be used in calculating the model results later.
  - Long comments will also be processed and converted into short.
  - If comments are written in a language other than English, Google translator will translate them into English.

  At the end of the three operations above, this stage ends Figure 3.3.

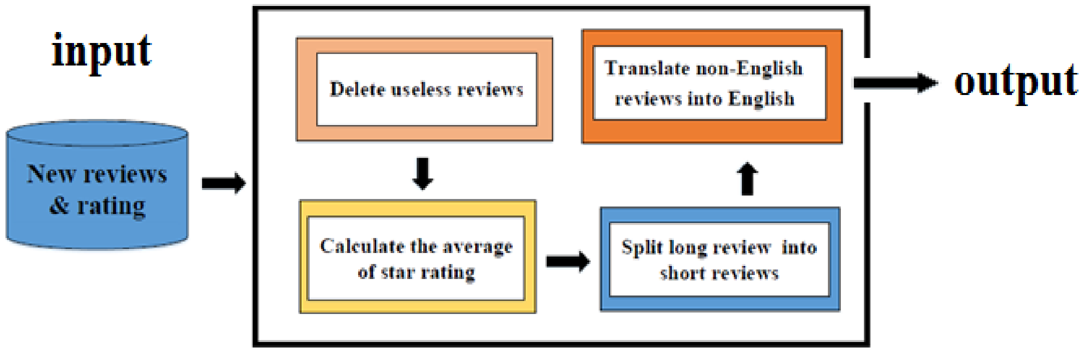

input
New reviews & rating
Delete useless reviews
Calculate the average of star rating
Split long review into short reviews
Translate non-English reviews into English
output

Figure 3.3 Pre-Processing stage of new reviews and *star rating*

- **Decision Tree classifier Model: -**

At this stage, the decision tree algorithm is building based on the results found in the sixth section of the data set. This section includes the results of the ratio between the main categories Intention and Topic, the number of comments classified by Intention and Topic, and the mathematical ratios of all classification categories (average_rating , total_reviews , total_sentences ,feature_requests, %feature_requests,problem_discoveries,%problem_discoveries,information_seeking,%information_seeking,information_giving, %information_giving , other , App , GUI , Contents , Download , Company , Feature/Functionality , Improvement , Pricing , Resources , Update/Version , Model , Security , Other). It also contains 629 rows. The model trained based on the results found in the fifth section of the data set (80% for training and 20% for testing).

The basic idea of making a decision tree algorithm is to determine the best feature using Attribute Selection Measures (ASM) to divide records, divide the data set into smaller subgroups, and repeat this process until all groups are categorized into the same attribute value. A test was performed to determine the important features to be used in building the tree, and only four were ignored, so the total number of influencing features became 21.

A Gini index is use to create data dividing points and start the decision trees building process.

- **Similarity measuring** and **building vector: -**

In this stage, user reviews, which have been processed in both the first and second stage, will be treated, as the rating of these reviews will be fetched from the main database.

This is done by comparing the new comments of users with the comments in the main data set, where the form will compare each comment separately and fetch the classification of the ‘data set comments that are similar to the user comments by a high percentage. This is done using the Sequence Matcher Tool. This tool searches for the longest chain to match between new user comments and comments classified in the data set, and also calculates and returns the similarity ratio according to the method, ratio = 2.0 * M / T, where M = matches , T = total number of elements in both sequence.

The number of classifications obtained through the above process for each intention classification and Topic be calculated, and the results of each category will be stored, separately. Then the rating rate for each subject intention is calculated. The number of new comments obtained after processing long comments will divide the total of similar rankings. At the end of that, a vector of 21 values is created, from which to test the quality of the application in the decision tree .Figure 3.4

In this stage, the application is tested for knowing whether it contains problems in its performance or not. In the event that the application contains problems, the form will print all the comments that indicate a problem or a defect in the performance of the application, as well as comments that indicate requests to add certain features.

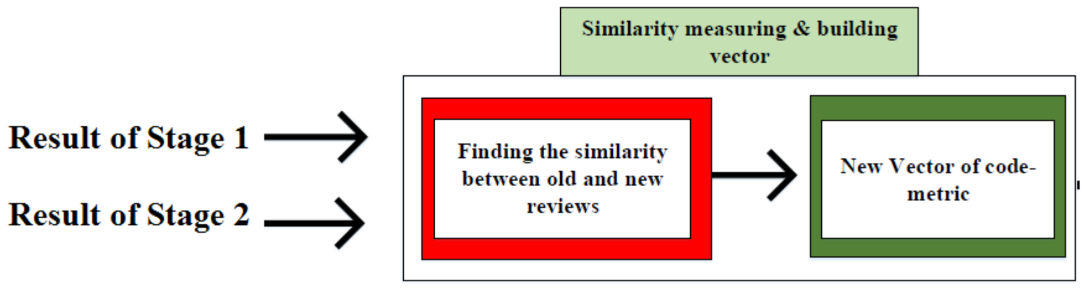

Similarity measuring & building vector
Result of Stage 1
Result of Stage 2
Finding the similarity between old and new reviews
New Vector of code-metric

Figure 3.4 Similarity measuring and *building vector*

3.3 Summary

This chapter includes a comprehensive description of the four main stages of the model. In the first stage, which is the process of building and training the decision tree based on the results found in the sixth section of the data set, which includes the results of the ratio between the two main categories ”Intention and Topic” of all Classification categories. In the second stage of the form, that includes two main processes, the first is to delete unhelpful comments, and the second process divided dataset into twenty parts with different lengths. In the third stage, user's feedback on the performance of Android applications will be dealt with; collect all information related to reviews and calculate the average star rating. Long comments are also processed and converted into short, easy-to-understand phrases. In addition, if the reviews are written in a language other than English, they will be translated into English. Deal with the fourth stage, the results of the first and second stages. By comparing new user comments with comments in the main dataset, the model will compare each comment separately and bring the rating of similar dataset comments with user comments at a high rate. This is done using the Sequence Matching Tool. The rating is then calculated for each intention and for each topic separately. Then the application is tested for finding out whether it has performance problems or not. In the event that the application contains problems, the form will print all comments indicating a problem or defect in the performance of the application, as well as comments that indicate requests to add certain features.

Chapter Four

The Results

CHAPTER FOUR
THE RESULTS

4.1 Overview

This chapter includes a full description of all attempts and steps that contributed to building the model, where each stage will be explained according to the general plan of the model, and this will be enhanced by adding pictures of stages and steps for the model. In order to describe all procedures and accounts in the project. It also describes the comparisons that occurred during the work of the project, and an explanation of the ways in which the final results were calculated to achieve the desired goal, which is to help software engineers to improve application performance, save costs and reduce time. In the beginning of the previous chapter, we focused on the need to adhere to the tasks of software engineering because they contribute significantly to achieving the goal of software engineers and building successful applications. When starting the design stage, many algorithms, techniques, and designs were tried to achieve the goal, and at the end of that, two models were made with the aim of obtaining the best results. The first model was built using automated “back propagation Neural” algorithms, as this model relies on negative comments about applications in stores . In addition, take advantage of them to discover errors in the applications. The second model, which will be adopted, was built using the decision tree algorithm . Decision tree are well-known machine learning technique for solving complex classification problems. Software engineering tasks were also strictly adhered to for the purpose of saving effort, time and cost. The model also aims to facilitate the understanding of the large and increasing number of user reviews about applications and discover the largest possible number of performance problems, and benefit from them in avoiding frequent maintenance and focus on tools that help in creating and developing applications while providing the required features by users quickly, with a view to raising application success rates and achieve the goal of software engineers. In this chapter, the work steps in each of the two models above will be explained

4.2 Initial attempts and steps to build the model

In this project, we will focus on some software engineering tasks such as testing, maintenance and development, with the purpose of building a model that contributes to the rapid completion of these tasks. Initially, many tools and algorithms were tried to construct the model. Where the **Keras** Package, one of the deep learning tools found in the Python programming language, has been tried out, and it is one of the high-level Natural Network algorithms. Where the model was built according to the steps of the Keras CallBacks, which are characterized by storing the resulting weights when training the model, as this method focuses on recalling the weight responsible for the best accuracy of the model to take advantage of when testing. Then, twenty-one "column" entries were specified for the data set. And allocate (80%) for training and (20%) to examine the model, with a frequency of 150 epoch. Only the designed model was examined from two layers and It was the highest result of accuracy (0,70635%), the number of layers was increased to reach the number to six layers in order to increase accuracy. Accuracy became (0,73810%, 0,74603%, 0,72222% and 0,69841%). These results are not considered high. So, another attempt was made to increase accuracy, which is to use the learning rate in the training process, despite the use of the learning rate, but the accuracy of the system is still not high. the results of accuracy (0,7308%). Therefore, it became necessary to search for other methods and methods and experiment with new algorithms for the purpose of raising the accuracy rate.

