# **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) 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. 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.

Against this backdrop, this research endeavour aims to make a substantive contribution to the field by developing a tailored and sophisticated machine learning-based automated quality assurance system for software development. 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.

Drawing upon cutting-edge advancements in machine learning, I will select and tailor a suite of sophisticated machine-learning algorithms. 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.

Ethical considerations related to the developed automated quality assurance system will be accorded the utmost attention and scrutiny. Potential biases 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. With the invaluable guidance and expertise of the esteemed professor, this research project is poised to make a significant and lasting impact in the field of software development.

## **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.

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.

### 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

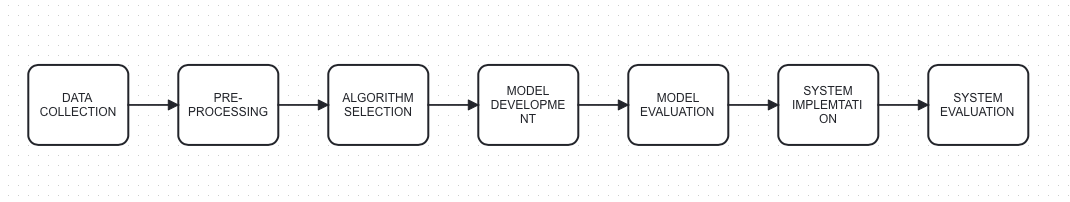


Figure 1 A block diagram showing the methodology of the system

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.

By following this structured sequence, the project report ensures a cohesive and organized presentation of the research, providing researchers with a clear understanding of the historical context, methodology, results, and conclusions of the Automated Quality Assurance system using Machine Learning.

# **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 a higher-quality software by ensuring the absence of failures in the SUT. A failure manifests when the observed behavior of the SUT deviates from the expected behavior outlined in its requirements or other defined descriptions of anticipated behavior (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).

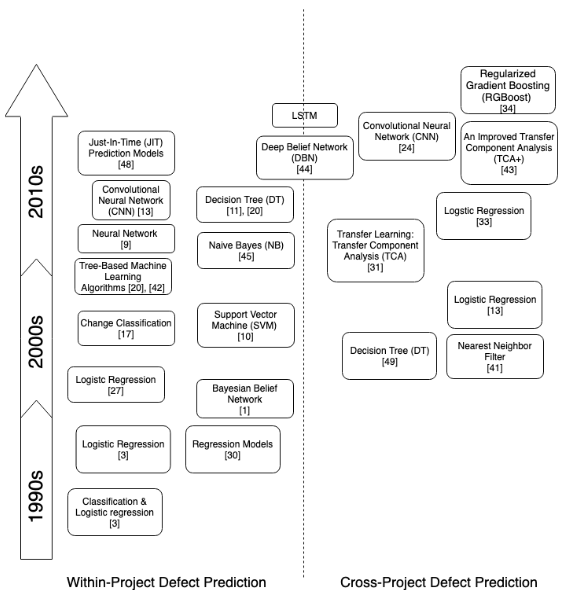


Figure 2 History of Software Defect Prediction

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 in order 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) prior to executing the test case.
2. The test steps entail detailing the specific actions that need to be executed in order to carry out the test case.
3. The expected results, which describe the expected results of the executing test case (Lima, Rosado Da Cruz and Ribeiro, no date).
4. 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 behavior or bugs (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 expected behavior, and documenting observations (Chen and Hossain, no date). 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 behavior 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 basically determine the appropriate inputs to provide and the expected outputs to be generated, while focusing on analyzing the external behavior 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 in order to identify any sections that exhibit undesirable behavior. This method 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 logics 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, shows 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 as 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).

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

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**