AI TESTING

Department of Software Engineering

Term Project

Student: Merve AYDIN

Project Adviser : Assoc. Prof. Dr. Vahide Bulut

January 2024

YAPAY ZEKA İLE TEST

Öz

Yapay zeka (AI), yazılım test süreçlerinde önemli bir rol oynamaktadır. Bu proje, geleneksel test stratejilerine kıyasla daha akıllı, etkili ve ölçeklenebilir test yöntemlerinin geliştirilmesine odaklanmaktadır. AI tabanlı test stratejileri, otomatik test araçları ve makine öğrenimi algoritmalarını içermektedir.

Bu bölümde, temel olarak test otomasyonu ve yapay zeka arasındaki bağlantıyı ele alacağız. Yapay zeka tekniklerinin test süreçlerine olan etkisi ve bu etkinin nasıl optimize edilebileceği üzerine odaklanacağız.

Proje kapsamında gerçekleştirilen AI tabanlı testlerin sonuçları analiz edilecek ve elde edilen bulgular, önceki çalışmalarla karşılaştırılacaktır. Bu bölümde elde edilen değerli veriler, projenin katkılarını vurgulayacaktır.

Bu proje, geleneksel test stratejilerinin ötesine geçerek, yazılım test süreçlerine daha akıllı ve dinamik bir yaklaşım getirmeyi amaçlamaktadır. Yapılan bu çalışma, gelecekteki yazılım test mühendislerine rehberlik etmek ve endüstrideki en iyi uygulamalara katkıda bulunmak için tasarlanmıştır.

Anahtar Kelimeler: Yapay Zeka, Test, Mühendislik, Test süreçleri, Makine Öğrenmesi, Derin Öğrenme

AI TESTING

Abstract

Artificial Intelligence (AI) plays an important role in software testing processes. This project focuses on the development of more intelligent, effective and scalable testing methods compared to traditional testing strategies. AI-based testing strategies include automated testing tools and machine learning algorithms.

In this section, we will mainly discuss the connection between test automation and artificial intelligence. We will focus on the impact of AI techniques on testing processes and how this impact can be optimized.

The results of the AI-based tests performed within the scope of the project will be analyzed and the findings will be compared with previous studies. The valuable data obtained in this section will highlight the contributions of the project.

This project aims to go beyond traditional testing strategies and bring a more intelligent and dynamic approach to software testing processes. This work is designed to guide future software test engineers and contribute to industry best practices.

Keywords: Artificial Intelligence, Testing, Engineering, Testing processes,Machine Learning,Deep Learning

Dedication

I dedicate this thesis to my sibling,Beyza, whose constant encouragement and love have been my source of strength...

Acknowledgement

To all those who contributed to the success achieved in the completion of this project, I dedicate these lines.

To my sibling: Throughout this process, you have always supported me and stood by my side. You have been a source of morale and motivation. This success would not have been possible without you. I am grateful to you.

To my advisor: I thank my advisor for their guidance, suggestions, and support in my project. Their wisdom and guidance played a significant role in enhancing the quality of the project.

To School and Department Authorities: I thank the school and department authorities. The opportunities and support you provided were instrumental in the successful completion of the project.

To Everyone who Provided Support: I thank everyone who supported and shared their ideas and suggestions during this process. Your contributions significantly enriched the project.

This work was realized with your support and contributions. I thank each and every one of you individually.

Table of Contents

Chapter 1

1.1 Introduction to AI

The rapid evolution in software development processes has prompted software test engineers to question traditional testing methods and seek more effective solutions. In this context, attention is drawn to the innovations brought by Artificial Intelligence (AI) to test processes. AI testing offers a more intelligent, dynamic, and scalable approach for software test engineers, carrying the potential to enhance software quality and identify errors more effectively.

Traditional test strategies often follow predefined scenarios and manually oversee a broad test scope. However, the complexity and fast-paced nature of software development processes, which these methods may not adequately address, highlight the prominence of AI-based test strategies. Through its learning capabilities, analytical skills, and adaptability, Artificial Intelligence optimizes test processes and identifies errors through previously undiscovered paths.

AI testing utilizes techniques such as machine learning and data analytics to interpret large amounts of data, recognize patterns, and offer the potential to improve test processes. This approach not only automates test processes but also provides flexibility to continuously update test strategies.

This article will thoroughly examine the concept of AI testing, addressing the advantages, current state, application examples, and future potentials of AI-based test strategies. AI testing aims to bring a new perspective to software test processes, surpassing traditional limitations and aiming to elevate software quality. Exploring the potential impact of this technology on future software development processes is an inevitable necessity in today's dynamic software world.