Another model was designed for the project, we named it the first model. This model relies on all categories of negative comment classification (both Intention and Topic categories) that indicate a problem and benefit from it in building a model that detects applications that contain malfunctions. Where the first model will be explained in a simplified manner, with clarification of all steps and stages of the model, according to the figure below. The first model consists of four steps: -

1- Pre-Processing stage.
2- Analysis stage.
3- Training stage.
4- Test stage and the evaluation of the results.

Since our project aims to detect errors in applications, we have focused on the categories that represent negative comments and indicate a malfunction in the application or one of its functions, and on this basis the model was trained by the backpropagation neural network algorithm and the result was (75,67). This result is low and does not rise to the level of system performance, but this accuracy increased to (92.7) after adding and classifying more comments and increasing the training process to 10,000 times. After that, the model became able to identify applications that have problems and need maintenance or updating (a problem with functions or the performance of some tasks ... etc.). Figure in Appendix C

However, the first modelis unable to determine the type of defect, so it became necessary to search for another way to diagnose Errors.

4.3 Main stages of the Model

The model consists of four main stages: -

1- Pre-Processing stage of Dataset reviews
2- Pre-Processing stage of new reviews and star rating.
3- Decision Tree classifier Model
4- Similarity measuring and building vector

Each of the above-mentioned stages plays an important role in diagnosing the largest possible number of application problems (malfunctions in the application or one of its functions), with the aim of expediting maintenance and adding what users want in the applications. At each stage of the model, one of the six parts of the data set is dealt with. All Android app versions collected from F-Droid and Google Play Store are dealt with in the first part of database. In addition, the second part that includes all comments on Android apps, in addition to rating comments according to the Intent and Topic. As well as the part that contains data about the categories and errors in each application. Calculated according to the standard criteria for testing Android applications. As well as full dependence on data in the sixth part of the database.

4.4 Program start interface: -

The main form interface contains all the commands and features of the form responsible for dealing with the data set, processing it and displaying the results by implementing the four main steps. The Python interface provides all of that easily and smoothly.

4.5 Final Results for each stage of the model

The proposed model consists of five main stages. These stages include many processing, recall and calculation processes to obtain the results. Below is a result of each these stages.

4-5-1 Pre-Processing stage of Dataset reviews

This stage consists of two main processes, the first is to delete all duplicate and unwanted comments, and the second process is to split the database. Where three methods were tried for completing this stage in a short time, and the completion time for each of them was calculated separately, as the first method included completing this stage without dividing the data set. While the remaining two experiments included dividing, the data set resulting from the first process into several parts. The second experiment included how to sort all the comments in the database and divide them into four groups, according to length. The first part is for comments 1 to 19 characters long. While the second part is dedicated to comments ranging from 20 to 39 characters. Comments that are 40 to 69 characters long are stored in Part 3. For the last part, it is intended for comments that are 70 or more characters long. At the end of that, we get four parts of the databases, and each part contains comments of similar lengths, after calculating and storing the final results, the outcome of the first part of (81247) comment, the second part of (94318) comment, the third part of (79144) comment, and the fourth part (75872) comment. Figure 3.5

In addition, the third method dividing the data set into 20 parts Table 3.3. The test results were as in Table 4.2. As this table includes the results of testing 40 applications and trying, the three methods mentioned above with the time required to test each application. The table also contains the number of important and influential comments for both the intention category and the Topic category from which construction and maintenance officials can benefit from applications in the future, as all unimportant comments are also removed at this stage.

From the above, its note that the third method helped to accelerate the achievement of results, with the same accuracy as in the first and second experiments, but with a large time difference. In this model, the third method of pre-treatment was used, as it takes less time.

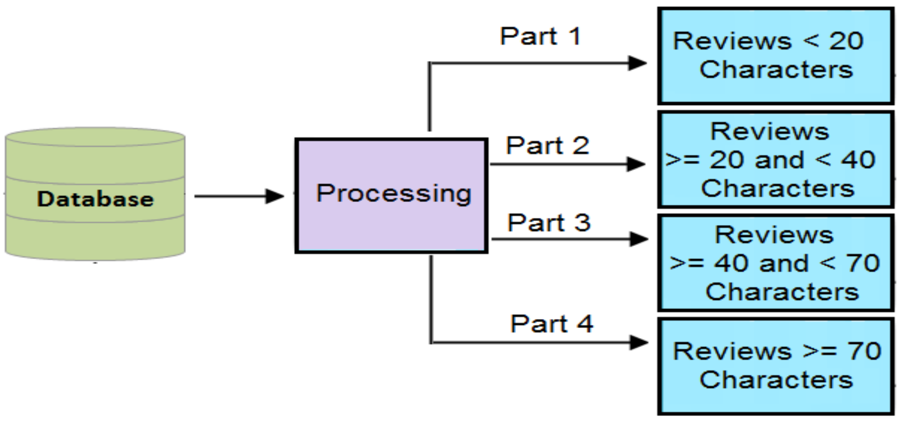


Figure 3.5 Split the data set.

4-5-2 Pre-Processing stage of new reviews and star rating

At this stage. Inputs is user reviews about apps in app download stores. These comments are user feedback on the performance of the Android application, where the model prepares these reviews to test the application, calculate the star rating rate, convert long sentences into short sentences, and convert comments to English if it is not written in this language.

4-5-3 Decision Tree classifier Model

Each the inputs to this stage are the results found in the sixth part of the data set, which includes the results of ratios between the main categories of both intention and subject. In addition to the star rating, the number of user comments, and the number of sentences in these comments. Initially, each feature was calculated and its degree of influence on model classification was measured. The results are as in Table 4.1. The data was then divided into a "training group and a test group." Then, several experiments were conducted for training with the aim of obtaining the highest accuracy, as shown in the explanation below:

- First experience: - When 60% is allocated to training and 40% is for testing, the accuracy is 85%.
- Second experience: - When 70% of the data set is allocated for training and 30% of it is for testing, accuracy is (89%).
- Third experience: - When (80%) is allocated to training and (20%) is for testing, the accuracy is (92%- to 93%).

For "random state", its value = 1

The third training result was approved in the project, because she got the best result

At the end of this stage, the construction of the decision tree model has finished. See Figure 3.6

