# **CHAPTER 1**

## **INTRODUCTION**

In the dynamic field of software development, the pursuit of high-quality software products stands as a crucial and ever-evolving objective. The presence of defects and errors within software applications can have far-reaching consequences, including system failures, security vulnerabilities, and customer dissatisfaction. Consequently, software assurance teams continuously strive to employ effective quality assurance practices throughout the software development life cycle (SDLC) (Chen and Hossain, 2022) to mitigate these risks. While traditional manual testing methods have long served as the foundation of quality assurance efforts, they are not without limitations, such as time-consuming processes, human error, and incomplete test coverage.

In recent years, the rapid advancements in machine learning and artificial intelligence have sparked a growing interest in harnessing these transformative technologies to develop automated quality assurance systems for software development (Omri and Sinz, 2021). These systems offer the promise of enhancing the efficiency and effectiveness of the quality assurance process by automating various tasks, including test case generation, defect detection, and performance optimization. By incorporating machine learning algorithms, these systems have the potential to learn from historical data, adapt to changing software landscapes, and continually improve their decision-making capabilities.

The objective of this research is to significantly contribute to the field of software engineering by creating an automated quality assurance system for software development, leveraging machine learning techniques. By leveraging a diverse array of machine learning techniques, we seek to automate and optimize different facets of the quality assurance process, thereby overcoming the limitations inherent in traditional manual testing methods and elevating the overall quality of software applications.

The development of the proposed automated quality assurance system encompasses several pivotal components that align with the research interests and expertise of researchers in this field. Firstly, a meticulous and comprehensive dataset will be gotten, this meticulously collected dataset will serve as the bedrock for training and evaluating the machine learning models embedded within the system.

Building upon current advancements in machine learning, this research aims to carefully choose and customize a set of advanced machine learning algorithms for the development of a sophisticated automated quality assurance system in software development. These algorithms will be trained on the dataset to identify intricate patterns and relationships between various software artifacts and their corresponding quality attributes. By capturing the essence of these relationships, the trained models will be poised to automate diverse quality assurance tasks, such as generating intelligent test cases, identifying potential defects with heightened accuracy, and predicting software performance with enhanced precision.

To ensure the validity and reliability of the developed automated quality assurance system models, evaluation procedures will be conducted to see its capabilities. These procedures will involve benchmarking the system against established manual testing methods to gauge its performance across key metrics, including accuracy, efficiency, and coverage (Prof. Rumana Anjum and Dr Madhub B, 2021).

Ethical considerations related to the developed automated quality assurance system will be accorded the utmost attention and scrutiny. Potential biases like imbalanced data classification inherent in the machine learning models and their impact on the quality assessment process will be carefully examined, and appropriate measures will be implemented to ensure fairness, transparency, and accountability in the system's decision-making process.

The anticipated outcomes of this research project bear immense significance for the software development industry. The developed machine learning-based automated quality assurance system holds immense promise in transforming the assessment and maintenance of software quality. By automating laborious and time-consuming tasks, software development teams can optimize resource allocation, significantly accelerate the software development life cycle, and effectively reduce time-to-market. Moreover, the system's capability to identify defects with heightened accuracy and optimize software performance can have a transformative impact on the overall quality and reliability of software applications, leading to enhanced customer satisfaction and improved market competitiveness.

In conclusion, this research project endeavours to develop a tailored and sophisticated machine learning-based automated quality assurance system for software development. By harnessing the potential of cutting-edge machine learning algorithms, this system aims to automate and optimize various quality assurance tasks, ultimately elevating the quality and reliability of software applications. This research aims to address the limitations inherent in traditional manual testing methods, presenting a unique opportunity to revolutionize software quality assurance practices.

## **STATEMENT OF PROBLEM**

The quality of software applications remains a significant concern in the field of software development. Despite the rigorous processes involved in requirements analysis, specification, and software development, defects persist in software applications, impacting their performance, functionality, and overall reliability. These defects can arise due to various factors, such as incomplete or ambiguous requirements, errors in design or implementation, and inadequate testing procedures (Trudova and Dolezel, no date).

The lack of effective software quality estimation techniques poses a challenge for software development teams. Without accurate and reliable methods to assess software quality, it becomes difficult to plan and implement appropriate quality assurance practices throughout the development life cycle. Additionally, the absence of a benchmarking mechanism hinders the ability to compare the quality of software products against industry standards, leading to potential inefficiencies and suboptimal software performance.

Furthermore, the relationship between internal and external software quality attributes remains a complex and understudied area. While internal quality attributes, such as code maintainability, modularity, and design coherence, are crucial indicators of software robustness, their direct impact on external quality attributes, including system performance, execution time, and user satisfaction, requires further investigation. Understanding this relationship is essential for developing effective quality models that can accurately predict software quality based on internal attributes.

Moreover, although various software quality prediction models have been proposed in the literature, there is a growing interest in exploring machine learning approaches for improved accuracy and effectiveness. However, the current state of research lacks a comprehensive review and synthesis of the existing machine-learning techniques specifically tailored to software quality prediction. A thorough examination of these approaches is necessary to identify their strengths, limitations, and potential areas for improvement.

Therefore, the central problem addressed in this research is the need for a comprehensive understanding of software quality estimation techniques, with a specific focus on exploring machine learning approaches for predicting software quality based on internal attributes. By addressing this problem, we aim to bridge the knowledge gap and provide valuable insights for software developers, quality assurance practitioners, and researchers seeking to enhance the quality assessment processes in software development projects.

## **AIMS AND OBJECTIVES**

### **1.3.1 Aims**

1. Develop an advanced and sophisticated machine learning-based automated quality assurance system tailored specifically for software development.
2. Employ state-of-the-art machine learning techniques to automate and optimize diverse facets of the quality assurance process in software development.
3. Enhance the overall quality and dependability of software applications by implementing the developed automated quality assurance system effectively.

### **1.3.2 Objectives**

1. Research and find a suitable standard dataset for preparing, evaluating and implementing the machine learning models.
2. Select and customize machine learning algorithms that align with the distinctive requirements of quality assurance in software development, leveraging their potential to maximize system performance.
3. Automate the generation of intelligent and comprehensive test cases through the development of robust algorithms and methodologies, resulting in improved test coverage and heightened efficiency.
4. Enhance defect detection capabilities by leveraging advanced machine learning techniques to identify intricate patterns and indicators of potential defects, enabling timely detection and resolution.
5. Optimize software performance by utilizing sophisticated machine learning approaches that analyse historical data to identify key factors contributing to optimal performance, facilitating the identification and mitigation of bottlenecks.
6. Validate and evaluate the effectiveness of the developed automated quality assurance system through rigorous testing and evaluation procedures utilizing real-world software development projects as test cases.
7. Address ethical considerations by conducting an in-depth examination of potential biases within the machine learning models, and implementing appropriate mitigation strategies to ensure fairness, transparency, and accountability in the decision-making process.

## **SCOPE OF STUDY**

This study aims to explore the multifaceted domain of software quality assurance, focusing on the identification and mitigation of defects originating from activities such as requirements analysis, specification, and other key stages of software development. The significance of software quality estimation is widely recognized, as it serves crucial purposes at various stages of the development process. Not only does it enable the planning of project-based quality assurance practices, but it also facilitates benchmarking against industry standards. A key indicator of software quality is the number of defects per unit, which plays a pivotal role in assessing and evaluating the overall quality of software applications.

Within the context of this study, software quality is defined as a comprehensive measure of the performance of the system on which the software is implemented, encompassing factors such as execution time, memory capacity utilization, and the probability of errors. Additionally, the effort invested by software developers represents a significant factor in assessing the quality of the end product. The concept of software quality extends to both internal and external dimensions. Internal quality pertains to the evaluation of software during the development life cycle (SDLC), focusing on attributes such as code maintainability, modularity, and design coherence. Conversely, external quality is measured during the implementation phase and encompasses the level of functionality and compliance with desired specifications.

Of particular importance is the interdependence between external and internal quality attributes. To effectively assess the external quality of a software product, it becomes essential to develop quality models that establish the relationship between internal and external quality attributes. This necessitates the identification of internal attributes and a thorough examination of their correlation with external quality attributes. Numerous software quality prediction models have been proposed by various authors, among which the machine learning approach has gained popularity and recognition for its claimed effectiveness. Motivated by the potential of machine learning, this study aims to conduct a comprehensive review of machine learning approaches in the context of software quality prediction models.

The primary focus of this study is to undertake an extensive and critical analysis of machine learning techniques employed in software quality prediction models. By synthesizing existing literature and research findings, the study aims to elucidate the theoretical underpinnings, methodological advancements, and practical implications associated with the utilization of machine learning in the development of software quality prediction models. A rigorous evaluation of the strengths, limitations, and efficacy of diverse machine learning approaches will be conducted, focusing on their ability to predict software quality based on internal attributes.

Within the scope of analysis, this study encompasses a wide range of machine learning approaches employed in software quality prediction. This includes but is not limited to supervised learning, unsupervised learning, and ensemble methods. Various software quality attributes, both internal and external, will be considered, such as code complexity, fault density, performance metrics, and user satisfaction. The study will examine how machine learning models capture and utilize the relationship between internal and external quality attributes to devise accurate predictions.

The anticipated outcome of this study is to provide a comprehensive and meticulously curated overview of machine learning approaches in software quality prediction models. By critically evaluating the existing body of research and identifying gaps in knowledge, this study intends to contribute to the understanding of how machine learning can be effectively harnessed to enhance software quality estimation. The insights gained from this study will have far-reaching implications for software developers, quality assurance practitioners, and researchers seeking to augment the accuracy and efficiency of software quality assessment practices.

### **1.4.1 Limitations of the System Scope:**

1. Limited to Software Development: The scope of this study is specifically focused on software development and does not encompass other domains or industries. The findings and recommendations may not be directly applicable to non-software development contexts.
2. Exclusion of Non-Machine Learning Approaches: The study specifically focuses on machine learning approaches for software quality prediction models, thereby excluding other non-machine learning techniques that may also be relevant in the field.
3. Scope of Ethical Considerations: While the study acknowledges the importance of ethical considerations in software quality assessment, the scope is limited to the examination of biases in machine learning models. Other ethical aspects, such as privacy, security, and fairness, are not extensively explored within the scope of this study (Prof. Rumana Anjum and Dr Madhub B, 2021).

### **1.4.2 Functionalities of the System Scope:**

1. Dataset Collection: The study emphasizes the collection of a comprehensive dataset encompassing various software development artefacts, including requirements, design specifications, code repositories, and test cases. This dataset forms the foundation for training and evaluating machine learning models.
2. Selection and Tailoring of Algorithms: The study aims to select and customize machine learning algorithms that are suitable for software quality prediction models. This includes techniques such as supervised learning, unsupervised learning, and ensemble methods, tailored to the unique requirements of quality assurance in software development.
3. Automation of Test Case Generation: The system scope involves the development of algorithms and methodologies to automate the generation of intelligent and comprehensive test cases. This functionality aims to improve test coverage and efficiency in software quality assurance practices.
4. Defect Detection and Performance Optimization: The study explores the application of machine learning techniques for improved defect detection and performance optimization in software applications. By analysing historical data and identifying patterns, the system aims to enhance the overall quality and reliability of software products.
5. Validation and Evaluation Procedures: Rigorous testing and evaluation procedures are outlined within the system scope.
6. Ethical Considerations: The study acknowledges the ethical implications of software quality assessment, particularly biases in machine learning models. Measures will be taken to address and mitigate these biases, ensuring fairness, transparency, and accountability in the decision-making process of the automated quality assurance system.

## **METHODOLOGY**

The methodology employed in this study focuses on the development of an automated software quality assurance system using machine learning algorithms. The methodology involves several interconnected steps that collectively contribute to the understanding and implementation of the system.

The study begins with a thorough literature review to explore the existing research and practices in automated software quality assurance and machine learning algorithms. This review provides valuable insights into the current state-of-the-art, highlighting the strengths and limitations of different approaches.

Data collection is a critical component of the methodology. A diverse and comprehensive dataset is curated, consisting of historical software development artefacts such as requirements, design specifications, code repositories, and test cases. The dataset represents a representative sample of both open-source and proprietary software projects to ensure robustness and applicability.

Pre-processing and feature engineering techniques are then applied to the collected dataset. Data pre-processing involves cleaning the data, handling missing values, and normalizing the features. Feature engineering techniques are employed to extract relevant and informative features from the software artefacts, including code complexity, maintainability, and performance metrics.

The next step involves algorithm selection. Through a comprehensive evaluation process, machine learning algorithms are carefully chosen based on their performance, interpretability, and scalability. Algorithms such as decision trees, random forests, support vector machines, and neural networks are considered for their suitability in automating various quality assurance tasks.

With the selected algorithms, machine learning models are developed to automate quality assurance tasks, including defect detection and performance optimization. Supervised learning techniques are employed to train the models using the curated dataset. Labelled instances are used for training and testing, ensuring the models learn and generalize effectively.

The developed machine learning models are integrated into a software application, forming the automated quality assurance system. The system is implemented using appropriate programming languages and frameworks, ensuring compatibility with different software development environments.

The performance of the automated quality assurance system is then evaluated and validated. Real-world software projects are used as test cases, and the system is benchmarked against existing manual testing methods and state-of-the-art quality assurance approaches. Performance metrics such as accuracy, precision, recall, and F1-score are used to assess the effectiveness of the system in defect detection and performance optimization.

Ethical considerations are an integral part of the methodology. Measures are taken to address potential biases in the machine learning models, ensure fairness, transparency, and privacy in the decision-making process, and adhere to legal requirements regarding data usage and protection.

Limitations and delimitations of the study are acknowledged, including the availability and representativeness of the dataset, potential biases in the machine learning models, and the generalizability of the findings to different software development contexts.

The collected data is analysed using appropriate statistical and machine-learning techniques to extract insights and evaluate the performance of the developed models. Descriptive statistics, data visualization, and model evaluation techniques are applied to interpret and present the findings.

The results and findings of the study are presented and discussed comprehensively, highlighting the effectiveness, limitations, and implications of the automated quality assurance system. Recommendations for future research and practical applications are provided based on the research outcomes.