1.2 AI definition and AI impact

The term Artificial Intelligence (AI) has undergone continuous evolution since the 1950s, originally emerging with the goal of constructing and programming "intelligent" machines. However, in today's context, the definition of Artificial Intelligence has significantly broadened, focusing on the capability of engineering systems to acquire, process, and apply knowledge and skills. This concept now encompasses not only the imitation of humans but also the ability to solve complex problems and learn.

The way people understand Artificial Intelligence has evolved in accordance with technological advancements over time. For instance, in the 1970s, the language processing capabilities of computer systems hinted at the possibility that machines could develop human-like language understanding skills in the future. However, in contemporary times, this has evolved beyond surface-level language processing to a more sophisticated understanding that delves into deep meanings.

As the perception of what qualifies as Artificial Intelligence changes, a phenomenon known as the "AI Effect" emerges. As societal perceptions of AI evolve, so does the definition of Artificial Intelligence. This allows for a better understanding of the impact of technology on human life.

Consequently, predicting how any current definition may evolve over time with technological progress and changing societal expectations is challenging, but it is evident that such evolution is inevitable.

1.3 Narrow, General and Super AI

At a high level, artificial intelligence can be classified into three main categories:

Narrow Artificial Intelligence (also known as weak AI) refers to systems specifically programmed to perform a particular task within a limited context. Such AI systems are widely used in various fields today, including gaming systems, unwanted email filters, test scenario generators, and voice assistants.

General Artificial Intelligence (also known as strong AI) represents systems with general cognitive abilities similar to humans. These AI-based systems can comprehend their surroundings like humans and act accordingly. However, as of 2021, fully realized general artificial intelligence systems have not been developed.

Super Artificial Intelligence systems are defined as those capable of mimicking human cognition and leveraging extensive processing power, almost unlimited memory, and access to all human knowledge (e.g., through web access). It is believed that super AI systems will rapidly surpass human intelligence over time. The point where AI-based systems transition from general AI to super AI is commonly referred to as the "technological singularity."

1.4 AI-Based and Traditional Systems

In a traditional computer system, software is typically programmed using an imperative language, incorporating structures like conditional statements and loops. Understanding how inputs are transformed into outputs by the system is usually straightforward for humans. In an AI-based system, however, the system typically analyzes patterns in data using machine learning and determines how it will respond to future data. For example, an AI-based image processor designed to recognize cat images is trained with a set of images containing cats.

The AI independently identifies patterns or features that can be used to recognize cats. These patterns and rules are then applied to new images to identify contained cats. In many AI-based systems, this process can lead to predictions that are less understandable by humans.

In practice, AI-based systems can be implemented with various technologies, and the "AI Effect" is effective in determining which systems are currently considered AI-based and which ones are considered traditional systems.

1.5 AI Technologies

AI can be implemented using a wide range of technologies. Among these AI can be implemented using a wide range of technologies, such as:

• Fuzzy logic

• Search algorithms

• Reasoning techniques

- Rule engines

- Deductive classifiers

- Case-based reasoning

- Procedural reasoning

• Machine learning techniques

- Neural networks

- Bayesian models

- Decision trees

- Random forest

- Linear regression

- Logistic regression

- Clustering algorithms

- Genetic algorithms

- Support vector machine (SVM)

AI-based systems typically implement one or more of these technologies.

1.6 AI Development Frameworks

There are numerous artificial intelligence development frameworks available, and some of them are tailored to specific domains. These frameworks support a variety of activities, including data preparation, algorithm selection, and the deployment of models onto various processors. Processors may include central processing units (CPUs), graphics processing units (GPUs), or Cloud Tensor Processing Units (TPUs), among others. The choice of a framework depends on factors such as the programming language used and ease of use. Here are some of the popular frameworks as of April 2021:

• Apache MxNet: An open-source deep learning framework used for Amazon Web Services (AWS).

• CNTK: Microsoft Cognitive Toolkit (CNTK), an open-source deep learning tool.

• IBM Watson Studio: A suite of tools supporting the development of AI solutions.

• Keras: A high-level open-source API written in Python, compatible with TensorFlow and CNTK.

• PyTorch: An open-source machine learning library operated by Facebook, used for image processing and natural language processing (NLP) applications.

It provides support for both Python and C++ interfaces.

• Scikit-learn: An open-source machine learning library for the Python programming language.

• TensorFlow: An open-source machine learning framework provided by Google, based on data flow graphs for scalable machine learning.

These development frameworks are continually evolving, sometimes merging, and occasionally being replaced by new frameworks.