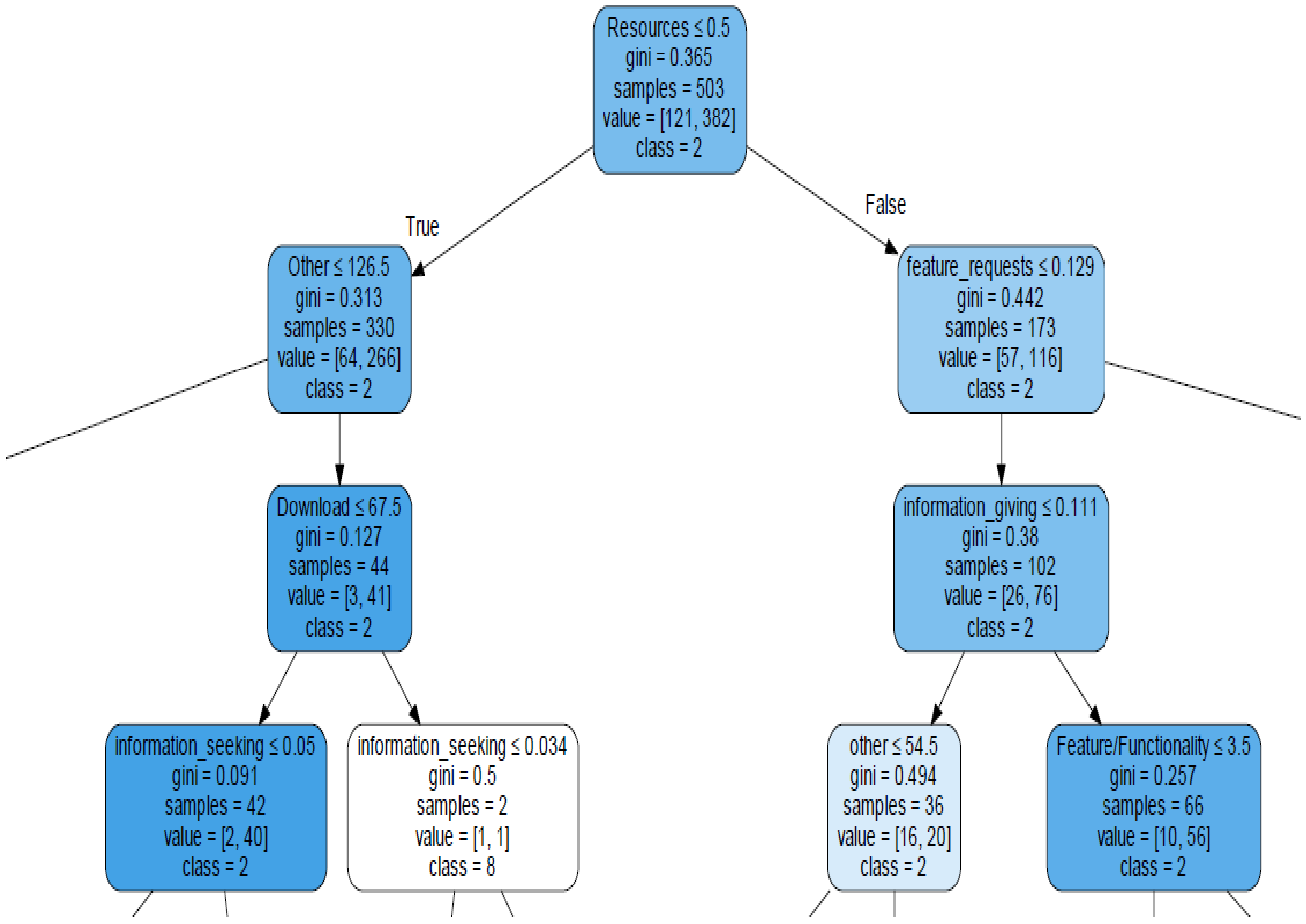


Figure 3.6 sub data from decision tree diagram.

Table 4.1 Results of the impact measure for each feature of the data set

n	Feature name	Degree of impact
1.	average rating	0.05106545
2.	total reviews	0.05349558
3.	total_sentences	0.05267467
4.	feature_requests	0.06806801
5.	problem_ discoveries	0.05105385
6.	information_seeking	0.04430142
7.	information_giving	0.05665265
8.	other	0.04630152
9.	App	0.05106545
10.	GUI	0.05349558
11.	Contents	0.05267467
12.	Pricing	0.06806801
13.	Feature or Functionality	0.05105385
14.	Improvement	0.04430142
15.	Updates/ Versions	0.05665265
16.	Resources	0.04630152
17.	Security	0.04509589
18.	Download	0.04808971
19.	Model	0.04944204
20.	Company	0.03572894
21.	Other	0.04628112
	Total	Summation = 1.0

Table 4.2 the results of the three experiments to measure the time required for the test

n	App-name	Number of comments	The number of sentences in the comment	The time required for testing if the dataset consists of 1 part, 4 parts or 20 parts.			Important Comments		
				1 Part	4 Parts	20 Parts	Intention Comment	Topic Comment	All
1.	526	10	14	00:06:28	00:01:34	00:00:28	2	8	8
2.	886	10	16	00:10:28	00:03:52	00:00:58	2	11	11
3.	1663	11	16	00:11:22	00:02:36	00:00:42	1	9	9
4.	1690	13	16	00:05:01	00:01:27	00:00:50	3	11	11
5.	179	16	16	00:11:51	00:03:20	00:00:58	9	16	16
6.	1198	12	18	00:10:43	00:02:28	00:00:39	5	9	10
7.	97	16	16	00:06:10	00:01:31	00:00:32	1	11	11
8.	1678	12	20	00:12:07	00:02:53	00:00:51	5	11	11
9.	809	10	18	00:13:07	00:04:37	00:01:24	5	15	15
10.	1976	16	16	00:11:00	00:04:00	00:01:00	9	16	16
11.	1995	10	21	00:06:30	00:01:10	00:00:35	9	13	15
12.	708	10	22	00:29:48	00:05:58	00:01:48	8	19	20
13.	1783	11	17	00:28:04	00:05:18	00:02:35	2	12	12
14.	1334	12	21	00:18:26	00:04:38	00:01:17	5	18	19
15.	1171	13	21	00:10:44	00:02:43	00:00:48	0	12	12
16.	845	13	22	00:10:45	00:02:57	00:00:59	6	16	16
17.	1960	20	45	00:18:30	00:06:00	00:01:45	15	32	34
18.	2044	20	48	00:34:05	00:06:00	00:02:40	15	34	36
19.	900	25	35	00:17:25	00:04:15	00:01:18	4	22	22
20.	1968	28	56	00:35:45	00:12:41	00:02:40	9	44	45
21.	653	29	58	00:50:40	00:08:15	00:04:30	19	37	37
22.	1344	30	59	00:36:40	00:12:55	00:03:52	11	42	42
23.	1042	32	63	00:58:10	00:10:47	00:05:45	14	49	49
24.	100	41	55	00:50:11	00:16:31	00:03:00	5	21	21
25.	867	52	59	00:19:43	00:05:14	00:01:39	8	35	35
26.	1527	102	133	00:32:30	00:05:57	00:04:43	12	70	70
27.	1526	152	247	02:13:46	00:42:52	00:12:00	35	153	154
28.	1983	195	350	04:05:12	01:30:20	00:21:02	86	164	248
29.	1338	248	424	02:47:08	01:01:58	00:22:34	105	291	300
30.	507	293	399	02:12:08	00:52:20	00:15:50	117	259	271
31.	1496	359	510	05:20:13	01:34:24	00:32:57	100	306	315
32.	1136	400	766	05:48:22	02:18:34	00:41:46	210	584	598
33.	642	448	549	06:00:08	00:50:07	00:27:32	95	224	237
34.	916	507	699	05:59:13	01:45:37	00:27:35	85	363	372
35.	1785	561	1070	09:13:50	03:29:45	01:01:34	463	737	758
36.	1135	608	1194	11:12:17	05:11:40	01:32:29	342	908	929
37.	1364	702	989	05:55:08	02:05:46	00:48:44	114	480	496
38.	1465	739	1252	06:48:19	02:57:35	01:45:27	244	799	817
39.	1568	806	1364	08:40:08	03:10:30	00:48:21	255	806	827
40.	2032	932	1888	14:12:55	08:50:10	01:20:11	400	1364	1387

Notice

All the experiments mentioned in the above table were performed on a laptop type computer with specifications / Core i7 microprocessors, Ram 8 GB, Hard Disk 500 GB

4-5-4 Similarity measuring and building vector

Inputs for Inputs for this stage will be user comments that processed in second stage, as this process involved categorizing new user comments into Intention and Topics categories. The model was trained on a portion of the data set (20,210 from 330,581 commends) in order to classify user comments by intention. By using the Sklearn package within the Python programming language and four different algorithms were tried for training the model. These are SVM, Naive Bayes, Logistic Regression, and Random Forest. The accuracy results are as follows: -

The accuracy score of SVM is 0.7294676429843657

The accuracy score of Naive Bayes is 0.7158123886799921

The accuracy score of Logistic Regression is 0.7361963190184049

The accuracy of Random Forest is between 0.5893528596873144 and 0,6406095388877895 , after completing this attempt; we see that the accuracy of the results is not high, so it is better to rely on the results of the team that published the data set. The results of classification accuracy were for the categories of intentions (ranging between 84% and 89%) and for the Topic categories (76%), new User Review ratings will be fetched from Pre-categorized User Reviews by intention and subject in the main database. In addition, link each new comment with the appropriate classification. By relying on the scale of similarity between text strings (to finding the greatest similarity between new user reviews and classified reviews in the database) where many measures such as (Jaccard Similarity, Cosine similarity, Jaro-Winkler,The Levenshtein distance,Sequence Matcher) have been tried .At the end . The Sequence Matcher scale has proven the efficiency and speed of the model.