The highlight of this includes;

1. Literature Review
2. Data Collection
3. Pre-processing and Feature Engineering
4. Algorithm Selection
5. Model Development
6. Model Evaluation
7. System Implementation
8. System Evaluation
9. Results and Findings

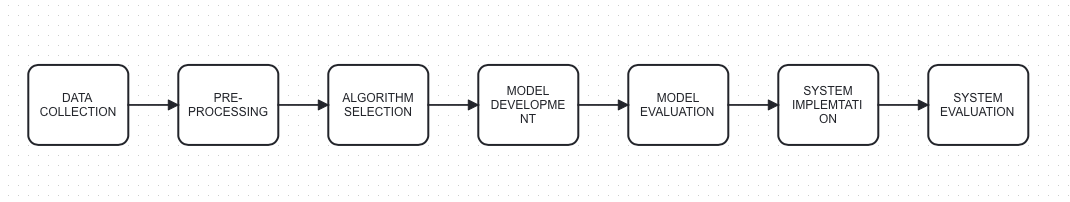


Figure 1 A block diagram showing the methodology of the system

By following this methodology, the study aims to contribute to the field of automated software quality assurance by leveraging machine learning algorithms to enhance defect detection and performance optimization. The systematic approach ensures a rigorous evaluation of the developed system and provides valuable insights for practitioners and researchers in the software engineering domain.

## **SIGNIFICANCE OF STUDY**

The significance of the machine learning-based automated software assurance includes;

1. Enhancing Software Quality: The project aims to significantly enhance software quality through the application of machine learning algorithms, ensuring reliable and robust software applications.
2. Time and Cost Efficiency: Automation of quality assurance tasks leads to improved efficiency, reduced manual effort, and cost savings, enabling faster time-to-market for software products.
3. Accuracy and Consistency: The automated system ensures accurate and consistent quality assessment, minimizing human errors and inconsistencies in defect identification.
4. Scalability and Adaptability: The project's focus on machine learning algorithms enables scalability and adaptability to diverse software projects and environments, accommodating varying requirements.
5. Insights and Recommendations: The project provides valuable insights and recommendations for stakeholders in software development, aiding informed decision-making and optimizing quality assurance practices.
6. Advancing the Field: By exploring the intersection of machine learning and software quality, the project contributes to advancements in automated testing and defect detection, inspiring further research and innovation in the field of software engineering.

The project's significance lies in enhancing software quality, improving time and cost efficiency, ensuring accuracy and consistency, enabling scalability and adaptability, providing insights and recommendations, and advancing the field of software engineering.

## **PROJECT OUTLINE**

The project report follows a structured sequence that outlines the subsequent chapters in a logical and organized manner:

* Chapter 2: Historical Overview and Critical Evaluation of Previous Research.

In this chapter, a comprehensive historical overview of the development of the Automated Quality Assurance system using Machine Learning is presented. The chapter also includes a critical evaluation of relevant previous research, highlighting key advancements and identifying gaps that the current project aims to address.

* Chapter 3: Methodology and Approach.

Chapter 3 discusses the methodology and approach employed to achieve the general aims and objectives of the project. It provides a detailed explanation of the research design, data collection, pre-processing techniques, algorithm selection, model development, and performance evaluation. This chapter serves as a blueprint for the implementation of the project.

* Chapter 4: Results and Implementation Discussion.

Chapter 4 delves into the discussion of the project's results and implementation. It presents and analyses the findings obtained from the application of the developed automated quality assurance system. The chapter highlights the performance, accuracy, and effectiveness of the system in defect detection and performance optimization. Additionally, it explores the practical aspects of implementing the system within a software development environment.

* Chapter 5: Conclusion and Summary.

The concluding chapter, Chapter 5, provides a summary of the project's key findings, accomplishments, and contributions. It reflects upon the achieved objectives, discusses the implications of the research outcomes, and provides recommendations for future research and practical applications. This chapter serves as a culmination of the project, summarizing the key insights and outlining the project's overall impact.

# **CHAPTER 2: LITERATURE REVIEW**

This literature review aims to offer a comprehensive overview of the present state of automated software quality assurance using machine learning algorithms. By consolidating existing knowledge and identifying areas for further research, it will serve as a valuable resource for researchers, practitioners, and organizations keen on exploring the transformative potential of machine learning in the field of software quality assurance.

## **2.1 OVERVIEW OF SOFTWARE QUALITY ASSURANCE (SQA)**

As software systems become increasingly complex, there is a heightened demand for advanced testing techniques. Manual software testing is often found to be ineffective due to its resource-intensive nature, slow execution speed, and limited test coverage. These are precisely the challenges that test automation aims to tackle, providing a solution in many cases (Trudova and Dolezel, no date).

Software testing serves as an investigative process aimed at providing stakeholders with valuable insights regarding the quality of the software product or system under test (SUT) (Felix and Lee, 2017). Typically, testing constitutes a significant portion, approximately 30% to 40%, of the total project effort, and it incurs more than 50% of the project's overall cost. The ultimate objective is to achieve higher-quality software by ensuring the absence of failures in the SUT. A failure manifests when the observed behaviour of the SUT deviates from the expected behaviour outlined in its requirements or other defined descriptions of anticipated behaviour (Khaliq, Farooq and Khan, 2022a). Dynamic analysis, often referred to as such because it involves the execution of the system under test (SUT), is a key aspect of this activity. On the other hand, there are quality assurance activities that can be conducted without the need for executing the SUT (Prof. Rumana Anjum and Dr Madhub B, 2021).

The field of software engineering encompasses a significant and continuously evolving research domain known as fault prediction. Researchers have devoted considerable resources to improving fault prediction performance by employing various techniques and metrics. Numerous studies have been conducted to explore software fault prediction, shaping the ongoing advancement in this area. To provide a concise overview of the research progression over the past two decades, Figure 2 illustrates the historical trajectory of software fault prediction studies, summarizing the key developments in this field (Omri and Sinz, 2021).

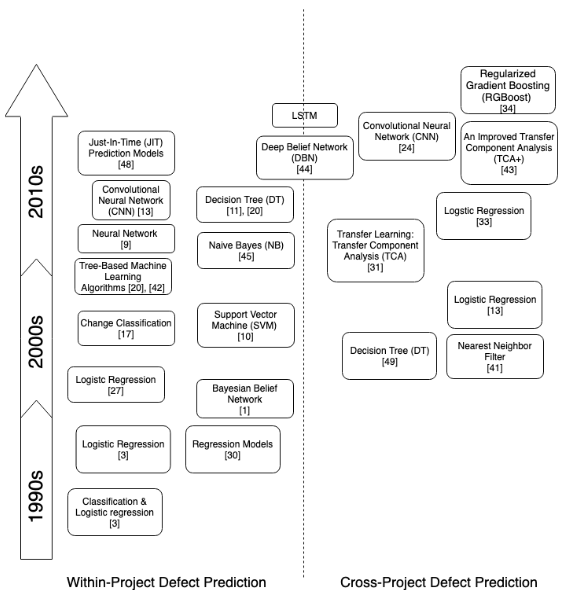


Figure 2 History of Software Defect Prediction (Omri and Sinz, 2021)

The incorporation of AI into software development is still in its nascent phase, and its degree of autonomy is not as advanced as in more mature fields like self-driving systems or voice-assisted control. However, there is ongoing progress towards achieving autonomous testing capabilities. The focus of applying AI in software testing tools is to streamline the software development lifecycle. By employing reasoning, problem-solving, and sometimes machine learning, AI has the potential to automate and reduce the burden of mundane and tedious tasks involved in development and testing (Prof. Rumana Anjum and Dr Madhub B, 2021).

A crucial component of the testing process is the test case, which outlines the specific conditions under which the system under test (SUT) should be executed to identify potential failures. When a test case successfully uncovers a failure, it is deemed effective. Test cases are typically derived from functional specifications, design specifications, or requirements specifications (Wable *et al.*, 2021). A test case specification encompasses the following elements:

1. The preconditions outline the initial environment and state of the system under test (SUT) before the test case.
2. The test steps entail detailing the specific actions that need to be executed into an array. The expected results of the executing test case (Lima, Rosado Da Cruz and Ribeiro, no date).
3. The actual results, which describe the results of the executing test case.

## **2.2 CURRENT SOFTWARE TESTING METHODS**

Software testing holds significant importance within the software development life cycle. During the software development process, organizations allocate approximately half of their budget to testing-related tasks (Kaur and Chopra, 2014).

Software testing can be performed either manually or automatically;

1. Manual Testing: During manual testing, testers perform test cases without relying on tools or scripts. In this approach, the tester assumes the role of an end-user and manually tests the software to uncover any unexpected behaviours (Jayanthi and Florence, 2019).
2. Automated Testing: Automated testing is a type of software testing that employs specialized software tools or frameworks to execute pre-defined tests. These tools, commonly known as test automation tools or test automation frameworks, alleviate testers from the manual execution of test cases. However, it is important to note that the planning and creation of test cases in the form of test scripts still require manual effort (Vanmali, Last and Kandel, no date).

In manual testing, humans carry out the testing process by manually examining the code, comparing the results with behaviours, and documenting observations (Chen and Hossain, 2022). However, manual testing often lacks effectiveness and efficiency. An optimal approach is to employ automated testing, which enhances effectiveness, efficiency, and test coverage (Al-Masri and Al-Sorori, 2022). Automated testing utilizes specialized tools that do not require human intervention, requiring only financial and resource investments. Compared to manual testing, automated testing delivers more accurate and prompt results. A common approach involves generating test cases and utilizing automated tools to conduct the tests. The results are then compared against expected outcomes to determine optimality.

### **2.2.1 Black-Box Testing**

Black-box testing also referred to as functional testing, focuses on examining the external behaviour of software without delving into its internal structure.

This approach involves testing the application primarily based on its outcomes, across various levels, without requiring an understanding of its internal workings. While this method may offer limited coverage, it is the least time-consuming and offers a straightforward test perception. Commonly employed techniques in this approach include fuzzing, which involves injecting malformed data to identify implementation bugs, and state transitioning testing, which focuses on testing state machines and navigation within graphical user interfaces (Lima, Rosado Da Cruz and Ribeiro, no date).

The primary focus of black-box testing is to analyze the inputs provided to the software and the corresponding outputs it produces (Khaliq, Farooq and Khan, 2022b).

Black box testing is conducted by testers during the later stages of software development, where they evaluate the software through user interfaces, data structures, databases, or application programming interfaces. Unlike other testing approaches, black box testing does not grant access to the underlying code. The success of the software is determined by its ability to execute test cases and produce the expected output, which is defined based on user requirements. The testers determine the appropriate inputs to provide and the expected outputs to be generated, while focusing on analyzing the external behaviour of the software (Tripathi, Bhadra and Singh, 2016).

In the software development life cycle, during the analysis phase, software architects gather requirements and translate them into program features that align with customer needs. In the subsequent design phase, developers receive these requirements and create design documents for programmers to implement. Various techniques employed in black box testing include boundary value analysis, equivalence classes, decision table testing, state transition testing, and use case testing. The advantages of black box testing lie in its simplicity and ease of use.

The primary challenge with the black-box approach lies in generating test cases that have a higher probability of uncovering faults (Vanmali, Last and Kandel, no date).

Existing techniques under black-box testing include;

1. Random Testing
2. Equivalence Partitioning
3. Boundary Value Analysis
4. State Transition-Based Testing (Ratna, Cheekaty and Kalidasu, no date)

### **2.2.2 White-Box Testing**

White box testing is a type of testing which uses source code to create test cases, it’s also known as structural testing (Khaliq, Farooq and Khan, 2022a).

White-box testing involves an in-depth examination of a program's code, including its internal logic and structure (Wable *et al.*, 2021). Testers conducting white-box testing require full access to the source code to identify any sections that exhibit undesirable behavior. This offers a high level of granularity, making it potentially time-consuming and necessitating skilled testers. However, it also enables maximum coverage when creating test scenarios (Lima, Rosado Da Cruz and Ribeiro, no date).

White box testing examines the logical flows, crucial control flow paths, and program logic of the software being tested (Panda *et al.*, 2020).

White box testing encompasses two types of testing: integration testing and unit testing. Integration testing involves using input and output file pairs to evaluate the overall functionality of the software (Deepak Nandal and Prakash Sangwan, no date). Each integration test is defined in a configuration file, with each test specified on a separate line. On the other hand, unit testing focuses on testing the smallest independently testable component of the software. The responsibility for conducting unit tests lies with the development team (Al-Masri and Al-Sorori, 2022).

The implementation code of the system is utilized to design the tests and functional testing, showing that the governing code is not visible to the programmer (Singh *et al.*, 2018).

### **2.2.3 Grey-Box Testing**

Introduced in the late 1980s, the grey box testing paradigm, which revolves around object-oriented testing, addresses the challenges and complexities associated with testing the unique features introduced by object-oriented programming concepts (Panda and Dash, 2020).

Grey box testing is a testing technique utilized to evaluate a software product or application with limited knowledge of its internal structure. The primary goal of grey box testing is to identify and detect potential defects arising from issues such as improper code structure or inadequate application usage (Khaliq, Farooq and Khan, 2022a).

Grey-box testing is employed to assess an application with restricted insight into its internal workings while maintaining a comprehensive understanding of the fundamental aspects of the system.

In grey-box testing, the modules are examined to formulate test cases, employing a white-box approach. However, the actual tests are conducted using the exposed interfaces, following a black-box approach. Due to the unavailability of source code, the test coverage may be limited. Common forms of grey-box testing include regression testing, which ensures that recent modifications to the software are functioning correctly, and pattern testing, which verifies the architecture and design of the application (Lima, Rosado Da Cruz and Ribeiro, no date).

Grey-box testing can be considered a fusion of white-box and black-box testing methodologies, encompassing aspects of both approaches. It is distinct from being purely categorized as either white-box or black-box testing. Software testing, on the other hand, is a verification process that ensures the software aligns with the expectations and requirements of the clients. The purpose of verification in software testing is to ensure that the developed software meets the specified requirements of the clients (Tripathi, Bhadra and Singh, 2016).