1.7 ardware for Artificial Intelligence-Based Systems

Various types of hardware are employed for the training and deployment of machine learning models. For instance, a model performing speech recognition can operate on a low-capacity smartphone but might require access to cloud computing power for training. In cases where the main device is not connected to the internet, a common approach is to train the model in the cloud and then deploy it to the main device.

Hardware supporting machine learning often benefits from the following features:

- Low-precision arithmetic: This involves using fewer bits for computations (e.g., using 8 bits instead of the typical 32 bits for machine learning).
- Capability to work with large data structures, such as supporting matrix multiplication.
- Massive parallel (concurrent) processing ability.

General-purpose CPUs typically support complex operations not often needed in machine learning applications and provide only a few cores. Consequently, their architectures are generally less efficient for training and running machine learning models compared to CPUs with faster clock speeds. As a result, GPUs are often the preferred choice for small-scale machine learning projects.

Some hardware is specifically designed for Artificial Intelligence, including purpose-built Application-Specific Integrated Circuits (ASICs) and System on a Chip (SoC) solutions. These AI-specific solutions come with features like

multiple cores, custom data management, and in-memory processing capabilities. Often, these solutions are most suitable for edge computing while the training of machine learning models is performed in the cloud.

Hardware with specific AI architectures is currently being developed (as of April 2021). This includes neuromorphic processors that mimic brain neurons but do not follow the traditional von Neumann architecture.

Examples of AI hardware providers and processors include (as of April 2021):

- NVIDIA: Offering GPUs like Volta and specialized processors for AI.
- Google: Developing application-specific integrated circuits for training and inference. Google TPUs (Cloud Tensor Processing Units) are accessible to users through Google Cloud, while Edge TPU is a purpose-built ASIC designed to run AI on individual devices.
- Intel: Providing Nervana neural network processors for deep learning and Movidius Myriad image processing units for computer vision and neural network applications.
- Mobileye: Producing the EyeQ SoC device family to support complex and computation-intensive image processing tasks, with low power consumption for use in vehicles.
- Apple: Producing the Bionic chip designed for AI on iPhones.
- Huawei: Having the Kirin 970 chip for smartphones, featuring built-in neural network processing capabilities for AI.

1.8 AI as a Service (AIaaS)

- AI components, such as ML models, can be created within an organization, downloaded from a third party, or used as a service on the web (AIaaS). A hybrid approach is also possible in which some of the AI functionality is provided from within the system and some is provided as a service. When ML is used as a service, access is provided to an ML model over the web and support can also be provided for data preparation and storage, model training, evaluation, tuning, testing, and deployment. Third-party providers (e.g., AWS, Microsoft) offer specific AI services, such as facial and speech recognition.

This allows individuals and organizations to implement AI using cloud-based services even when they have insufficient resources and expertise to build their own AI services. In addition, ML models provided as part of a third-party service are likely to have been trained on a larger, more diverse training dataset than is readily available to many stakeholders, such as those who have recently moved into the AI market.

Chapter 2

2.1 Software Automation Testing and Artificial Intelligence Integration

In software development processes, effective management and optimization of testing stages are continually researched by industry players with the aim of enhancing product quality and achieving time and cost savings. At this point, we can delve into how the integration of software automation testing and artificial intelligence (AI) contributes to these objectives in more detail.

2.2 Increased Speed and Efficiency

Software automation testing provides a faster and more repeatable testing environment compared to manual testing processes. This ensures quicker completion of testing stages in the software development process, leading to the efficient execution of processes.

2.3 Automatic Test Scenarios with AI

Artificial intelligence enables the more effective creation of automatic test scenarios by understanding the structure of the application and user behaviors. AI algorithms can optimize test scenarios by comprehending interactions between different components of the application.

2.4 Identification of Weak Points and Improvement Suggestions

AI can identify weak points in the testing processes and provide improvement suggestions. This accelerates the continuous improvement process, aiding in the prevention of future errors.

2.5 Cost Savings

Software automation testing and AI contribute to cost savings when compared to manual testing processes. The rapid creation and implementation of automatic test scenarios reduce development costs and save time.

2.6 Error Analysis and Discovery

AI enhances the error analysis process by thoroughly analyzing test results. This facilitates faster detection of errors and provides developers with more effective guidance for error correction.

2.7 Performance and Load Testing

AI can optimize performance and load testing by automatically generating complex scenarios. This results in a more comprehensive testing strategy to determine how the application performs under real-world conditions.

2.8 Learning for Future Scenarios:

AI, with its ability to learn from data obtained during test processes, allows for better planning and implementation of future test scenarios. This enables the continuous development of a dynamic test strategy.