At this end, a test was conducted for 20 Android applications were randomly chosen sixteen of them contained problems and four without problems. This has been confirmed by the results of the manual examination of these applications with known accuracy measures and as found in the fourth part of the data set that contains the name of the application and the type of error detected by these standards. After testing the applications, it was found that eighteen from it had been they are correctly categorized and only two result are incorrect. Table 4.3. Which means that the model has 90% succeeded in testing, and that the model is able to detect errors and problems in the application based on user feedback, as well as quickly sort and display user requests.

n.	App-name	Rite - Class	Model-class	Prediction result
1.	526	1	1	Right
2.	1976	1	0	Wrong
3.	886	1	1	Right
4.	1663	1	1	Right
5.	1690	1	1	Right
6.	179	1	1	Right
7.	1780	0	1	Wrong
8.	1992	0	0	Right
9.	2033	0	0	Right
10.	1198	1	1	Right
11.	97	1	1	Right
12.	1678	1	1	Right
13.	809	1	1	Right
14.	1171	1	1	Right
15.	1334	1	1	Right
16.	1783	1	1	Right
17.	708	1	1	Right
18.	724	0	0	Right
19.	845	1	1	Right
20.	1995	1	1	Right

Table 4.3 Results of Test Model

4.6 Summary

This chapter includes the results of the four main stages of the model. In addition to describing the inputs for each stage of the model, this chapter also examines the results of the training process, where the results of the training were between (92% to 93%). However, this chapter included schedules of application completion time according to the number of parts of the data set, as well as the results of testing 20 randomly chosen Android applications. As the accuracy of the test results reached (90%) by 18 applications, they were correctly classified and only two did not succeed in classifying. Which showed us that the designed model is able to diagnose applications with errors and present their problems to application officials to address them for the success of this application

Chapter Five

Conclusion and Future works

CHAPTER FIVE
CONCLUSION AND FUTURE WORKS

5.1 Overview

After completing all the stages of our project and getting the results, we present in this chapter a simple summary of the project with some useful points and possible work in the future, in order to make the most of the idea of the project.

5.2 Conclusion

The most important characteristics that were discovered through implementing the proposed methodology and discussing the results are that it can be confirmed that the user's comments on the application's performance are among the important criteria in evaluating the application. Especially with regard to the features provided by developers after the process of maintenance or application development, which helps developers greatly to understand what users demand, and maximize Benefit from the app, where user feedback is characterized by the users style and monitoring of app performance, because the user is the beneficiary and influential in the success or failure of the application. Therefore, it is necessary to involve the views of users in the application building process as well as after and before performing maintenance and updating of applications, for creating applications that, meet the growing aspirations and demands of users. As well, as contribute to reduce the time, cost and effort spent by software engineers in the maintenance and development of Android applications.

5.3 Future Works

Some potential work to help test Android apps and increase system accuracy.

- Building a larger data set to include more rankings, to contribute to the classification of new comments about applications. In addition, increase the accuracy of the system.

- Reclassification of category (Other) comments in both the intention and Topic tables, and classify them into one of the main categories in order to use them to increase number of comments .

- Find a mechanism to diagnose feedback from specialists and people with experience in the performance of Android apps for the purpose of relying heavily on their feedback and increasing system accuracy.

REFERENCES

[1] N. J.-G. J. of C. S. and and U. 2019, “A Review of Mobile Application Development in the Agile Software Development Environment,” *Computerresearch.Org*, vol. 19, no. 1, pp. 0–4, 2019.

[2] J. Amedie, “The impact of social media on society,” 2015.

[3] K. Biswas and V. Muthukkumarasamy, “Securing smart cities using blockchain technology, ” in *2016 IEEE 18th international conference on high performance computing and communications; IEEE 14th international conference on smart city; IEEE 2nd international conference on data science and systems (HPCC/SmartCity/DSS),* 2016, pp. 1392–1393.

[4] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, “A survey of app store analysis for software engineering,” *IEEE Trans. Softw. Eng.*, vol. 43, no. 9, pp. 817–847, 2016.

[5] S. McIlroy, N. Ali, and A. E. Hassan, “Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store,” *Empir. Softw. Eng.*, vol. 21, no. 3, pp. 1346–1370, 2016.

[6] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile software development methods: Review and analysis,” *arXiv Prepr. arXiv1709.08439*, 2017.

[7] V. N. Inukollu, D. D. Keshamoni, T. Kang, and M. Inukollu, “Factors influencing quality of mobile apps: Role of mobile app development life cycle,” *arXiv Prepr. arXiv1410.4537*, 2014.

[8] G. Hecht, O. Benomar, R. Rouvoy, N. Moha, and L. Duchien, “Tracking the software quality of android applications along their evolution (t),” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 236–247.

[9] L. Azevedo, A. Dantas, and C. G. Camilo-Junior, “DroidBugs: An Android Benchmark for Automatic Program Repair,” *arXiv Prepr. arXiv1809.07353*, 2018.

[10] G. Hecht, R. Rouvoy, N. Moha, and L. Duchien, “Detecting Antipatterns in Android Apps To cite this version : Detecting Antipatterns in Android Apps,” 2015.

[11] H. K. Flora, X. Wang, and S. V. Chande, “An Investigation on the Characteristics of Mobile Applications: A Survey Study,” *Int. J. Inf. Technol. Comput. Sci.*, vol. 6, no. 11, pp. 21–27, 2015.

[12] P. Clarke, R. V O’Connor, and B. Leavy, “A complexity theory viewpoint on the software development process and situational context,” in *Proceedings of the International Conference on Software and Systems Process*, 2016, pp. 86–90.

[13] D. L. Amoroso and Y. A. N. Chen, “Constructs Affecting Continuance intention in consumers with mobile financial apps : A dual factor approach,” *J. Inf. Technol. Manag.*,

vol. XXVIII, no. 3, 2017.

[14] M. Nagappan and E. Shihab, "Future trends in software engineering research for mobile apps," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, vol. 5, pp. 21–32.

[15] J. O. Kephart, "Research challenges of autonomic computing," no. January 2005, p. 15, 2005.

[16] M. Nayebi, B. Adams, and G. Ruhe, "Release Practices for Mobile Apps--What do Users and Developers Think?," in *2016 ieee 23rd international conference on software analysis, evolution, and reengineering (saner)*, 2016, vol. 1, pp. 552–562.

[17] S. A. Licorish, A. Tahir, M. F. Bosu, and S. G. MacDonell, "On Satisfying the Android OS Community: User Feedback Still Central to Developers' Portfolios," pp. 78–87, 2015.

[18] A. Yamashita, "Experiences from performing software quality evaluations via combining benchmark-based metrics analysis, software visualization, and expert assessment," *2015 IEEE 31st Int. Conf. Softw. Maint. Evol. ICSME 2015 - Proc.*, pp. 421–428, 2015.

[19] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," in *2015 IEEE international conference on software maintenance and evolution (ICSME)*, 2015, pp. 281–290.

[20] A. Di Sorbo, S. Panichella, C. V Alexandru, C. A. Visaggio, and G. Canfora, "SURF: summarizer of user reviews feedback," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 55–58.

[21] N. Nascimento, C. Lucena, P. Alencar, and D. Cowan, "Software Engineers vs. Machine Learning Algorithms: An Empirical Study Assessing Performance and Reuse Tasks," pp. 1–22, 2018.

[22] O. Hidmi and B. E. Sakar, "Software Development Effort Estimation Using Ensemble Machine Learning," *Int. J. Comput. Commun. Instrum. Eng.*, vol. 4, no. 1, pp. 1–5, 2017.