## **2.3ARTIFICIALL INTELLIGENCE AND APPLICATIONS TO SOFTWARE QUALITY ASSURANCE**

According to John McCarthy's definition in 1959, AI algorithms can be understood as a field within computer science that focuses on simulating intelligent behaviour in computers. Tom Mitchell's definition in 1997 defines Machine Learning as the study of algorithms that enable computer programs to enhance their performance through experience. Drawing inspiration from the learning capabilities of rational beings, Machine Learning is structured into different types of learning, such as supervised, unsupervised, and reinforcement learning. On the other hand, AI is organized into paradigms and approaches that encompass rational characteristics, including connectionist, genetic, statistical, and probabilistic methods. These paradigms are realized through techniques like Artificial Neural Networks, Genetic Algorithms, naive Bayes, Case-Based Reasoning, inductive rules, and decision trees. By combining AI algorithms with a Machine Learning approach, it becomes possible to extract and explore information for tasks such as classification, association, optimization, clustering, forecasting, pattern identification, and more (Lima, Rosado Da Cruz and Ribeiro, no date).

Researchers have indicated that artificial intelligence (AI) exhibits a high level of accuracy in predicting software defects. Furthermore, the performance of AI-based defect prediction can be enhanced through the implementation of feature reduction techniques, leading to reduced complexity and improved overall performance (Jayanthi and Florence, 2019).

### **2.3.1 Artificial Neural Network (ANN)**

An artificial neural network (ANN) is a mathematical model that draws inspiration from the structure and functional characteristics of biological neural networks. It exhibits distributed and parallel data processing capabilities. During the learning phase, an ANN can understand and extract underlying patterns from observed data by adjusting the strengths (weights) of its network connections. This attribute enables ANNs to effectively approximate arbitrary functions (Xu *et al.*, no date).

The functioning of an artificial neural network (ANN) can be described as follows. Initially, the neural network receives the data variables' values as input through the input layer's nodes. The nodes within the network are interconnected through weighted connections, which are adjusted based on the network's learning and adaptive capabilities. As the data variables' values propagate through the network, the weights of the connections influence the resulting parameter values. Eventually, at the output node, the parameter value is compared to the target value, and the impact or effect is anticipated (SCAD College of Engineering and Technology and Institute of Electrical and Electronics Engineers, no date).

Similar to its biological counterpart, an artificial neural network (ANN) consists of interconnected nodes that can be likened to neurons. Three fundamental components play a crucial role in the ANN: node character, network topology, and learning rules. The node character governs the processing of signals within each node. Network topology defines the arrangement and connections between nodes. Learning rules, on the other hand, automatically determine how the weights are initially set and subsequently adjusted using various weight adjustment schemes (Khaliq, Farooq and Khan, 2022b).

### **2.3.2 Machine Learning**

Machine learning can be broadly described as computational techniques that utilize experience to enhance performance or make precise predictions. In this context, "experience" refers to the historical information accessible to the learning system, often in the form of electronic data that can be collected and analysed. Such data may consist of digitized training sets labelled by humans or other types of information acquired through interaction with the surrounding environment (Khaliq, Farooq and Khan, 2022a).

Machine learning algorithms like Xgboost, Random Forest, KNN, SVM and Decision Tree are utilized on data to forecast software quality and uncover the connection between quality and development attributes (Regulapati Akhila, A.Anusha and Dr P.Srinivasa Rao, 2022).

* **Random Forest:** Random Forest is a widely used supervised machine learning algorithm that applies to both classification and regression tasks. It operates based on the concept of ensemble learning, which involves combining multiple classifiers to address complex problems and enhance model performance (Ramayanam and A, 2022).

In Random Forest, a collection of decision trees is constructed on different subsets of the provided dataset. The algorithm takes the average of the predictions from these decision trees to enhance the accuracy of the predictions. Unlike relying on the output of a single decision tree, Random Forest considers the majority votes of predictions from all the trees and utilizes them to make the final output prediction (Seliya and Khoshgoftaar, 2007).

* **Decision Tree:** The Decision Tree is a supervised learning technique that is commonly employed for both classification and regression problems, with a particular emphasis on classification tasks. It functions as a classifier structured in the form of a tree. The internal nodes of the tree represent the features of a given dataset, while the branches represent the decision rules. Each leaf node signifies an outcome or a final prediction (Ramayanam and A, 2022).

Within the Decision Tree, two types of nodes are present: the Decision Node and the Leaf Node. Decision nodes are responsible for making decisions and possess multiple branches, which determine the subsequent paths to follow. Conversely, leaf nodes serve as the results of those decisions and do not contain any further branches.

* **K-Nearest Neighbour (KNN):** K-Nearest Neighbour (KNN) is a straightforward machine learning algorithm rooted in the supervised learning approach. It operates under the assumption of similarity between new data points and existing cases, assigning the new data to the category that most closely resembles the available categories (Prof. Rumana Anjum and Dr Madhub B, 2021).

The KNN algorithm stores all available data and classifies a new data point based on its similarity to the stored data. This enables easy classification of new data into suitable categories using the KNN algorithm. While KNN can be utilized for both regression and classification tasks, it is predominantly employed for classification problems (Khaliq, Farooq and Khan, 2022a).

* **Support Vector Machine:** The learning approach employed by Support Vector Machines (SVM) enables the interpretation of information used in categorization and regression analysis. The SVM model is created to distribute test samples in a range, aiming to separate them by a significant gap based on their respective categories. The classification of new samples is determined by which side of the gap they fall into after being mapped onto a specific area (SCAD College of Engineering and Technology and Institute of Electrical and Electronics Engineers, no date).

SVM constructs a set of hyperplanes in a non-dimensional space for correlation and classification purposes. A hyperplane with the maximum distance to the set of points within a specific class is referred to as the functional margin. The functional margin is inversely related to the generalization error, meaning that a larger functional margin corresponds to a lower error in generalization (Murphy, Kaiser and Arias, no date).

* **Xgboost:** XGBoost is a highly efficient machine learning algorithm recognized for its outstanding capabilities in tasks such as classification and regression. By combining weaker prediction models, it creates a powerful and precise model. In the realm of software quality assurance, Xgboost proves valuable in analysing data to detect patterns associated with software defects and enhancing overall quality. With its resilience and capacity to handle intricate datasets, it aids in prioritizing testing efforts and facilitating informed decision-making throughout the development process. As a result, Xgboost enhances testing procedures and contributes to the production of superior software products (Regulapati Akhila, A. Anusha and Dr P. Srinivasa Rao, 2022).

## **2.4 RELATED WORKS AND PROPOSED METHODOLOGIES**

1. David Yarowsky (Yarowsky, no date) proposed a methodology for software quality assurance that leverages unsupervised word sense disambiguation, a technique that rivals supervised methods. Unsupervised word sense disambiguation aims to resolve the ambiguity in word meanings by automatically clustering words based on their context and determining the most appropriate sense for each occurrence.

Yarowsky's methodology for software quality assurance using unsupervised word sense disambiguation presents a promising approach for enhancing the accuracy and efficiency of software development processes. Addressing potential ambiguities and misunderstandings in software artefacts contributes to the production of higher-quality software and supports the overall success of software projects.

1. Singpurwalla and Wilson's (Singpurwalla and Wilson, 1994) methodology integrates probability models, stochastic techniques, and statistical methods to enhance software quality assurance. By quantifying uncertainties, simulating software behaviour, and employing rigorous statistical analysis, this approach provides a systematic and data-driven approach to identify, analyse, and improve software quality. It offers valuable insights and actionable recommendations for software development teams and organizations striving to deliver high-quality software products.
2. Huihua Lu, Bojan Cukic, and Mark Culp (Goedicke *et al.*, 2012) introduced a methodology for software quality assurance that utilizes semi-supervised learning with dimension reduction. The authors emphasize the significance of a pre-processing strategy, specifically multidimensional scaling, to alleviate the dimensional complexity associated with software metrics. Their research demonstrates that the semi-supervised learning algorithm, in conjunction with dimension reduction, outperforms even the highly effective supervised learning algorithm, random forest, particularly when only a limited number of modules with known fault content are accessible for training.

By employing semi-supervised learning, which incorporates both labelled and unlabelled data, the proposed methodology effectively addresses the challenges posed by limited training data. The dimension reduction technique, in this case, multidimensional scaling, assists in reducing the complexity of software metrics, making them more manageable for analysis and prediction.

1. Steven Abney (Abney, no date) utilizes the Yarowsky Algorithm to propose a methodology for software quality assurance. By drawing on the Yarowsky Algorithm, which is based on unsupervised word sense disambiguation, Abney's approach aims to enhance software quality through improved understanding and interpretation of textual artefacts.
2. Hongkai Chen and Mohammad Hossain (Chen and Hossain, 2022) conducted a study that reviewed various machine learning journals to investigate the application of machine learning techniques in software quality assurance. Their study provides a comprehensive understanding of the current landscape and serves as a valuable resource for researchers and practitioners seeking to enhance software quality through the utilization of machine learning.
3. Authors Safa Omri and Carsten Sinz (Omri and Sinz, 2021) have proposed a methodology study for software quality assurance that explores various approaches in fault prediction and test case prioritization. They specifically focus on the use of deep learning algorithms for fault prediction and the application of machine learning methods for test case prioritization. The authors also review recently proposed machine learning methods for test case prioritization (TCP). Test case prioritization plays a vital role in regression testing, aiming to reduce the cost and time associated with testing while maintaining effective fault detection. Omri and Sinz examine the effectiveness of various machine learning techniques in prioritizing test cases to achieve cost savings without compromising fault detection capabilities.
4. Authors Zubair Khaliq, Sheikh Umar Farooq, and Dawood Ashraf Khana (Khaliq, Farooq and Khan, 2022a) researched software quality assurance, focusing on the challenges faced by software testers when applying AI to testing. Their study aims to identify and explain these challenges, as well as propose key contributions of AI to the future of software testing.

The research acknowledges that while AI has the potential to revolutionize software testing, it presents significant challenges for testers. The authors explore factors such as test case generation complexity, interpretability of AI models, data quality and bias, and the impact on traditional testing approaches.

1. Authors Rui Lima, António Miguel Rosado da Cruz, and Jorge Ribeiro (Lima, Rosado Da Cruz and Ribeiro, no date) conducted a methodology study for software quality assurance, focusing on the state of the art in software testing and the application of machine learning (ML) approaches and artificial intelligence (AI) algorithms. The study analyses the progress made in AI and ML methods used in software testing over the past three years by examining databases such as Scopus Elsevier, Web of Science, and Google Scholar.

The paper categorizes the algorithms used in software testing based on different test types. It also establishes relationships between the main AI approaches and the specific types of tests to which they are commonly applied, including white-box, grey-box, and black-box software testing.

1. Study authors Prof. Rumana Anjum and Dr. Madhu (Prof. Rumana Anjum and Dr Madhub B, 2021) conducted a study for software quality assurance that explores the application of Artificial Intelligence (AI) in software testing. The study aims to examine the key pillars of AI and their potential use in software testing, while also addressing the challenges, issues, and needs associated with AI-based software testing.

The study delves into the discussion of various AI key pillars that can significantly contribute to software testing. These pillars encompass a range of AI techniques, including machine learning, natural language processing, expert systems, and data mining. The authors analyse how these AI pillars can be effectively applied to enhance different aspects of software testing, such as test case generation, fault detection, and test optimization.

1. Authors Anna Trudova, Michal Dolezel, and Alena Buchalcevova (Trudova and Dolezel, no date) conducted a study on software quality assurance, specifically focusing on the implementation of artificial intelligence (AI) techniques in software test automation. Their study aimed to conduct a systematic literature review (SLR) to identify research studies that reported the utilization of AI techniques in software test automation.

The reviewed primary studies highlighted various improvements achieved by leveraging AI techniques in software test automation. These improvements included enhanced reusability of test cases, reduction in manual effort, improved test coverage, and increased detection of faults and vulnerabilities.

1. Authors Prajwal Wable, Mangesh Kumar, Shubham Thorat, Omkar Gaikwad, and Prof. S. S. Kolte (Wable *et al.*, 2021) have proposed a methodology study for software quality assurance that focuses on generating unit test cases by leveraging machine learning. Their approach involves learning from developer-written test cases and utilizing machine-learning techniques to translate code snippets into corresponding test cases. The training dataset for the machine learning model is mined from open-source repositories hosted on GitHub.

By utilizing developer-written test cases as a source of knowledge, the authors aim to improve the efficiency and effectiveness of unit test case generation. They recognize the value of the existing test cases created by developers and harness this knowledge to automate the process of generating additional test cases. By leveraging machine learning, the proposed approach can learn patterns and relationships between code snippets and their associated test cases, enabling the model to generate accurate and relevant unit test cases.

1. Authors Michael R. Lyu and Allen Nikora (Lyu and Nikora, no date) proposed a methodology study for software quality assurance that addresses the predictive accuracy problem in software reliability modelling. Their study introduces a general combination approach that utilizes linear combinations of existing software reliability models, referred to as component models, to create a series of linear-combination models.

Rather than relying on a single model, the authors advocate for the use of multiple component models combined through linear combinations. In an experimental investigation using industrial project data, the set of linear combination models demonstrated superior performance compared to the individual component models.

1. Authors Christian Murphy, Gail Kaiser, and Marta Arias (Murphy, Kaiser and Arias, no date) have proposed a methodology study for software quality assurance that focuses on testing and debugging algorithm implementations without a reliable test oracle. They present a conceptual framework specifically designed for algorithm implementations where determining the "right" answer can be challenging, particularly in machine learning applications where precise input/output specifications may not exist.

The proposed framework simplifies the process of conducting regression and comparison testing, especially in scenarios where multiple implementations of the same algorithm are available. By providing a structured approach, it enables testers to assess the performance and accuracy of different algorithm implementations. The framework facilitates the identification of discrepancies, inconsistencies, or improvements between various implementations, aiding in the refinement and enhancement of the software.

1. Professor (Dr.) Om Prakash Sangwan and Deepak Nandal (Deepak Nandal and Prakash Sangwan, no date) have proposed a methodology study for software quality assurance that focuses on optimizing estimation models such as Constructive Cost Model (COCOMO) and Function Point Analysis. The objective is to improve the performance of these models by leveraging fuzzy logic, neural networks, and metaheuristic techniques available in the market.

The study explores the optimization of COCOMO using the BAT (Bat Algorithm) optimization technique. The BAT algorithm mimics the hunting and routing behaviour of bats, utilizing echolocation properties. This algorithm is beneficial for optimization as it dynamically switches between the exploration and exploitation phases.