In conclusion, the integration of software automation testing and artificial intelligence has the potential to make testing stages in software development more effective, faster, and cost-efficient. This integration represents a significant step for the software industry to increase its competitive advantage and deliver high-quality products.

Chapter 3

3.1 AI models used in software test automation

3.1.1 Machine Learning Models

-Linear Regression: This model is used to model the relationship between a dependent variable and one or more independent variables. For example, it can be used to predict factors influencing the performance of an application.

-Decision Trees: Decision trees are used in classification and regression tasks. For instance, decision trees can be employed to assess the security status of an application.

-Support Vector Machines (SVM): SVM is used in classification and regression tasks. For example, it can be applied to evaluate the user experience of an application.

3.1.2 Deep Learning Models

-Artificial Neural Networks (ANN): ANN is used for understanding complex structured data. For instance, it can be utilized to analyze user interactions in an application and detect errors.

-Convolutional Neural Networks (CNN): CNN is employed in image processing and recognition tasks. For example, it can be used to test the accuracy of graphic elements in the user interface of an application.

-Recurrent Neural Networks (RNN): RNN is used for working with time series and sequential data. For instance, it can be utilized to monitor and understand the performance of an application over time.

3.1.3 Genetic Algorithms

Genetic algorithms are used to evolve test scenarios to achieve optimal performance. For example, genetic algorithms can be applied to generate the most effective test scenarios based on specific usage scenarios of an application.

3.1.4 Natural Language Processing (NLP)

NLP is used to analyze test documents, reports, and error logs to detect issues and perform error analysis. For example, NLP can be employed to identify common problems through error reports of an application.

3.1.5 Clustering Algorithms

Clustering algorithms are used to group test scenarios with similar characteristics and analyze these groups. For instance, clustering algorithms can be applied to group similar errors in different modules of an application.

3.1.6 Autonomous Test Scenario Generation

Artificial intelligence can autonomously generate test scenarios by understanding the dynamic structure of an application and modeling user behaviors. For example, it can automatically record user interactions and create test scenarios based on them.

Chapter 4

4.1 Traditional Software Automation Testing

Traditional software automation testing is an activity that is usually conducted within a specific process and model. Here is an overview describing the traditional software automation testing process and some of the models used:

4.2 Determination of Test Needs

At the beginning of the software development process, business requirements, user scenarios, and software design documentation are thoroughly reviewed. At this stage, the testing team understands the goals of the software and identifies the features that need to be tested.

4.3 Creation of Test Scenarios

Test scenarios are created based on the identified test needs. These scenarios include steps that cover different components and features of the software. Scenarios can be written manually by the testing team or in a test scenario language suitable for automation.

4.4 Creation of the Test Plan

A general test plan is created based on the test scenarios. This plan includes the management of the test process, test stages, test tools to be used, the test environment, test processes, and responsibilities. Additionally, the sequencing and prioritization of test stages are determined at this stage.

4.5 Preparation of the Test Environment

An appropriate test environment is prepared to successfully execute the test scenarios. This environment may include different configurations, platforms, and databases of the software. If necessary, test data is prepared and loaded at this stage.

4.6 Execution of Test Scenarios

The prepared test scenarios are executed according to the established test plan. At this stage, it is ensured that the test scenarios run successfully and produce the expected results. The test process is typically organized to cover different modules or components of the software.

4.7 Evaluation of Test Results

The results obtained after the execution of test scenarios are thoroughly evaluated. Errors, deficiencies, and performance issues are identified. Relevant teams are informed about these errors, and corrective actions are taken.

4.8 Reporting and Monitoring

Test results are shared by creating a test report. This report includes the achievements, errors, and improvement suggestions of the test process. If necessary, the test process is updated, and lessons are learned for future test activities.

This process is crucial for achieving the goals set in the software development process and enhancing the quality of the software. Additionally, based on a specific software development model (such as the Waterfall model or V-model), these processes can be applied in a more specific manner.

Chapter 5

5.1 Traditional Software Automation Testing Models

5.1.1 Waterfall Model

- The Waterfall model addresses the software development process in a sequential and phased manner. Each stage builds upon the previous one, and a phase must be completed before moving on to the next.
- Testing stages typically come towards the end of the software development process. During this stage, numerous test scenarios are planned and executed.

5.1.2 V-Model

- The V-Model aligns each development stage with a corresponding testing stage. After each development step, a testing stage is executed.
- In this model, testing stages wait for the completion of development steps. Relevant testing stages are used to verify the accuracy of development steps and to detect errors early.