[23] G. Grano, A. Di Sorbo, F. Mercaldo, C. A. Visaggio, G. Canfora, and S. Panichella, "Android apps and user feedback: a dataset for software evolution and quality improvement," no. ii, pp. 8–11, 2017.

[24] N. Korfiatis, P. Stamolampros, P. Kourouthanassis, and V. Sagiadinos, "Measuring service quality from unstructured data: A topic modeling application on airline passengers' online reviews," *Expert Syst. Appl.*, vol. 116, pp. 472–486, 2019.

[25] A. Di Sorbo *et al.*, "What would users change in my app? summarizing app reviews for recommending software changes," in *Proceedings of the 2016 24th ACM SIGSOFT*

International Symposium on Foundations of Software Engineering, 2016, pp. 499–510.

[26] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2016, pp. 14–24.

[27] G. Hecht, N. Moha, and R. Rouvoy, "An empirical study of the performance impacts of Android code smells," pp. 59–69, 2016.

[28] L. Wei, Y. Liu, and S.-C. Cheung, "Taming Android fragmentation: characterizing and detecting compatibility issues for Android apps," pp. 226–237, 2016.

[29] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 91–102.

[30] J. Mitra and V.-P. Ranganath, "Ghera: A Repository of Android App Vulnerability Benchmarks," 2017.

[31] D. R. Thomas, A. R. Beresford, and A. Rice, "Security Metrics for the Android Ecosystem," pp. 87–98, 2015.

[32] B. Reaves *et al.*, "Mo(bile) Money , Mo(bile)Problems : Analysis of Branchless Banking Applications in the Developing World," *Usenix*, vol. 20, no. 3, pp. 1–31, 2017.

[33] D. Savchenko, T. Hynninen, and O. Taipale, "Code quality measurement: Case study," *2018 41st Int. Conv. Inf. Commun. Technol. Electron. Microelectron. MIPRO 2018 - Proc.*, pp. 1455–1459, 2018.

[34] H. Hu, C. P. Bezemer, and A. E. Hassan, "Studying the consistency of star ratings and the complaints in 1 & 2-star user reviews for top free cross-platform Android and iOS apps," *Empir. Softw. Eng.*, vol. 23, no. 6, pp. 3442–3475, 2018.

[35] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. De Lucia, "Detecting code smells using machine learning techniques: are we there yet?," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 612–621.

[36] E. Fernandes, J. Oliveira, G. Vale, T. Paiva, and E. Figueiredo, "A review-based comparative study of bad smell detection tools," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 2016, p. 18.

[37] V. Guna and M. S. Kumar, "A Survey on Software Code Clone Detection to Improve the Maintenance Effort and Maintenance Cost of the Software," *Int. J. Comput. Sci. Eng.*, vol. 06, no. 03, pp. 188–192, 2018.

[38] N. M. Devadiga, “Software engineering education: Converging with the startup industry,” in *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, 2017, pp. 192–196.

[39] M. P. S. Bhatia, A. Kumar, and R. Beniwal, “Ontologies for software engineering: Past, present and future,” *Indian J. Sci. Technol.*, vol. 9, no. 9, pp. 1–16, 2016.

[40] A. Basiri *et al.*, “Chaos engineering,” *IEEE Softw.*, vol. 33, no. 3, pp. 35–41, 2016.

[41] C. Rosenthal, L. Hochstein, A. Blohowiak, N. Jones, and A. Basiri, *Chaos Engineering*. O’Reilly Media, Incorporated, 2017.

[42] M. R. d. A. Souza, L. Veado, R. T. Moreira, E. Figueiredo, and H. Costa, “A systematic mapping study on game-related methods for software engineering education,” *Inf. Softw. Technol.*, vol. 95, pp. 201–218, 2018.

[43] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC press, 2014.

[44] M. A. Ardis, D. Budgen, G. W. Hislop, J. Offutt, M. J. Sebern, and W. Visser, “SE 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.,” *IEEE Comput.*, vol. 48, no. 11, pp. 106–109, 2015.

[45] T. R. Shaffer, J. L. Wise, B. M. Walters, S. C. Müller, M. Falcone, and B. Sharif, “itrace: Enabling eye tracking on software artifacts within the ide to support software engineering tasks,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 954–957.

[46] C. Marshall, P. Brereton, and B. Kitchenham, “Tools to support systematic reviews in software engineering: a feature analysis,” in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, p. 13.

[47] K. K. Silveira and R. Prikladnicki, “A systematic mapping study of diversity in software engineering: a perspective from the agile methodologies,” in *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2019, pp. 7–10.

[48] R. L. Nord and J. E. Tomayko, “Software architecture-centric methods and agile development,” *IEEE Softw.*, vol. 23, no. 2, pp. 47–53, 2006.

[49] G. Look, S. Peters, and H. Shrobe, “Plan-driven ubiquitous computing,” in *Student Oxygen Workshop*, 2003.

[50] T. Dyba and T. Dingsoyr, “What do we know about agile software development?,” *IEEE Softw.*, vol. 26, no. 5, pp. 6–9, 2009.

[51] D. F. Rico and H. H. Sayani, “Use of agile methods in software engineering education,” in

2009 Agile Conference, 2009, pp. 174–179.

[52] B. Boehm and R. Turner, “Using risk to balance agile and plan-driven methods,” *Computer (Long. Beach. Calif).*, vol. 36, no. 6, pp. 57–66, 2003.

[53] T. Imani, M. Nakano, and V. Anantatmula, “Does a Hybrid Approach of Agile and Plan-Driven Methods Work Better for IT System Development Projects?,” *development*, vol. 1, no. 2, p. 3, 2017.

[54] M. S. Swetha, “Software Engineering notes,” 2018.

[55] M. Kramer, “Best practices in systems development lifecycle: An analyses based on the waterfall model,” *Rev. Bus. Financ. Stud.*, vol. 9, no. 1, pp. 77–84, 2018.

[56] A. M. Dima and M. A. Maassen, “From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management,” *J. Int. Stud.*, vol. 11, no. 2, pp. 315–326, 2018.

[57] A. Armas-Cervantes, N. R. T. P. van Beest, M. La Rosa, M. Dumas, and S. Raboczi, “Incremental and interactive business process model repair in apromore,” 2017.

[58] W. Scacchi, “Process models in software engineering,” *Encycl. Softw. Eng.*, 2002.

[59] R. S. Pressman, “Software Engineering: A Practitioners’ Approach Sition.” ANDI. Yogyakarta, 2010.

[60] A. Alshamrani and B. Abdullah, “A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model,” *Int. J. Comput. Sci. Issues*, vol. 12, no. 1, pp. 106–111, 2015.

[61] J. Dick, E. Hull, and K. Jackson, *Requirements engineering*. Springer, 2017.

[62] M. Batra and A. Bhatnagar, “A Comparative Study of Requirements Engineering Process Model.,” *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 3, 2017.

[63] J. Elijah, A. Mishra, E. M. C. Udo, A. Abdulganiyu, and A. Musa, “Survey on Requirement Elicitation Techniques: It’s Effect on Software Engineering,” 2017.

[64] R. Ellis-Braithwaite, R. Lock, R. Dawson, and T. King, “Repetition between stakeholder (user) and system requirements,” *Requir. Eng.*, vol. 22, no. 2, pp. 167–190, 2017.

[65] B. Berenbach, D. Paulish, J. Kazmeier, and A. Rudorfer, *Software & systems requirements engineering: in practice*. McGraw-Hill, Inc., 2009.

[66] D. Beyer, “Software verification with validation of results,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2017, pp. 331–349.

[67] N. Anwar and S. Kar, “Review Paper on Various Software Testing Techniques & Strategies,” *Glob. J. Comput. Sci. Technol.*, 2019.

[68] J. Melegati, A. Goldman, F. Kon, and X. Wang, “A model of requirements engineering in

software startups,” *Inf. Softw. Technol.*, vol. 109, pp. 92–107, 2019.

[69] S. Marsland, *Chapman & Hall/CRC Machine Learning & Pattern Recognition Series Chapman & Hall/CRC Machine Learning & Pattern Recognition Series M AC H I N E LEARNING An Algorithmic Perspective*. 2014.