Additionally, the research proposes an improved BAT algorithm that incorporates a random walk during the exploration phase to enhance convergence towards global optima. This improved BAT algorithm is applied to optimize the constants used in the COCOMO model, resulting in more accurate software cost estimation.

Furthermore, the study introduces a hybrid BATGSA algorithm that combines the improved BAT algorithm with the Gravitation Search Algorithm (GSA) to optimize the COCOMO model. The integration of GSA improves the exploration phase by considering the gravitation force, which affects the bat's velocity.

1. Authors Ramayanam Venkata Sai Prathap and A. Suneetha (Ramayanam and A, 2022) have proposed a methodology study for software quality assurance with a focus on improving estimation accuracy using relevant features from a large dataset. Their study utilizes feature selection methods and correlation analysis to achieve higher accuracy in software quality estimation. Additionally, they explore the effectiveness of machine learning algorithms, such as K-Nearest Neighbors (KNN), Random Forest, and Decision Tree, in predicting software quality and establishing the relationship between quality and development attributes.

The authors conducted experiments to evaluate the performance of the machine-learning algorithms in estimating software quality. By analyzing the data and utilizing the selected features, their methodology aims to provide more accurate estimations of software quality levels. The study explores the potential of machine learning algorithms to uncover the intricate relationship between software quality and various development attributes.

1. Authors Ram Chillarege, Shriram Biyani, and Jeanette Rosenthal (Chillarege, Biyani and Rosenthal, no date) conducted a study addressing a critical need in the field of fault-tolerant computing. With the distribution of commercial software to numerous customers and software being a significant contributor to outages and failures, accurately measuring the actual failure rate perceived by end users becomes crucial. This parameter has a profound influence on the overall design for dependability, yet it has been challenging to measure directly, with limited data available.

This paper aims to fill the gap by providing quantitative results in measuring the failure rate. The study focuses on two releases of a software product with a substantial codebase, which has been distributed to a large customer base. The distinguishing aspect of this study is that it leverages failures reported by customers worldwide, directly measuring the failure rate.

1. Authors Chun Shan, Hongjin Zhu, Changzhen Hu, Jing Cui, and Jingfeng Xue (International Conference on Computer Science and Network Technology 4. 2015 Harbin *et al.*, 2015) have proposed a methodology for software quality assurance that addresses the challenges faced by the Locally Linear Embedding-Support Vector Machines (LLE-SVM) model in software defect prediction. To overcome these challenges, they introduce an improved version called the Improved Locally Linear Embedding-Support Vector Machines (ILLE-SVM) model.

The ILLE-SVM model incorporates several enhancements to address the limitations of the LLE-SVM model. One improvement is the utilization of a coarse-to-fine grid search algorithm to optimize the model's parameters. This algorithm ensures high parameter accuracy and reduces the time required for parameter optimization by gradually narrowing the search scope and enlarging the parameter steps.

1. Authors Wei Wu, Kun Han, Chengming He, and Shujian (Huang, Zuo, *et al.*, no date) Wu have proposed a methodology study for software quality assurance that focuses on improving predictive quality through a dynamically-weighted software reliability combination model. The objective of this model is to enhance the dependability of predictions.

The process of constructing the combination model involves two main steps: the selection of component models and the determination of weight assignment. In selecting the component models, criteria such as the life cycle phase, trend exhibited by failure data, the validity of model assumptions, and predictive biases of the models are taken into account. These criteria help ensure that the chosen models are appropriate for the specific context and provide accurate predictions.

The determination of weights for the component models is based on the prequential likelihood approach. This approach takes into consideration both the prior knowledge and the observed data, resulting in a robust weight assignment that optimally combines the predictions from the component models.

1. Authors Yumei Wu and Risheng Yang (Xu *et al.*, no date) have proposed a methodology study for software quality assurance that introduces a novel software reliability prediction method based on the General Regression Neural Network (GRNN). This method offers an alternative approach to predicting software failures without the need to construct statistical models like traditional software reliability models, thereby avoiding the complexities associated with solving multivariate likelihood equations.

The proposed method takes into account test coverage, which enhances the accuracy of the predictions. By utilizing the probability plot technique and least square fitting, the probability distribution functions of the original failure data are determined. This enables the generation of a large amount of simulated data, facilitating reliable predictions. This approach addresses the challenge of having a small sample size of test failure data, which often leads to inaccuracies in predictions.

1. Authors Regulapati Akhila, A. Anusha, Dr P. Srinivasa Rao (Regulapati Akhila, AA. AnushaandDr .P.Srinivasa Rao, 2022) have conducted a methodology study for software quality assurance. Their study aims to improve the accuracy of estimation by utilizing relevant features from a large dataset. To achieve higher accuracies, they employ a feature selection method and correlation matrix analysis.

In addition to these techniques, the authors explore the effectiveness of recent methods that have demonstrated success in other prediction tasks. They apply machine learning algorithms such as Xgboost, Random Forest, and Decision Tree to the dataset to predict software quality and uncover the relationship between quality and development attributes.

1. Authors Ebubeogu Amarachukwu Felix and Sai Peck Lee (Felix and Lee, 2017) proposed a methodology study for software quality assurance that focuses on defect acceleration and its correlation with the number of defects. The study presents a method utilizing predictor variables derived from defect acceleration, including defect density, defect velocity, and defect introduction time. The goal is to determine the correlation between each predictor variable and the number of defects.

To achieve this, an integrated machine machine-learning based on regression models is employed. These regression models are constructed using the predictor variables derived from defect acceleration. The study conducted an experiment using ten different datasets obtained from the PROMISE repository, consisting of 22,838 instances.

The results of the experiment demonstrate the effectiveness of the methodology. The regression model constructed using the average defect velocity achieved an adjusted R-square of 98.6% with a p-value of less than 0.001. The average defect velocity exhibited a strong positive correlation with the number of defects, as indicated by a correlation coefficient of 0.98.

1. Author Norman F. Schneidewind (Schneidewind and Swss, no date) proposed a methodology study for software quality assurance that involves the development of a model to validate and apply metrics for quality control. The study utilizes the example of the Space Shuttle flight to validate the metrics and assess their effectiveness in achieving quality objectives.

The validation of metrics is conducted following a previously developed metrics validation methodology. The study focuses on a specific quality factor and aims to demonstrate the applicability and reliability of the metrics in measuring and controlling software quality.

1. Authors Ajmer Singh, Parul Bhandari, and Rajvir Singh (Singh *et al.*, 2018) have proposed a methodology study for software quality assurance that focuses on the automatic selection of test cases using the NSGA-II algorithm. The study specifically applies this methodology to the testing of vending machine software based on object-oriented coupling metrics.

The vending machine serves as a practical case study in which test cases are manually written. However, the researchers aim to improve the efficiency and effectiveness of test case selection by applying the Multi-objective Evolutionary Algorithm, NSGA-II. This algorithm optimizes the objective functions of coupling measures (to be maximized), code coverage (to be maximized), and execution time (to be minimized).

By incorporating the NSGA-II algorithm, the study seeks to automatically prioritize test cases based on their importance and relevance. This approach can enhance the overall quality assurance process by selecting optimal test cases that maximize coupling measures and code coverage while minimizing execution time.

1. Authors P.D. Ratna Raju, Suresh Cheekaty, and Harish Babu Kalidasu (Ratna, Cheekaty and Kalidasu, no date) proposed a methodology study for software quality assurance, focusing on testing object-oriented (OO) software based on UML specifications created in Rational Rose. They highlight the unique challenges associated with testing OO software compared to software developed using procedural languages.

In the past, testing methods for OO software were often extensions of existing methods for conventional software, but they were found to be inadequate. To address this, the authors introduce a new technology and tool that enables testing at both the unit (class) level and integration level, with an added maintenance-level component. They have applied this tool to sample Rational Rose files, and the results have been deemed satisfactory.

1. Authors Vishnu Yash Tripathi and Veer Bhadra Pratap Singh (Tripathi, Bhadra and Singh, 2016) proposed a methodology study for software quality assurance that focuses on generating test cases for object-oriented software using UML diagrams, particularly Sequence diagrams. The study proposes the use of an evolutionary algorithm, specifically a Genetic Algorithm, to optimize the generated test cases.

The authors recognize the importance of generating effective and efficient test cases for object-oriented software, and they leverage UML diagrams, specifically Sequence diagrams, as a basis for generating these test cases. Sequence diagrams provide valuable insights into the interactions between objects, allowing for the identification of potential test scenarios.

To optimize the generated test cases, the study proposes the application of a Genetic Algorithm. This evolutionary algorithm mimics natural selection and genetic processes to iteratively refine and improve the test cases. By employing Genetic Algorithm, the study aims to enhance the quality and effectiveness of the generated test cases, ensuring comprehensive coverage of critical scenarios and potential faults.

1. Authors Meenakshi Vanmali, Mark Last, and Abraham Kandel (Vanmali, Last and Kandel, no date) have proposed a methodology study for software quality assurance that introduces a novel concept of utilizing an artificial neural network as an automated oracle for tested software systems. The study focuses on training a neural network using the backpropagation algorithm on a set of test cases applied to the original version of the system.

The training process of the neural network follows a "black-box" approach, where only the inputs and outputs of the system are presented to the algorithm. This approach allows the network to learn and generalize patterns without knowledge of the internal workings of the system. Once trained, the neural network can serve as an artificial oracle to evaluate the correctness of outputs generated by new and potentially faulty versions of the software.

1. Authors Ohood Al-Masri and Wedad Al-Sorori (Al-Masri and Al-Sorori, 2022) conducted a methodology study for software quality assurance that focuses on optimizing the generation of test cases using a teaching-learning-based optimization algorithm. The study compares the results of this algorithm with other state-of-the-art methods based on path coverage for ten Java programs.

The main objective of the proposed algorithm is to minimize the number of test cases required to cover all code paths in unit tests. By optimizing the test case generation process, the algorithm aims to achieve full coverage of the code paths while minimizing the number of test cases needed.

1. Authors Kun Han, Jun-Hai Cao, Shou-Hua Chen, and Wei-Wei Liu (Huang, Liu, *et al.*, no date) proposed a methodology study for software quality assurance that focuses on software reliability prediction based on the software development process. The study introduces a software reliability prediction method that utilizes Keene's Development Process Prediction Model (DPPM), the Rayleigh model, and the computer-Aided Software Reliability Estimation (CASRE) tool.

The proposed methodology aims to progressively enhance the prediction of software reliability by incorporating these prediction models and tools. By utilizing Keene's DPPM, the study takes into account the software development process and its impact on software reliability. The Rayleigh model is employed to model the software failure distribution, which helps in estimating the reliability characteristics.

Furthermore, the authors utilize the CASRE tool to assist in the software reliability estimation process. This tool aids in analysing software reliability data and providing insights into potential failure patterns and trends.

1. Authors Kirandeep Kaur and Vinay Chopra (Kaur and Chopra, 2014) proposed a methodology study for software quality assurance that focuses on the automatic generation of test cases by analyzing the dynamic behaviour of UML diagrams during the design phase of the software development life cycle (SWDLC). The study introduces the use of a multi-objective genetic algorithm as an evolutionary algorithm for this purpose.

The paper reviews the existing approaches of automatic test case generation and highlights the use of single-objective genetic algorithms in previous studies. Building upon this prior work, Kaur and Chopra propose the application of a multi-objective genetic algorithm to enhance the automatic testing process.

1. Authors R. Jayanthi and Lilly Florence (Jayanthi and Florence, 2019) proposed a methodology study for software quality assurance that focuses on a combined approach for software defect prediction and prediction of software bugs. The study introduces a concept of feature reduction and incorporates artificial intelligence techniques to improve the accuracy of predictions.

In their proposed approach, feature reduction is achieved through the well-known principle component analysis (PCA) scheme. The authors enhance the PCA data reconstruction by incorporating maximum-likelihood estimation to reduce errors. This improved feature reduction process helps to identify the most relevant features for defect prediction.

To make predictions, the authors employ a neural network-based classification technique. This technique utilizes the reduced feature set obtained through PCA and applies it to the classification model. The resulting predictions are evaluated for their accuracy and effectiveness.

1. Authors Naeem Seliya and Taghi M. Khoshgoftaar (Seliya and Khoshgoftaar, 2007) proposed a methodology study for software quality assurance that focuses on utilizing semi-supervised learning with the Expectation Maximization (EM) algorithm for software quality estimation when there is limited fault-proneness data available.

The study aims to investigate the hypothesis that the knowledge contained in the software attributes of unlabelled program modules can enhance the accuracy of software quality estimation. To validate this hypothesis, software data from a large NASA software project is collected and used during the semi-supervised learning process.

The proposed methodology is evaluated using multiple test datasets collected from other NASA software projects. The performance of the software quality model, trained using the EM-based semi-supervised learning scheme, is compared to models trained solely with the available set of labelled program modules.

1. Authors C. Lakshmi Prabha and Dr N. Shivakumar (SCAD College of Engineering and Technology and Institute of Electrical and Electronics Engineers, no date) proposed a methodology study for software quality assurance focused on the prediction of software defects. The study emphasizes the importance of accurately predicting defects at an early stage. The challenge lies in dealing with high-dimensional software-defected datasets.

To address this challenge, the authors propose a hybridized approach that combines Principal Component Analysis (PCA), Random Forest, Naïve Bayes, and Support Vector Machine (SVM) software frameworks. They conduct software analysis using the WEKA simulation tool with five datasets: PC3, MW1, KC1, PC4, and CM1.

The study includes a systematic research analysis, evaluating various performance metrics such as confusion, precision, recall, and recognition accuracy. These metrics are measured and compared with existing schemes to assess the effectiveness of the proposed approach.

The analytical analysis conducted by Prabha and Shivakumar indicates that their proposed hybridized approach yields more reliable and effective solutions for predicting software defects. The combination of PCA, Random Forest, Naïve Bayes, and SVM frameworks shows promising results in terms of accuracy and performance.

1. Authors Madhumita Panda, Sujata Dash, Anand Nayyar, Muhammad Bilal, and Raja Majid Mehmood (Panda *et al.*, 2020) proposed a methodology study for software quality assurance that focuses on a concrete model-based testing framework. The framework utilizes a combination of UML behavioural state chart models and a hybrid version of two popular nature-inspired algorithms: Firefly Algorithm (FA) and Differential Algorithm (DE).