[70] T. O. Ayodele, “Types of machine learning algorithms,” *New Adv. Mach. Learn.*, pp. 19–48, 2010.

[71] R. C. Deo, “Machine learning in medicine,” *Circulation*, vol. 132, no. 20, pp. 1920–1930, 2015.

[72] R. P. Bunker and F. Thabtah, “A machine learning framework for sport result prediction,” *Appl. Comput. informatics*, vol. 15, no. 1, pp. 27–33, 2019.

[73] M. Kubat, *An introduction to machine learning*, vol. 2. Springer, 2017.

[74] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, “A survey on deep transfer learning,” in *International conference on artificial neural networks*, 2018, pp. 270–279.

[75] G. Litjens *et al.*, “A survey on deep learning in medical image analysis,” *Med. Image Anal.*, vol. 42, pp. 60–88, 2017.

[76] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[77] I. Portugal, P. Alencar, and D. Cowan, “The use of machine learning algorithms in recommender systems: A systematic review,” *Expert Syst. Appl.*, vol. 97, pp. 205–227, 2018.

[78] W. Du and Z. Zhan, “Building decision tree classifier on private data,” in *Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14*, 2002, pp. 1–8.

[79] M. Shouman, T. Turner, and R. Stocker, “Using decision tree for diagnosing heart disease patients,” in *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*, 2011, pp. 23–30.

[80] A. Developers, “What is android.” Android Developers, http://developer. android. com/guide/basics/what-is …, 2011.

[81] C. Wang, W. Duan, J. Ma, and C. Wang, “The research of Android System architecture and application programming,” in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, 2011, vol. 2, pp. 785–790.

[82] S. Brahler, “Analysis of the android architecture,” *Karlsruhe Inst. Technol.*, vol. 7, no. 8, 2010.

[83] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, “What are the characteristics of high-

rated apps? a case study on free android applications," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 301–310.

[84] U. A. Mannan, I. Ahmed, R. A. M. Almurshed, D. Dig, and C. Jensen, "Understanding code smells in android applications," in *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2016, pp. 225–236.

[85] V. Arnaoudova, M. Di Penta, and G. Antoniol, "Linguistic antipatterns: what they are and how developers perceive them," *Empir. Softw. Eng.*, vol. 21, no. 1, pp. 104–158, 2016.

[86] S. Habchi, G. Hecht, R. Rouvoy, and N. Moha, "Code Smells in iOS Apps: How Do They Compare to Android?," *Proc. - 2017 IEEE/ACM 4th Int. Conf. Mob. Softw. Eng. Syst. MOBILESoft 2017*, pp. 110–121, 2017.

[87] F. A. Fontana, M. Mangiacavalli, D. Pochiero, and M. Zanoni, "On experimenting refactoring tools to remove code smells," pp. 1–8, 2015.

[88] M. Di Penta, F. Palomba, R. Oliveto, G. Bavota, A. De Lucia, and D. Poshyvanyk, "Mining Version Histories for Detecting Code Smells," *IEEE Trans. Softw. Eng.*, 2014.

[89] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk, "Detecting bad smells in source code using change history information," *2013 28th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2013 - Proc.*, pp. 268–278, 2013.

[90] G. Hecht, R. Rouvoy, N. Moha, and L. Duchien, "Detecting antipatterns in android apps," in *2015 2nd ACM international conference on mobile software engineering and systems*, 2015, pp. 148–149.

[91] S. Habchi, "Understanding Mobile-Specific Code Smells." Université de Lille, 2019.

[92] B. Mason, L. Jain, and R. Nowak, "Learning low-dimensional metrics," in *Advances in neural information processing systems*, 2017, pp. 4139–4147.

[93] F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "Lightweight detection of Android-specific code smells: The aDoctor project," *SANER 2017 - 24th IEEE Int. Conf. Softw. Anal. Evol. Reengineering*, pp. 487–491, 2017.

[94] S. Kaur and R. Maini, "Analysis of various software metrics used to detect bad smells," *Int J Eng Sci*, vol. 5, no. 6, pp. 14–20, 2016.

[95] A. Kaur and G. Dhiman, "A review on search-based tools and techniques to identify bad code smells in object-oriented systems," in *Harmony search and nature inspired optimization algorithms*, Springer, 2019, pp. 909–921.

[96] N. Padhy, R. P. Singh, and S. C. Satapathy, "Utility of an object-oriented metrics component: examining the feasibility of. Net and C# object-oriented program from the perspective of mobile learning," *Int. J. Mob. Learn. Organ.*, vol. 12, no. 3, pp. 263–279,

2018.

[97] S. Yang, D. Yan, and A. Rountev, "Testing for poor responsiveness in Android applications," in *2013 1st international workshop on the engineering of mobile-enabled systems (MOBS)*, 2013, pp. 1–6.

[98] F. Mercaldo, A. Di Sorbo, C. A. Visaggio, A. Cimitile, and F. Martinelli, "An exploratory study on the evolution of android malware quality," *J. Softw. Evol. Process*, vol. 30, no. 11, p. e1978, 2018.

[99] M. Allahyari *et al.*, "A brief survey of text mining: Classification, clustering and extraction techniques," *arXiv Prepr. arXiv1707.02919*, 2017.

[100] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView IDC Anal. Futur.*, vol. 2007, no. 2012, pp. 1–16, 2012.

[101] S. V. Gaikwad, A. Chaugule, and P. Patil, "Text mining methods and techniques," *Int. J. Comput. Appl.*, vol. 85, no. 17, 2014.

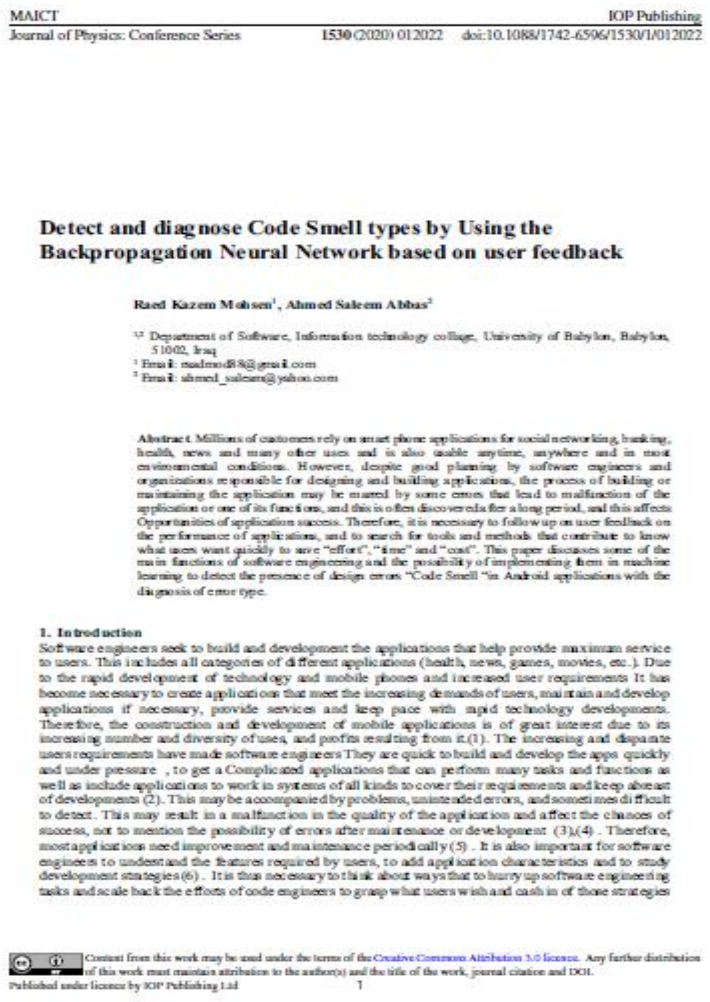
[102] R. Talib, M. K. Hanif, S. Ayesha, and F. Fatima, "Text mining: techniques, applications and issues," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 11, pp. 414–418, 2016.

[103] A. Kao and S. R. Poteet, *Natural language processing and text mining*. Springer Science & Business Media, 2007.

[104] S. Sun, C. Luo, and J. Chen, "A review of natural language processing techniques for opinion mining systems," *Inf. fusion*, vol. 36, pp. 10–25, 2017.