The study employs the hybrid FA-DE algorithm to generate optimized test suits for the benchmark triangle classification problem. By integrating these nature-inspired algorithms with the UML models, the framework aims to improve the efficiency and effectiveness of the test case generation process. The hybrid algorithm offers benefits such as enhanced time complexity, improved exploration and exploitation capabilities, and increased variations in test case generation.

1. Authors Madhumita Panda and Sujata Dash (Panda and Dash, 2020) proposed a methodology study for software quality assurance that focuses on test case generation using model-based testing of object-oriented programs. Their approach involves utilizing well-known metaheuristic algorithms and their hybridizations to enhance the test case generation process.

The study utilizes seven well-known metaheuristic algorithms, namely GA, DE, PSO, ABC, CS, FA, and GSA, along with three popular hybrid metaheuristics: CS-SA, FA-DE, and PSO-GSA. These algorithms are applied to a benchmark triangle classification problem to evaluate their performance.

The results of the proposed approaches are compared to identify robust metaheuristic algorithms for test case generation in object-oriented programs. The experimental findings demonstrate that the hybrid metaheuristic algorithms outperform the single metaheuristic algorithms in terms of test case generation. These hybrid algorithms overcome the limitations of single metaheuristic algorithms by exploring both local and global optimal solutions within the large search space.

## **2.5 LIMITATIONS RELATED TO AUTOMATED SOFTWARE QUALITY ASSURANCE SYSTEM**

While software quality assurance using machine learning presents promising opportunities, it also has certain limitations that researchers should consider. Some of the research limitations in this field include:

1. Limited Availability of Quality Data: Machine learning algorithms heavily rely on high-quality and labelled data for training. Obtaining a sufficient amount of accurate and representative data can be challenging, especially for certain niche domains or proprietary software systems.
2. Generalization Challenges: Machine learning models may struggle to generalize well to unseen or unfamiliar software systems. The performance of these models heavily depends on the similarity between the training data and the target application. Therefore, the effectiveness of machine learning techniques in software quality assurance may vary across different contexts.
3. Interpretability and Explainability: Some machine learning algorithms, such as deep learning models, often operate as black boxes, making it difficult to interpret and explain their decisions. This lack of interpretability can hinder the trust and acceptance of these models in critical software quality assurance tasks.
4. Data Bias and Imbalanced Datasets: Biases present in the training data can result in biased models. Additionally, imbalanced datasets, where one class dominates the majority of samples, can lead to skewed model performance and a focus on the majority class, potentially ignoring important minority class defects.
5. Need for Domain Expertise: While machine learning algorithms automate certain aspects of software quality assurance, domain expertise remains crucial. Understanding the software development process, domain-specific quality requirements, and context-specific nuances is essential for the effective application of machine learning techniques.
6. Scalability and Computational Requirements: Training and deploying complex machine learning models can be computationally demanding, requiring significant computational resources and time. Scaling these techniques to handle large-scale software systems with numerous features and millions of lines of code remains a challenge.
7. Ethical Considerations: Applying machine learning in software quality assurance raises ethical concerns related to privacy, bias, fairness, and accountability. It is important to address these ethical considerations and ensure the responsible and ethical use of machine learning algorithms in the context of software quality assurance.

## **2.6 SUMMARY**

Software quality assurance using machine learning algorithms is an evolving and promising field. The literature reviews provide valuable insights into methodologies, approaches, and challenges.

These reviews offer an overview of the current state of automated software quality assurance using machine learning. They serve as valuable resources for researchers, practitioners, and organizations interested in leveraging ML to enhance software quality.

One important area highlighted is fault prediction in software engineering. The reviews emphasize the development of techniques and metrics to improve the accuracy of fault prediction. Effective test case generation and selection of appropriate testing approaches based on software requirements are also emphasized.

Machine learning algorithms such as XGBoost, Random Forest, and Decision Tree are showcased for their exceptional performance in software quality assurance. They enable data analysis, pattern identification, and overall improvement in software quality.

The application of artificial neural networks (ANNs) in software quality assurance is another significant aspect. ANNs, inspired by biological neural networks, learn and extract patterns from data, enabling accurate approximations of complex functions.

The reviews also discuss the benefits of grey-box testing, which combines elements of white-box and black-box testing. Grey-box testing focuses on the external behaviour of software while considering limited knowledge of its internal structure.

Additionally, the use of metaheuristic algorithms and their hybrids in software quality assurance shows promise for test case generation, model-based testing, and defect prediction. These algorithms enhance accuracy, coverage, and efficiency in quality assurance tasks.

In summary, these literature reviews provide comprehensive insights into software quality assurance using machine learning. They highlight methodologies, challenges, and advancements, offering guidance for further research and development in this field.

# **CHAPTER 3: SYSTEM ANALYSIS AND DESIGN**

Software Quality Assurance is a crucial activity employed at various stages of the software development lifecycle. It serves the purpose of guiding project planning for quality assurance practices and establishing benchmarks for comparison. In the past, two methods, namely Multiple Criteria Linear Programming and Multiple Criteria Quadratic Programming, were commonly used to estimate software quality. Additionally, explorations were conducted using C5.0, SVM, and Neural Networks for this purpose. However, with the aim of enhancing estimation accuracy, I embarked on a research endeavour focused on leveraging relevant features from a large dataset.

To achieve improved accuracy, I adopted a meticulous approach involving feature selection and correlation matrix analysis. This enabled me to attain higher accuracies in predicting software quality. Furthermore, I conducted experiments with cutting-edge methods that have demonstrated success in other prediction tasks. The application of machine learning techniques, such as K-nearest neighbours (Knn), Random Forest, and Decision Trees, facilitated the prediction of software quality and revealed insightful relationships between quality and various development attributes.

The culmination of my efforts resulted in compelling experimental outcomes, showcasing the efficacy of machine learning algorithms in accurately estimating software quality levels. By leveraging these advanced methodologies, stakeholders can make informed decisions about project quality and enhance overall software development processes.

## **3.1 REQUIREMENTS**

The Software Requirement Specification (SRS) serves as a comprehensive depiction of the intended behaviour of the system under development. It should encompass two vital components: the definition of user requirements and the specification of system requirements. The Software Requirement Specification leaves no room for ambiguity, providing a thorough account of the software's intended functionality and performance expectations.

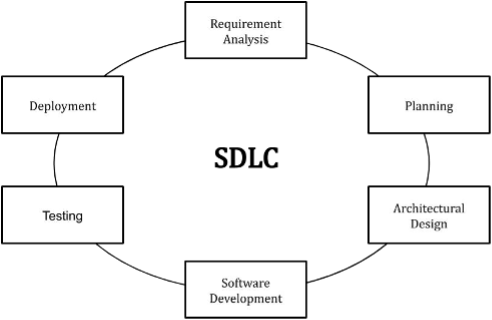


Figure 3 SDLC REQUIREMENTS CYCLE

### **3.1.1 FUNCTIONAL REQUIREMENTS**

1. Data Collection and Integration from various sources.
2. Pre-processing and Data Cleaning for high-quality data.
3. Feature Extraction and Selection to identify relevant metrics.
4. Model Training and Validation using machine learning algorithms.
5. Anomaly Detection for identifying unusual patterns.
6. Reporting and Visualization of predictions.
7. Scalability and Performance for handling large datasets.
8. User-Friendly Interface for easy interaction.

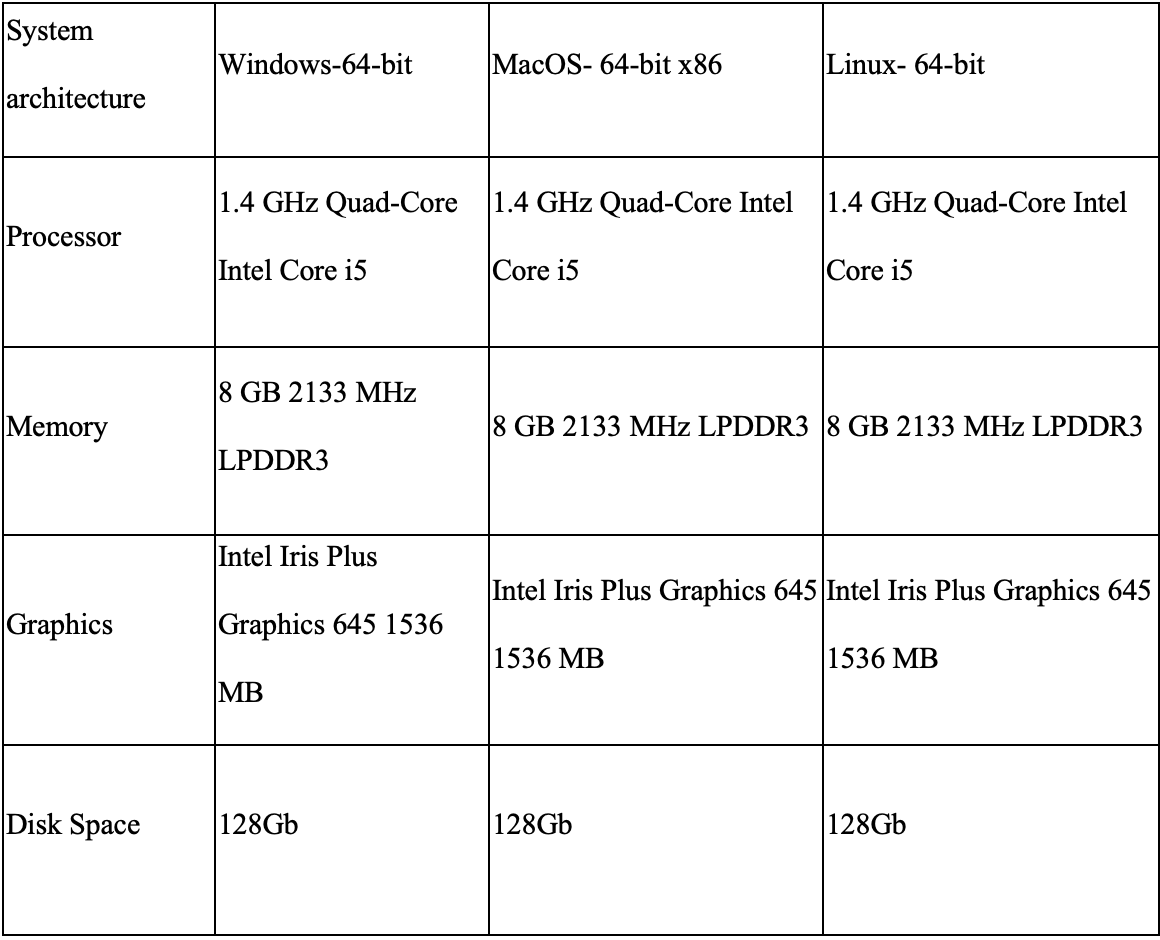
### **3.1.2 NON-FUNCTIONAL REQUIREMENTS**

1. Maintainability
2. Scalability
3. Accuracy
4. Performance
5. Usability
6. Compliance

### **3.1.3 HARDWARE REQUIREMENTS**

These are the computer system specifications to be able to process the machine learning models smoothly and efficiently.

Table 1 Hardware requirements for the proposed system



### **3.1.4 SOFTWARE REQUIREMENTS**

Given Python was chosen as the language for our implementation since it is the most widely used data science programming language in the world today. It's a user-friendly, open-source language that dates back to at least 1991. Object-oriented programming is included in this versatile and wide language. Other programming paradigms are also supported, including functional, structured, and procedural programming. It's a better and quicker way for data processing, machine learning, transfer learning, robotics, the Internet of Things, and other processes that don't require many iterations. Natural data processing and data mining become straightforward tasks with Python's features. Python also generates a CSV file, which makes reading data from a spreadsheet easier for programmers.

The version of Python used was Python 3.9.7. The following libraries in Python were used for the implementation of the model:

1. **OS module**: Offers a portable means of accessing operating system-specific functions.
2. **NumPy**: A Python module that provides support for large, multidimensional arrays as well as a wide range of powerful mathematical methods for manipulating them.
3. **Seaborn**: This is a Matplotlib-based Python visualization package. It has a user-friendly interface that allows users to create graphical and instructive statistics representations.
4. **Matplotlib**: Matplotlib is a Python library that allows you to make dynamic graphs.
5. **SciPy**: SciPy is a Python module that can be used to solve statistical and technical problems. It's based on the NumPy library and allows you to manipulate and visualize data with a number of high-level commands.
6. **Scikit-learn:** The most advanced and useful machine learning library is Scikit-learn (Sklearn). Through a Python consistency interface, it delivers a set of effective data science modeling capabilities, including that of segmentation, extrapolation, clustering, and data preprocessing.
7. **Pandas**: This is a Python programming dependency that offers high performance, data structures, and data analysis capabilities that are seamless.

## **3.2 SYSTEM DESIGN**

In the System Design section, a detailed plan is presented for the Automated Software Quality Assurance System, which seamlessly integrates Machine Learning capabilities. This blueprint encompasses the system's architecture, components, and interactions, offering a comprehensive view of how it will meet the defined functional and non-functional requirements. The design strategically tackles the complexities of gathering, processing, and analysing software development data, resulting in precise predictions and valuable insights into software quality for all stakeholders. This section extensively explores the system's design decisions, algorithms, data flow, and seamless integration with current development tools, emphasizing its scalability, security, and user-friendly attributes.