Appendix A

time



MAICT IOP Publishing
Journal of Physics: Conference Series 1530 (2020) 012022 doi:10.1088/1742-6596/1530/1/012022

Detect and diagnose Code Smell types by Using the Backpropagation Neural Network based on user feedback

Raed Kazem Mohsen[1], Ahmed Saleem Abbas[2]

[1,2] Department of Software, Information technology collage, University of Babylon, Babylon, 51002, Iraq
[1] Email: raadmod88@gmail.com
[2] Email: ahmed_saleem@yahoo.com

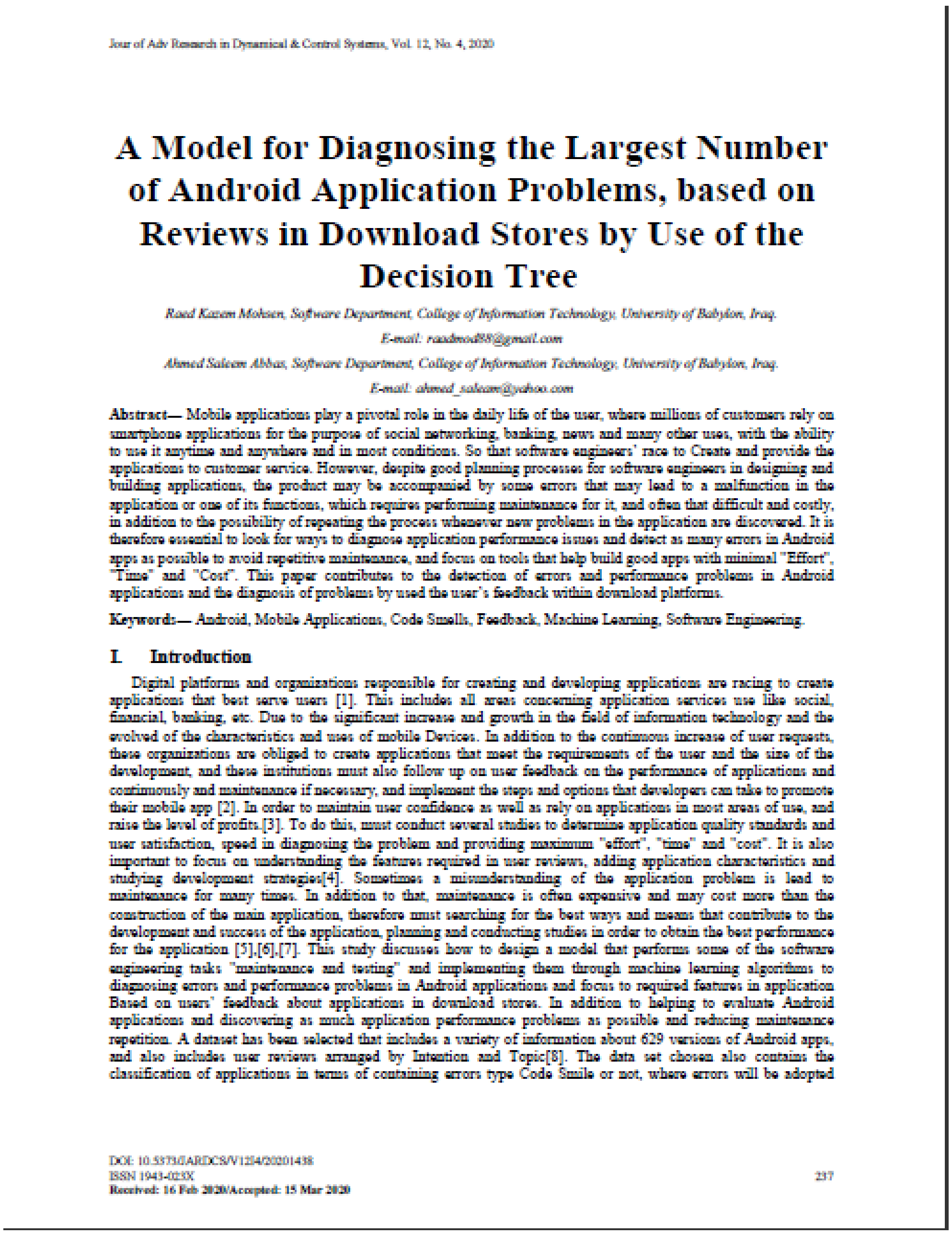
Abstract. Millions of customers rely on smart phone applications for social networking, banking, health, news and many other uses and is also usable anytime, anywhere and in most environmental conditions. However, despite good planning by software engineers and organizations responsible for designing and building applications, the process of building or maintaining the application may be marred by some errors that lead to malfunction of the application or one of its functions, and this is often discovered after a long period, and this affects Opportunities of application success. Therefore, it is necessary to follow up on user feedback on the performance of applications, and to search for tools and methods that contribute to know what users want quickly to save "effort", "time" and "cost". This paper discusses some of the main functions of software engineering and the possibility of implementing them in machine learning to detect the presence of design errors "Code Smell "in Android applications with the diagnosis of error type.

1. Introduction

Software engineers seek to build and development the applications that help provide maximum service to users. This includes all categories of different applications (health, news, games, movies, etc.). Due to the rapid development of technology and mobile phones and increased user requirements It has become necessary to create applications that meet the increasing demands of users, maintain and develop applications if necessary, provide services and keep pace with rapid technology developments. Therefore, the construction and development of mobile applications is of great interest due to its increasing number and diversity of uses, and profits resulting from it.(1). The increasing and disparate users requirements have made software engineers They are quick to build and develop the apps quickly and under pressure , to get a Complicated applications that can perform many tasks and functions as well as include applications to work in systems of all kinds to cover their requirements and keep abreast of developments (2). This may be accompanied by problems, unintended errors, and sometimes difficult to detect. This may result in a malfunction in the quality of the application and affect the chances of success, not to mention the possibility of errors after maintenance or development (3),(4) . Therefore, most applications need improvement and maintenance periodically (5) . It is also important for software engineers to understand the features required by users, to add application characteristics and to study development strategies (6) . It is thus necessary to think about ways that to hurry up software engineering tasks and scale back the efforts of code engineers to grasp what users wish and cash in of those strategies

1

Appendix B



Jour of Adv Research in Dynamical & Control Systems, Vol. 12, No. 4, 2020

A Model for Diagnosing the Largest Number of Android Application Problems, based on Reviews in Download Stores by Use of the Decision Tree

Raed Kazem Mohsen, Software Department, College of Information Technology, University of Babylon, Iraq.

E-mail: raadmod88@gmail.com

Ahmed Saleem Abbas, Software Department, College of Information Technology, University of Babylon, Iraq.

E-mail: ahmed_saleam@yahoo.com

Abstract— Mobile applications play a pivotal role in the daily life of the user, where millions of customers rely on smartphone applications for the purpose of social networking, banking, news and many other uses, with the ability to use it anytime and anywhere and in most conditions. So that software engineers' race to Create and provide the applications to customer service. However, despite good planning processes for software engineers in designing and building applications, the product may be accompanied by some errors that may lead to a malfunction in the application or one of its functions, which requires performing maintenance for it, and often that difficult and costly, in addition to the possibility of repeating the process whenever new problems in the application are discovered. It is therefore essential to look for ways to diagnose application performance issues and detect as many errors in Android apps as possible to avoid repetitive maintenance, and focus on tools that help build good apps with minimal "Effort", "Time" and "Cost". This paper contributes to the detection of errors and performance problems in Android applications and the diagnosis of problems by used the user's feedback within download platforms.