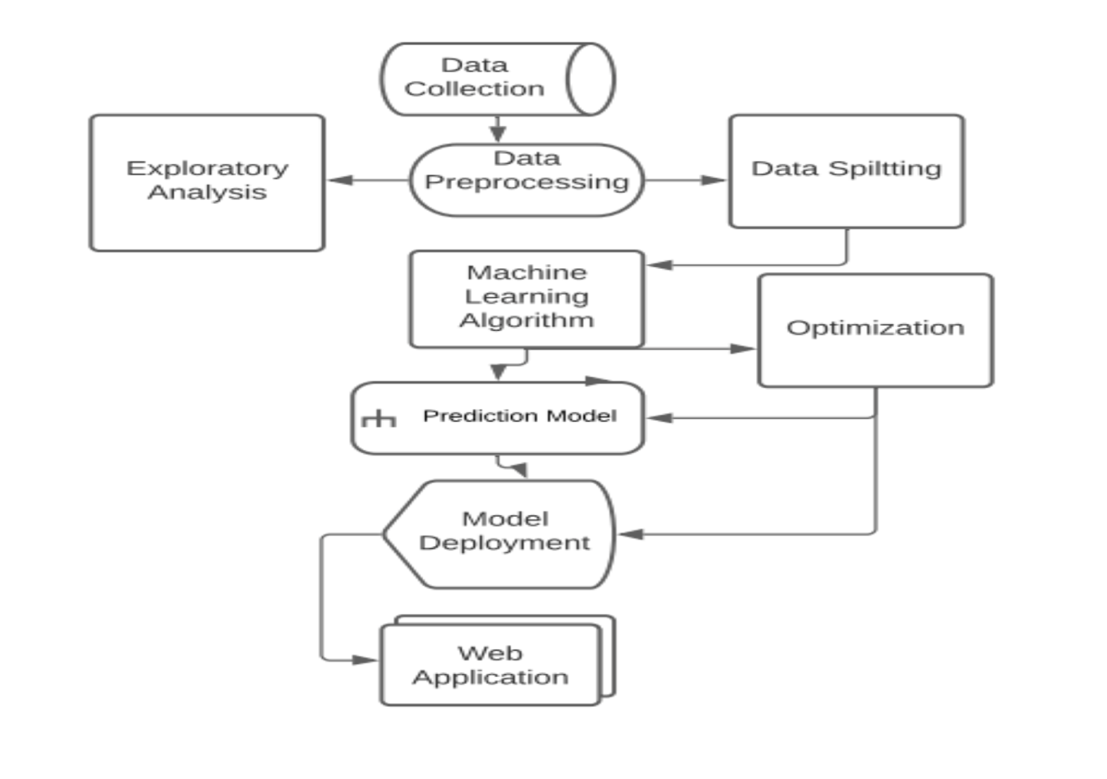


Figure 4 Flowchart of the designed system

The proposed system employs [Logistic Regression], [Random Forest Classifier], [K-Nearest Neighbor] and [Extreme Gradient Boosting] to provide a comprehensive approach to software quality assurance. Following that, the model is built with the data set before being deployed into live a system.

**Dataset Collection**

Data collection is a method of gathering information based on scientific understanding that is systematic and thorough. It is used to generate replies to research questions. It's also a way of gathering and analyzing data in order to find answers to urgent problems and evaluate the results. It focuses on knowing everything there is to know about a certain subject. In the discipline of machine learning, datasets are crucial. Data collection is one of the most challenging activities, especially when dealing with financial issues like card fraud. The dataset is linked in the project report's appendix section.

**Data Preprocessing and Exploratory Analysis**

This is the process of converting unstructured data into a format that can be used. This is a crucial stage in data mining since programmers are unable to deal with raw data. Before using deep learning or data mining methods, make sure the data is of excellent quality.

The data was found to be substantially skewed, with the minority class belonging to fraud instances. The issues were fixed by utilizing SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic data points and framing new train and test sets accordingly.

**Data Splitting**

For the project, the dataset is separated into two sections: a train set and a test set. The train set is used to create a classification model, whereas the test set is used to verify the performance of the trained model. The data was evenly dispersed among the machine learning models used in the comparison. The data must be separated to avoid overfitting the models, which is a typical concern when creating models.

**Machine Learning Algorithms**

Sklearn was used to import the machine learning algorithms. Five machine algorithms were used in this investigation; K-Nearest Neighbor, Logistic Regression, Extreme Gradient Boosting, and Random Forest Classifier are some of the techniques used. These are all classification algorithms. All of the algorithms were constructed, analyzed, and compared using testing measures such as accuracy, sensitivity, specificity, F1 score, precision, and confusion matrix.

**Prediction model**

This is the machine learning model that is selected following a comparison of multiple assessment measures to determine which is the best for model deployment. In this project's chapter 4, the results are reported and described.

**Model Deployment**

Machine learning model deployment requires not only bringing models into production, but also making them accessible to other systems within the Software Engineering institutions or on the internet so that they may receive data and return predictions. To allow for numerous live assessments and testing, the various models were stored and then integrated in the web application.

**Web Application Deployment**

Four of the algorithms used to create the system were used in this webapp; [Logistic Regression], [Random Forest Classifier], [K-Nearest Neighbour] and [Extreme Gradient Boosting].

Flask which is an open-source app framework written in Python, was used to deploy the machine learning models. It facilitates the rapid development of web apps for data science and machine learning. It works with all main Python libraries.

### **3.2.1 ALGORITHMS USED**

**XGBOOST CLASSIFIER:**

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that has gained significant popularity in various domains, including software quality assurance. Leveraging the strengths of gradient boosting and tree-based methods, XGBoost is well suited for handling complex datasets, making it an ideal candidate for automating software quality assurance processes.

Steps in using the algorithm:

1. Train the XGBoost model using the dataset
2. Optimize model performance through hyperparameter tuning
3. Validate the model's performance using cross-validation
4. Assess model performance using relevant metrics like; accuracy, precision, recall, and F1 score.

**RANDOM FOREST CLASSIFIER:**

The Random Forest algorithm has emerged as a powerful and popular machine learning technique for various prediction tasks, including software quality assurance. Leveraging its ensemble learning capabilities, the Random Forest algorithm proves to be an effective and versatile choice for automating software quality assessment. Random Forest works by constructing multiple decision trees and combining their predictions to make more accurate and robust estimations. This allows the algorithm to handle complex and nonlinear relationships within the software development data, resulting in reliable quality predictions.

Steps in using the algorithm;

1. Building Trees: A set of decision trees is generated, each trained on distinct data subsets. Each tree is trained on a randomly selected subset of the data, and each split in the tree is made by selecting the feature that maximizes the information gain.
2. Ensemble: Once there are multiple decision trees, they combine to create a random forest. The random forest makes predictions by aggregating the predictions of each decision tree. For classification problems, the prediction is based on the mode of the class predictions of the individual trees. For regression problems, the prediction is based on the average of the predictions of the individual trees.
3. Prediction: Once the random forest is trained, it can be used to predict the quality of new software.

**DECISION TREE CLASSIFIER:**

Decision Trees are powerful and interpretable machine learning models that have proven to be valuable in the domain of automated software quality assurance. They are binary tree-like structures that facilitate decision-making based on a sequence of attribute tests, leading to the prediction of software quality metrics. Decision Trees offer a transparent decision-making process that allows stakeholders to easily comprehend the factors influencing software quality predictions. Each node represents a specific attribute test, enabling a clear path from the root to the leaf node, where the final prediction is made.

**LOGISTIC REGRESSION:**

Logistic Regression is a well-established and widely used statistical technique that can be harnessed for automated software quality assurance. Despite its name, it is primarily employed for binary classification tasks, making it suitable for predicting software quality metrics that fall into distinct categories.

Steps in using the algorithm:

1. Model Training: After feature selection, the logistic regression model is trained using labeled training data. This enables the model to make probabilistic predictions and classify software instances into the appropriate categories based on the learned patterns from the training data.
2. Prediction: Once the logistic regression model is trained, it becomes capable of predicting the quality of new software instances. The probability score allows us to assess the confidence level of the prediction.
3. Evaluation: Assess model performance using relevant metrics like; accuracy, precision, recall, and F1 score.

**K-NEAREST NEIGHBOUR (KNN):**

K-Nearest Neighbors (KNN) is a non-parametric and simple yet powerful machine learning algorithm that can be effectively employed for automated software quality assurance. KNN is primarily used for classification tasks and is particularly suitable for predicting software quality metrics with categorical outcomes. Using K-Nearest Neighbors (KNN) for Automated Software Quality Assurance follows a structured process:

1. Choosing k: Selecting an appropriate value for 'k,' which represents the number of nearest neighbors considered when making predictions. This can be determined using techniques such as cross-validation to find the optimal value for k that yields the best performance.
2. Calculating Distances: For each new software instance, the algorithm calculates the distance to all other data points in the training set. The most commonly used distance metric is the Euclidean distance, although other metrics can also be utilized based on the specific context.
3. Finding the k-Nearest Neighbors: Based on the computed distances, the algorithm identifies the k-nearest neighbors, which are the data points in the training set that are closest to the new instance being evaluated.
4. Prediction: For classification tasks, the prediction is made by considering the majority class among the k-nearest neighbors. In contrast, for regression tasks, the prediction is obtained by taking the average value of the target attribute for the k-nearest neighbors.
5. Evaluation: To assess the performance of the KNN algorithm, it is evaluated on a validation set using metrics like accuracy, precision, recall, and F1 score.

### **3.2.2 USE CASE DIAGRAM**

A use case diagram is a type of behavioural diagram in the Unified Modelling Language (UML) that originates from Use-case analysis. Its primary goal is to provide a visual representation of the system's functionalities in relation to actors, use cases (representing their goals), and the interdependencies among these use cases. The main objective of a use case diagram is to illustrate the specific system functions executed for each actor involved. Additionally, the roles of these actors within the system can be clearly depicted using this diagram.

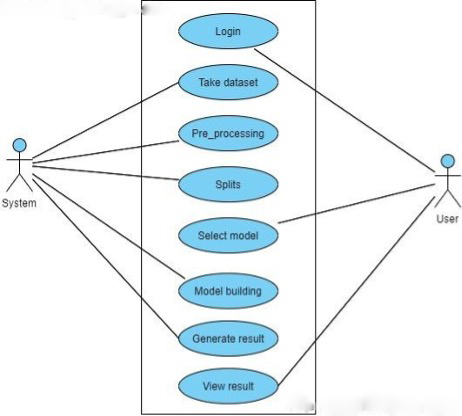


Figure 5 Use Case Diagram for the Developed System

### **3.2.3 CLASS DIAGRAM**

In the field of software engineering, a class diagram within the Unified Modeling Language (UML) is a static structure diagram that presents a comprehensive view of a system's structure. It achieves this by visualizing the system's classes, their associated attributes, operations (or methods), and the relationships that exist between these classes. This diagram is instrumental in understanding the distribution of information across various classes within the system.



Figure 6 Class Diagram of the system

### **3.2.4 COLLABORATION DIAGRAM**

In a collaboration diagram, the sequence of method calls is represented using numbered techniques, indicating the order in which the methods are invoked. To illustrate this, we'll use the order management system as an example. The method calls in a collaboration diagram are similar to those in a sequence diagram. However, the key distinction lies in the fact that while the sequence diagram focuses solely on the sequence of method calls, the collaboration diagram goes further to depict the organization of objects involved in the system.

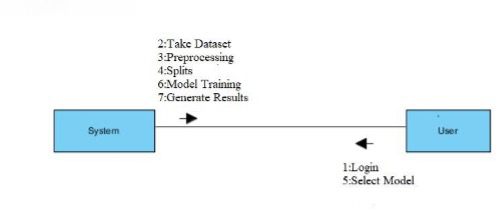


Figure 7 Collaboration Diagram of the system

**3.2.5 ACTIVITY DIAGRAM**

Activity diagrams are graphical depictions of workflows, illustrating step-by-step activities and actions with capabilities for handling choices, iterations, and concurrency. Within the Unified Modelling Language (UML), these diagrams are utilized to describe the sequential workflows of system components in both business and operational contexts. They present a clear representation of the flow of control, providing a visual understanding of the various activities and interactions occurring within the system.

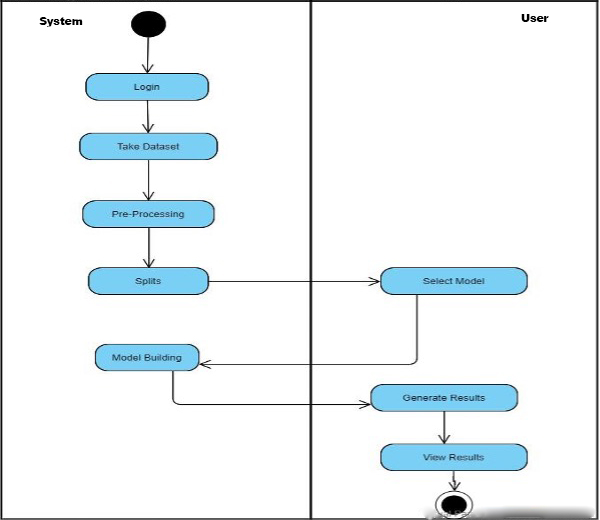


Figure 8 Activity Diagram of the system

## **3.3 CONCLUSION**

The proposed system design demonstrates an innovative approach for identifying software anomalies and bugs. The implementation of the models is well-executed, resulting in minimal loss and optimal efficiency when detecting anomalies.

# **CHAPTER 4**

# **RESULTS AND DISCUSSION**

## **INTRODUCTION**

This section talks about the various performance of the different models that were used during the system design of the project. I also give the visualization data of the different models, and talk about the dataset for a better understanding.

The results were scrutinized to make sure no error in recording the results were made, the models were trained a number of times and results compared.

## **DATASET COLLECTION**

The main information source for crafting the envisioned system is acquired directly from end-user clients who will utilize the software package, serving as the primary wellspring of insights to guide the development of this application.

A comprehensive examination and grasp of the current system are undertaken using either surveys or interviews prior to the creation of the proposed system. Varieties of inputs, processes, and outputs are thoroughly comprehended before the system design phase.

EBPSM, which stands for Enhanced Bug and Project Scenario Modelling, encapsulates a diverse array of bug reports, project scenarios, and associated metadata. These elements are drawn from real-world software development environments, ensuring the dataset's authenticity and relevance to actual industry practices. Each bug report and project scenario are richly annotated with a spectrum of attributes, encompassing details such as bug severity, project complexity, coding standards adherence, and more.

Researchers and practitioners leverage the EBPSM dataset as a valuable asset, leveraging its comprehensive nature to establish a robust foundation for their machine learning endeavours. The dataset's versatility and depth enable the development of predictive models that can forecast software quality issues, recommend optimization strategies, and ultimately elevate the software development process to new levels of efficiency and excellence. As a result, the EBPSM dataset stands as a critical enabler in the quest to engineer automated software quality assurance systems that are intelligent, proactive, and aligned with the ever-evolving demands of modern software engineering.