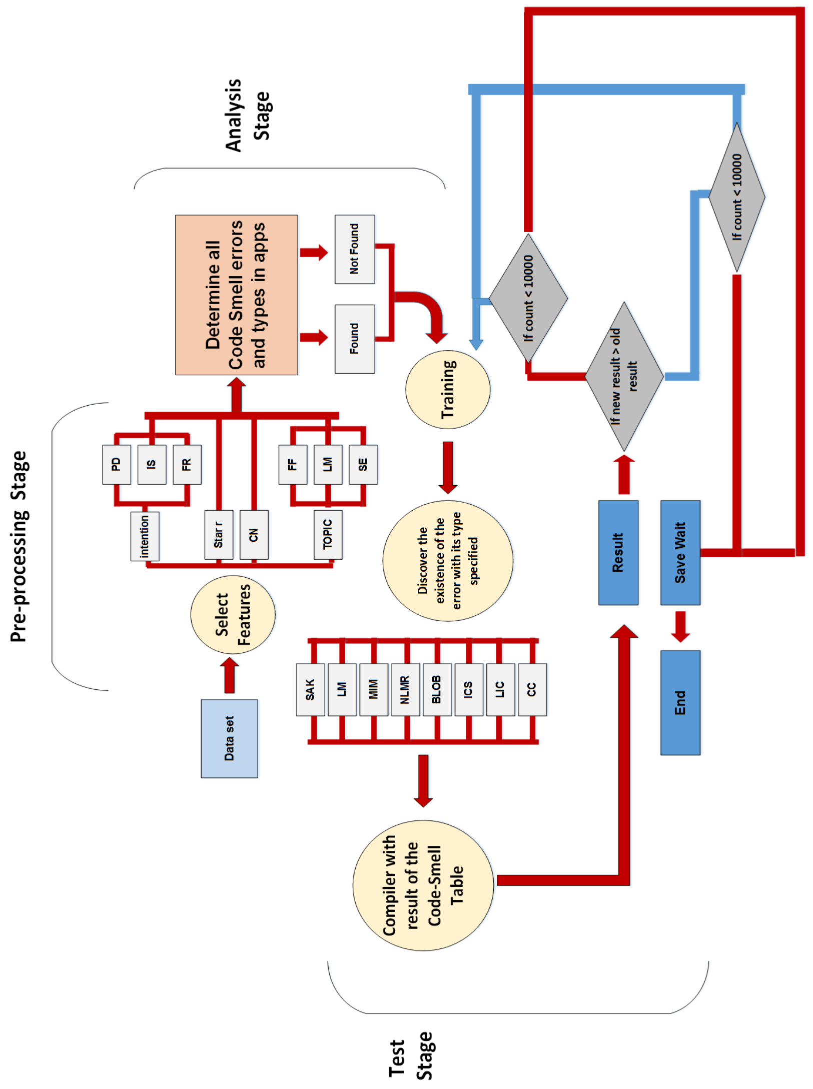
Keywords— Android, Mobile Applications, Code Smells, Feedback, Machine Learning, Software Engineering.

I. Introduction

Digital platforms and organizations responsible for creating and developing applications are racing to create applications that best serve users [1]. This includes all areas concerning application services use like social, financial, banking, etc. Due to the significant increase and growth in the field of information technology and the evolved of the characteristics and uses of mobile Devices. In addition to the continuous increase of user requests, these organizations are obliged to create applications that meet the requirements of the user and the size of the development, and these institutions must also follow up on user feedback on the performance of applications and continuously and maintenance if necessary, and implement the steps and options that developers can take to promote their mobile app [2]. In order to maintain user confidence as well as rely on applications in most areas of use, and raise the level of profits.[3]. To do this, must conduct several studies to determine application quality standards and user satisfaction, speed in diagnosing the problem and providing maximum "effort", "time" and "cost". It is also important to focus on understanding the features required in user reviews, adding application characteristics and studying development strategies[4]. Sometimes a misunderstanding of the application problem is lead to maintenance for many times. In addition to that, maintenance is often expensive and may cost more than the construction of the main application, therefore must searching for the best ways and means that contribute to the development and success of the application, planning and conducting studies in order to obtain the best performance for the application [5],[6],[7]. This study discusses how to design a model that performs some of the software engineering tasks "maintenance and testing" and implementing them through machine learning algorithms to diagnosing errors and performance problems in Android applications and focus to required features in application Based on users' feedback about applications in download stores. In addition to helping to evaluate Android applications and discovering as much application performance problems as possible and reducing maintenance repetition. A dataset has been selected that includes a variety of information about 629 versions of Android apps, and also includes user reviews arranged by Intention and Topic[8]. The data set chosen also contains the classification of applications in terms of containing errors type Code Smile or not, where errors will be adopted

DOI: 10.5373/JARDCS/V12I4/20201438
ISSN 1943-023X
Received: 16 Feb 2020/Accepted: 15 Mar 2020

Appendix C


Pre-processing Stage
Analysis Stage
Test Stage
Data set
Select Features
intention
Star r
CN
TOPIC
PD
IS
FR
FF
LM
SE
Determine all Code Smell errors and types in apps
Not Found
Found
Training
Discover the existence of the error with its type specified
SAK
LM
MIM
NLMR
BLOB
ICS
LIC
CC
Compiler with result of the Code-Smell Table
Result
If new result > old result
If count < 10000
If count < 10000
Save Wait
End

الخلاصة

نحاول في هذه الدراسة مساعدة مهندسي البرمجيات على بناء التطبيقات وتطويرها وصيانتها. حيث توجد العديد من التحديات والعوامل التي تؤثر على نجاح التطبيق، خاصة في بداية طرح التطبيق للاستخدام، أو بعد إجراء عمليات الصيانة والتطوير، والمحاولة في زيادة سرعة تقديم الميزات المطلوبة، ان التطور في مجال التطبيقات و الهواتف المحمولة واستخدامه في معظم مجالات الحياة أدى إلى الحاجة إلى إنشاء تطبيقات متوافقة مع المتطلبات الجديدة، في الوقت الحالي، معظم الوزارات والخدمات المؤسسات لديها تطبيقات في المتاجر، وتقدم العديد من الخدمات من خلالها. مثل التقدم للوظائف والقبول في الجامعات والحصول على تأشيرة أو جواز سفر. هذا جعل "مهندسي البرمجيات" والمطورين يقومون بإنشاء وتطوير التطبيقات بسرعة لإرضاء المستخدمين وتجنب فقدانهم، وقد تتضمن عملية التطوير والبناء العديد من الأخطاء غير المقصودة، مما يؤدي إلى خلل في التصميم أو مشكلة في عمل التطبيق وهو أمر صعب الكشف عنه خاصة إذا كان التطبيق معقدًا، مما يجعل من الصعب إجراء الصيانة. لذلك، من الضروري البحث عن طرق سريعة تساعد في الصيانة واستكشاف الأخطاء وإصلاحها وتطوير التطبيقات وتجنب عمليات الصيانة المتكررة. سنستخدم مجموعة متنوعة من الأساليب لمساعدة مهندسي البرمجيات على فهم مراجعات المستخدمين بسرعة. حيث تم عمل نموذج باستخدام تصنيف شجرة القرار، لغرض تحديد تطبيقات **Android** التي تحتوي على أخطاء (مشكلة، خطأ في التطبيق، أو إحدى الوظائف) وتشخيص الميزات التي يريدها المستخدمون في التطبيقات، لإظهارها لمسؤولي التطبيق لدراستها وإضافة الميزات المطلوبة في التطبيقات المستقبلية. بالإضافة إلى المساعدة في العثور على أكبر عدد ممكن من الأخطاء من أجل تقليل وتيرة الصيانة. والمساهمة في تقليل جهود مهندسي البرمجيات وتسريع بناء تطبيقات جيدة وموثوقة وفي أقل وقت. تم التحقق من دقة النموذج ووجد أنه يعمل بدقة من ٩٢٪ إلى ٩٣

جمهورية العراق

وزارة التعليم العالي والبحث العلمي

جامعة بابل

كلية تكنلوجيا المعلومات

قسم البرمجيات

كشف الخطأ في هندسة البرمجيات لتحسين مهمة الاختبار بناءً على خوارزمية شجرة القرار: دراسة حالة لتطبيقات أندرويد

رسالة

مقدمة إلى مجلس كلية تكنولوجيا المعلومات في جامعة بابل والتي هي جزء من متطلبات الحصول على درجة الماجستير في تكنولوجيا المعلومات – البرمجيات

الطالب

رائد كاظم محسن مزعل

بأشراف

الأستاذ المساعد الدكتور احمد سليم عباس

1439 هـ 2020 م