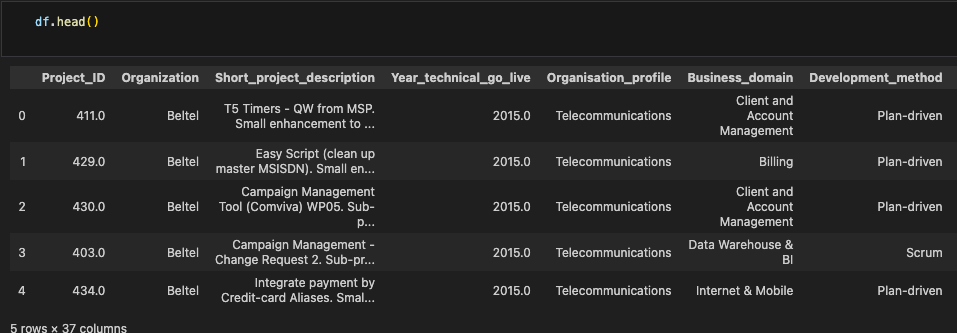


Figure 9 A look at the first 5 rows of the dataset

## **EVALUATION OF THE VARIOUS MACHINE LEARNING ALGORITHMS**

The precision is derived using the formula tp / (tp + fp), where tp is the number of true positives and fp is the number of false positives. Precision refers to the classifier's ability to avoid mislabeling a negative sample as positive. The best value is 1, and the poorest value is 0.

The model recall score reflects how well the system can distinguish false positives from true positives. Precision, on the other hand, measures how many positive predictions a model makes out of all positive predictions.

The F-score, often known as the F1-score, is a measure of a model's accuracy on a particular dataset.

**Random Forest:**

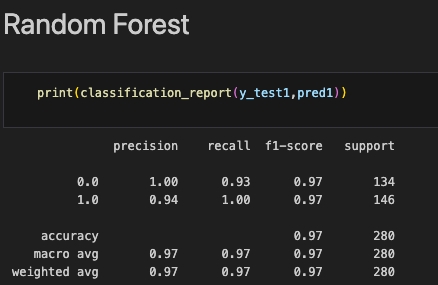
****

Figure 10 Result of the Random Forest Model

**Xgboost:**

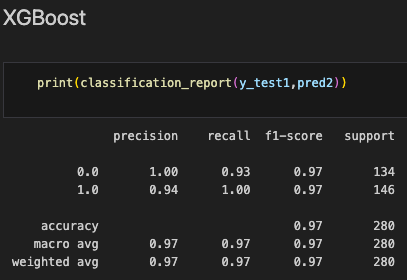
****

Figure 11 Results of the XGboost Model

**Logistic Regression:**



Figure 12 Results of the Logistic Regression Model

**KNN:**

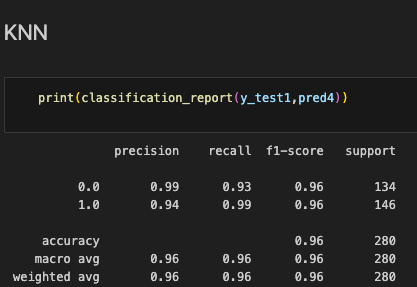


Figure 13 Results of the KNN Model

Table 2 Table of values for the model accuracy

|  |  |  |
| --- | --- | --- |
| **S/n** | **Model** | **Score(Accuracy)** |
| 1 | K-Nearest Neighbor | 97.972973 |
| 2 | Logistic Regression | 97.297297 |
| 3 | Random Forest | 96.621622 |
| 5 | XGBoost | 97.297297 |



Figure 14 Graphical Plot of the Model Accuracy

## **GRAPHICAL USER INTERFACE**

The Model was given a graphical user interface that is user-friendly and interactive. To view the files and access the system for testing, this user requires a fully functional web browser. This can also be viewed on a mobile device without having to resize the page.

The webapp is used to navigate through the project from dataset balancing to model creation, as well as the prediction of outcomes using various applied models.

###### HOME PAGE

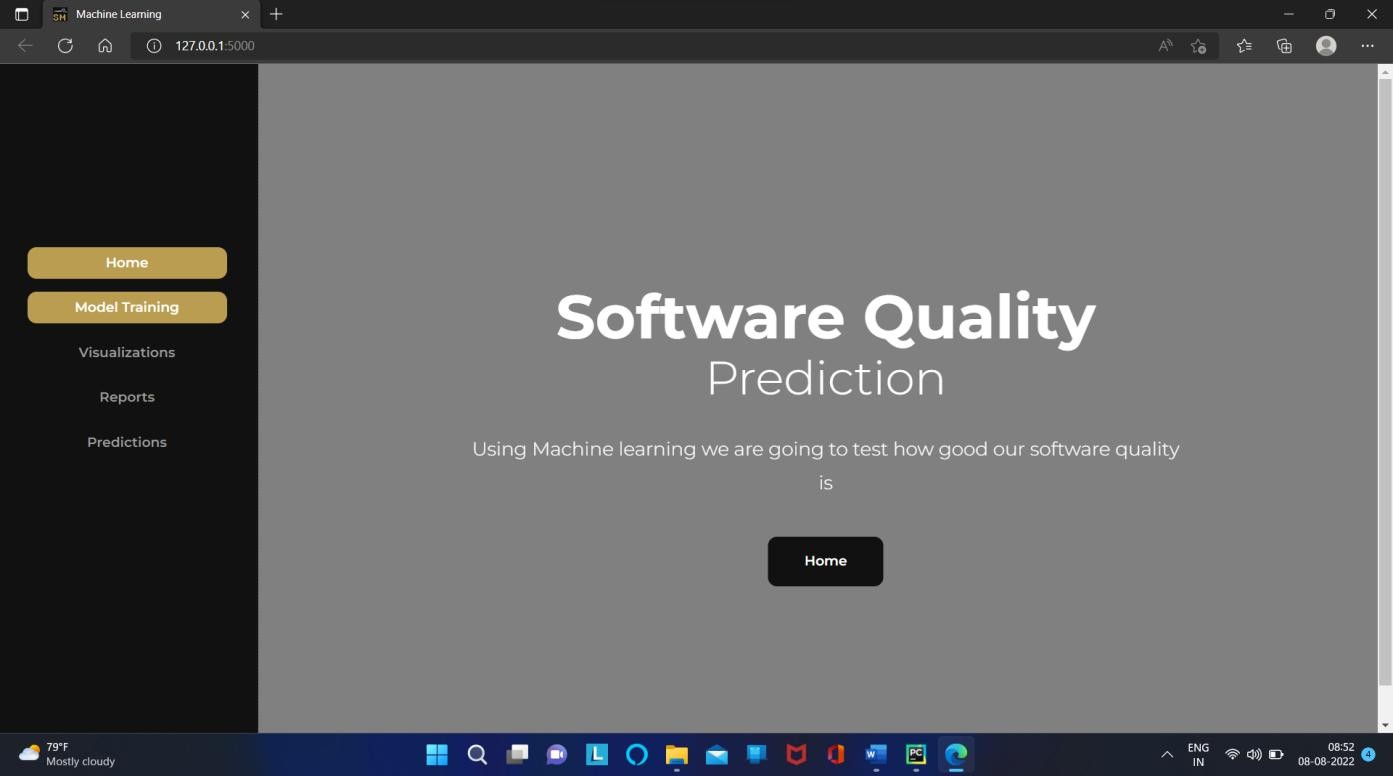


Figure 15 Home Page

###### ACCURACY PAGE

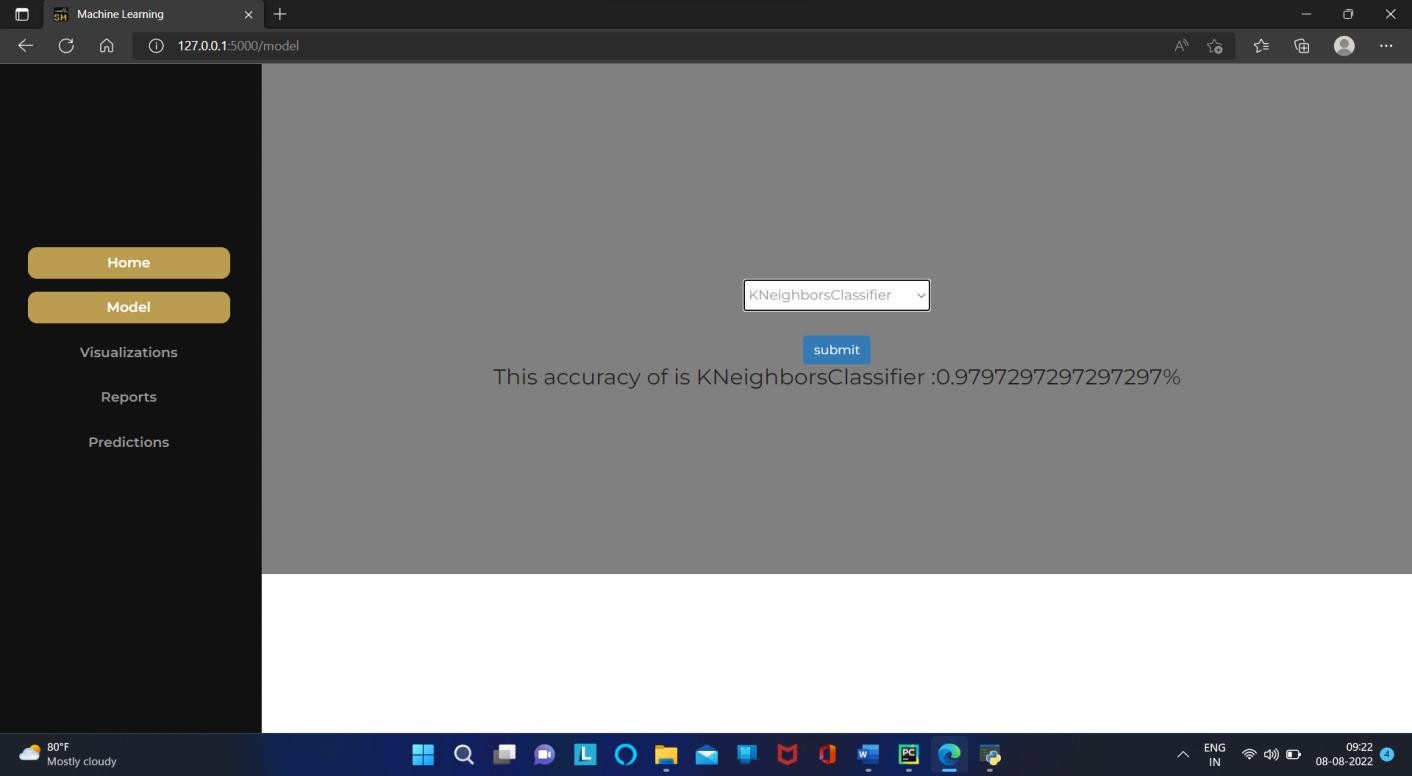


Figure 16 Accuracy Page

###### INPUT FEATURES PAGE

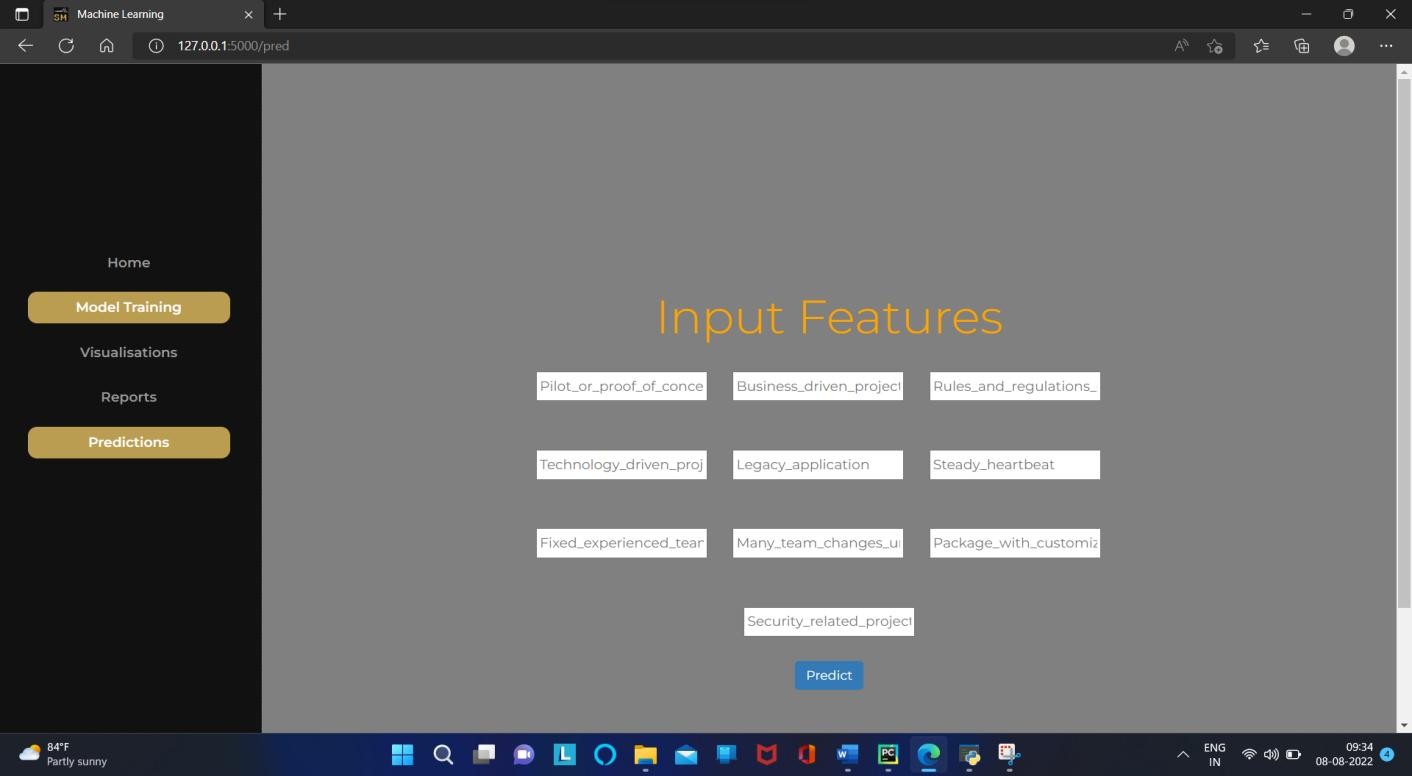


Figure 17 Input Features Page

FINAL OUTPUT PAGE

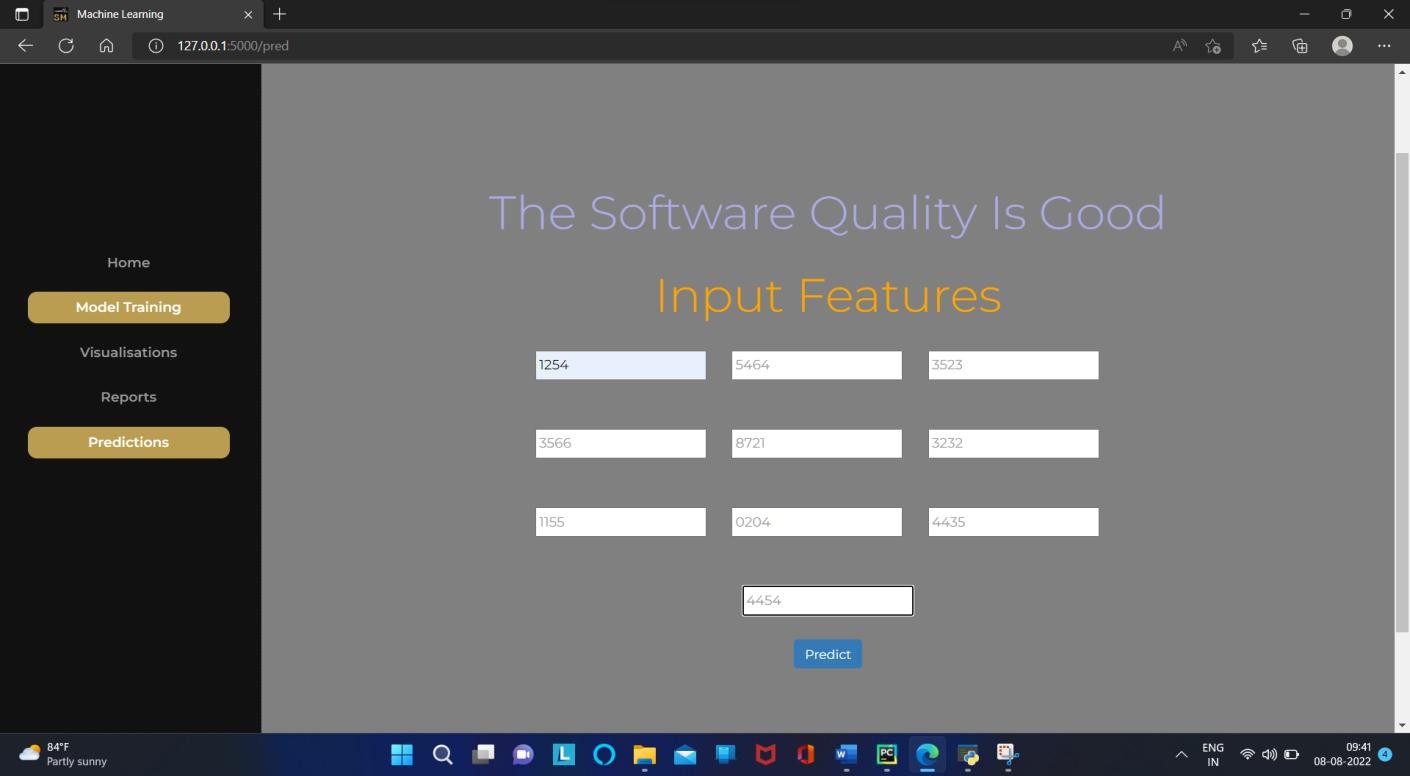


Figure 18 Final Output

## **SYSTEM TESTING**

The system built was tested using the Blackbox testing for the various models and functionalities.

Table 3 Test Case 1

|  |  |
| --- | --- |
| **Tested By:** | Sam David |
| **Test Type** | Black Box Testing |
| **Test Case Number** | 001 |
| **Test Name** | Model Training |
| **Test Description** | Choosing of algorithm, which one gives the best accurate results. |
| **Item(s) to be tested** | |
| 4 | KNN Algorithm, Random Forest, Xgboost and Logistic Regression |
| **Specifications** | |
| Input | Expected Result |
| KNN Algorithm | It Shows accurate results |
| **Procedural Steps** | |
| 1 | Open Home Page |
| 2 | Select Model Training |
| 3 | Enter the Algorithm Name |
| 4 | It Shows Accurate Results |

Table 4 Test Case 2

|  |  |
| --- | --- |
| **Tested By:** | Gbenga Daramola |
| **Test Type** | Black Box Testing |
| **Test Case Number** | 002 |
| **Test Name** | Software Quality Checking |
| **Test Description** | User can give the input Data |
| **Item(s) to be tested** | |
| 1 | Input Values |
| **Specifications** | |
| Input | Expected Result |
| After entering the input Data or Values | Software Quality Checking |
| **Procedural Steps** | |
| 1 | Open prediction |
| 2 | Enter Input Values |
| 3 | It Shows Whether the Software Quality Good or Bad |

# **CHAPTER FIVE**

# **CONCLUSION**

## **Introduction**

This Section talks about the summary, recommendations and conclusions. The various recommendations will be able to help in advancing the automated software quality assurance era.

In the given application, supervised Machine Learning models were utilized to predict software quality. A set of five ML algorithms, namely Random Forest Classifier, Decision Tree Classifier, XG Boost Classifier, and K-Nearest Neighbors (KNN), were employed for this purpose. Notably, all these algorithms exhibited strong performance, achieving favorable levels of accuracy.

## **Summary**

In conclusion, software quality assessment plays a crucial role throughout diverse phases of software development, serving purposes like project quality assurance planning and benchmarking. Previous approaches, encompassing Multiple Criteria Linear Programming, Multiple Criteria Quadratic Programming, C5.0, Support Vector Machine (SVM), and Neural Networks, were explored for quality estimation. To enhance the precision of estimation, this study focused on leveraging pertinent attributes from an extensive dataset. The application of feature selection techniques and correlation matrices was employed to elevate accuracy rates. Furthermore, contemporary methodologies, proven effective in various prediction tasks, were incorporated. Machine learning techniques, including K-Nearest Neighbors (KNN), Random Forest, and Decision Trees, were employed to predict software quality and uncover its relationship with development attributes. Empirical findings affirm that machine learning algorithms effectively facilitate accurate software quality estimation.

## **Achievements**

This research has contributed the following to knowledge;

1. The design, development and implementation of this work will aid other researchers in their works.

2. The successful adaptation of machine learning algorithms.

3. The potential for continuous improvement.

## **Conclusion**

This project entailed the creation of a system that uses machine learning techniques for software quality assurance. This system can provide the majority of the necessary capabilities for detecting software faults. In This system can be further extended to analyze the in depth about the software quality by plotting the graphs of Area under Curve (AUC) vs Receiver Operating characteristic Curve (ROC) and analyzing them for software quality.

**BIBLIOGRAPHY**

Abney, S. (no date) *Understanding the Yarowsky Algorithm*.

Al-Masri, O. and Al-Sorori, W.A. (2022) ‘Object-Oriented Test Case Generation Using Teaching Learning-Based Optimization (TLBO) Algorithm’, *IEEE Access*, 10, pp. 110879–110888. Available at: https://doi.org/10.1109/ACCESS.2022.3214841.

Chen, H. and Hossain, M. (no date) *Application of Machine Learning on Software Quality Assurance and Testing: A Chronological Survey*.

Chillarege, R., Biyani, S. and Rosenthal, J. (no date) *Measurement of Failure Rate in Widely Distributed Software*.

Deepak Nandal, by and Prakash Sangwan, O. (no date) *DESIGN AND EVALUATION OF SOFTWARE EFFORT ESTIMATION TECHNIQUE USING METAHEURISTIC APPROACH DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE & ENGINEERING*.

Felix, E.A. and Lee, S.P. (2017) ‘Integrated Approach to Software Defect Prediction’, *IEEE Access*, 5, pp. 21524–21547. Available at: https://doi.org/10.1109/ACCESS.2017.2759180.

Goedicke, Michael. *et al.* (2012) *Proceedings of the 27th IEEEACM International Conference on Automated Software Engineering.* ACM.

Huang, H.-Z., Zuo, M.J., *et al.* (no date) *The Proceedings of 2012 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering : June 15-18, 2012, Chengdu, China*.

Huang, H.-Z., Liu, Y., *et al.* (no date) *The proceedings of 2013 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering : July 15-18, 2013 : Emeishan, Sichuan, China*.

International Conference on Computer Science and Network Technology 4. 2015 Harbin *et al.* (2015) *Proceedings of 2015 4th International Conference on Computer Science and Network Technology - ICCSNT 2015 December 19-20, 2015, Harbin, China*. IEEE.

Jayanthi, R. and Florence, L. (2019) ‘Software defect prediction techniques using metrics based on neural network classifier’, *Cluster Computing*, 22, pp. 77–88. Available at: https://doi.org/10.1007/s10586-018-1730-1.

Kaur, K. and Chopra, V. (2014) *Review of Automatic Test Case Generation from UML Diagram using Evolutionary Algorithm*, *International Journal of Inventive Engineering and Sciences (IJIES)*.

Khaliq, Z., Farooq, S.U. and Khan, D.A. (2022a) ‘Artificial Intelligence in Software Testing : Impact, Problems, Challenges and Prospect’. Available at: http://arxiv.org/abs/2201.05371.

Khaliq, Z., Farooq, S.U. and Khan, D.A. (2022b) ‘Artificial Intelligence in Software Testing : Impact, Problems, Challenges and Prospect’. Available at: http://arxiv.org/abs/2201.05371.

Lima, R., Rosado Da Cruz, M. and Ribeiro, J. (no date) *Artificial Intelligence Applied to Software Testing: A Literature Review*.

Lyu, M.R. and Nikora, A. (no date) *Software Reliability Measurements through Combination Models: Approaches, Results, and a CASE Tool*.

Murphy, C., Kaiser, G. and Arias, M. (no date) *A Framework for Quality Assurance of Machine Learning Applications*.

Omri, S. and Sinz, C. (2021) ‘Machine Learning Techniques for Software Quality Assurance: A Survey’. Available at: http://arxiv.org/abs/2104.14056.

Panda, M. *et al.* (2020) ‘Test suit generation for object oriented programs: A hybrid firefly and differential evolution approach’, *IEEE Access*, 8, pp. 179167–179188. Available at: https://doi.org/10.1109/ACCESS.2020.3026911.

Panda, M. and Dash, S. (2020) ‘Test-Case Generation for Model-Based Testing of Object-Oriented Programs’, in, pp. 53–77. Available at: https://doi.org/10.1007/978-981-15-2455-4\_3.

Prof. Rumana Anjum and Dr Madhub B, K. (2021) ‘Artificial Intelligence based Software Testing’, *InternationalJournalforResearchinEngineeringApplication&Management(IJREAM)* [Preprint].

Ramayanam, V. and A, S. (2022) ‘SOFTWARE QUALITY PREDICTION BY USING MACHINE LEARNING ALGORITHMS’, *International Research Journal of Modernization in Engineering Technology and Science* [Preprint]. Available at: https://doi.org/10.56726/irjmets31403.

Ratna, P.D., Cheekaty, S. and Kalidasu, H. (no date) *Object Oriented Software Testing*.

Regulapati Akhila, A.Anusha and Dr.P.Srinivasa Rao (2022) ‘AN EXPERIMENTAL STUDY FOR SOFTWARE QUALITY PREDICTION USING MACHINE LEARNING METHODS’. Available at: www.ijrar.org.

SCAD College of Engineering and Technology and Institute of Electrical and Electronics Engineers (no date) *Proceedings of the 4th International Conference on Trends in Electronics and Informatics (ICOEI 2020) : 15-17, June 2020*.

Schneidewind, N.F. and Swss, C. (no date) *Software Metrics Model for Quality Control*.

Seliya, N. and Khoshgoftaar, T.M. (2007) ‘Software quality estimation with limited fault data: a semi-supervised learning perspective’, *Software Quality Journal*, 15(3), pp. 327–344. Available at: https://doi.org/10.1007/s11219-007-9013-8.

Singh, A. *et al.* (2018) ‘OBJECT-ORIENTED COUPLING BASED TEST CASE SELECTION’, *IJRAR1944570 International Journal of Research and Analytical Reviews* [Preprint]. Available at: www.ijrar.org.

Singpurwalla, N.D. and Wilson, S.P. (1994) *Software Reliability Modeling Reliability Modeling*, *International Statistical Review*.

Tripathi, V.Y., Bhadra, V. and Singh, P. (2016) ‘Impact Factor (PIF): 2.243 International Journal OF Engineering Sciences & Management Research OBJECT ORIENTED TEST CASE GENERATION BASED ON GENETIC ALGORITHMS’, *International Journal of Engineering Sciences & Management Research* [Preprint], (1). Available at: http://www.ijesmr.com©.

Trudova, A. and Dolezel, M. (no date) *Artificial Intelligence in Software Test Automation: A Systematic Literature Review*. Available at: http://orcid.org/0000-0002-5963-5145]andAlenaBuchalcevova1[0000-0002-8185-5208].

Vanmali, M., Last, M. and Kandel, A. (no date) *Using a Neural Network in the Software Testing Process*.

Wable, P. *et al.* (2021) ‘Automated Unit Test Cases Generation using Machine Learning’, *International Research Journal of Engineering and Technology* [Preprint]. Available at: https://www.ponicode.com/blog/machine-learning-on-.

Xu, B. *et al.* (no date) *9th International Conference on Reliability, Maintainability and Safety (ICRMS), 2011 12-15 June 2011, Guiyang, China*.

Yarowsky, D. (no date) *UNSUPERVISED WORD SENSE DISAMBIGUATION RIVALING SUPERVISED METHODS*.