5.1.3 Iterative Model

- In the Iterative model, the software development process is divided into small and repeatable iterations. Each iteration is developed based on the previous one.
- Testing stages are applied at the end of each iteration. This allows the confirmation of changes made in each iteration and the early detection of errors.

5.1.4 Dual Vee Model (Inverted V-Model)

- In this model, a structure similar to the V-Model is used, but testing stages occur before development steps.

- This allows for the early detection and resolution of errors, enabling a more reliable and rapid development process.

5.1.5 Agile Model

- The Agile model embraces a flexible and continuous software development process. Small, independent, and frequent deliveries are made.
- In the Agile model, testing occurs in every iteration of the software and is frequently performed. Continuous integration and automation testing play a significant role in the Agile model.

5.1.6 Spiral Model

- The Spiral model involves repeating the software development process in cycles (spiral). Each spiral has a stage.
- In each cycle, a stage begins with risk analysis and prototyping. Testing stages are applied at the end of each cycle.

These models represent different approaches in traditional software development and testing processes. Each model has its advantages and disadvantages, and the choice of which model to use depends on the project's characteristics and requirements.

Chapter 6

6.1 Test levels on AI Based system

AI-based systems typically encompass both components involving artificial intelligence and those without it. While conventional approaches can be used to test components without AI, those with AI components and systems containing them should undergo a distinct testing process for various reasons outlined below. For all test levels involving AI components, close support from data engineers/scientists and domain experts is crucial.

A notable difference from the test levels used for traditional software lies in the inclusion of two new specialized test levels explicitly addressing the testing of input data and the models used in AI-based systems. This section is largely applicable to all AI-based systems, with some parts specifically focusing on machine learning (ML).

6.2 Input Data Testing

The main objective of input data testing is to ensure that the data used by the system for training and prediction is of the highest quality. It includes the following:

Reviews

- Statistical techniques (e.g., testing data for bias)

-Exploratory Data Analysis (EDA) of the training data

-Static and dynamic testing of the data pipeline

The data pipeline typically consists of several components performing data preparation tasks. The testing of these components involves both component testing and integration testing. The data pipeline used for training may differ significantly from the fully engineered, automated version used for operational predictions.

Therefore, it is crucial to test both versions of the data pipeline. However, testing the functional equivalence of these two versions should also be considered.

6.3 ML Model Testing

The primary objective of ML model testing is to ensure the existence of the selected model meeting the specified performance criteria. This includes the following: • ML functional performance criteria • ML non-functional acceptance criteria appropriate for the ML model in isolation, such as training speed, prediction speed, utilized computational resources, adaptability, and transparency.

ML model testing also aims to determine that the ML framework, algorithm, model, model settings, and hyperparameters are chosen as optimally as possible. If applicable, ML model testing may also encompass testing to meet white-box coverage criteria. The selected model is subsequently integrated with other components, both AI and non-AI.

6.4 Component Testing

It is a conventional test level that can be applied to any non-model components, such as user interfaces and communication components.

6.5 Component Integration Testing

It is a traditional test level conducted to validate the interaction of system components (both containing AI and non-AI elements). This test verifies whether inputs from the data pipeline are received as expected by the model and ensures that predictions generated by the model are accurately received and utilized by relevant system components, such as the user interface. When AI is offered as a service, API testing of the provided service is commonly performed as part of Component Integration Testing.

6.6 System Testing

System testing is a traditional testing level conducted to verify whether integrated components (both AI and non-AI) perform as expected as a whole. This test is carried out in a test environment closely representative of the operational environment, considering both functional and non-functional aspects. System testing may involve field trials or test scenarios to evaluate the system's operation, especially in situations that are hazardous or challenging to replicate in the operational environment. During system testing, the ML functional performance criteria are re-evaluated to ensure that the initial results from ML model testing are not adversely affected when the model is fully integrated into the system. This testing is particularly important if intentional changes have been made to the AI component (e.g., compressing a DNN to reduce its size). System testing is also a test level used to evaluate a variety of non-functional requirements for the system. For example, adversarial tests may be conducted for reliability, and the system can be tested for explainability. If applicable, interfaces with hardware components (e.g., sensors) may also be tested as part of system testing.

6.7 Acceptance Testing

Acceptance Testing is a conventional test level conducted to determine whether a completed system is acceptable to the customer. Defining acceptance criteria for AI-based systems can be challenging. If AI is provided as a service, acceptance testing may be required to assess the suitability of the service for the intended system and whether, for example, ML functional performance criteria have been adequately met.

Chapter 7

7.1 Comparison of AI and traditional models on positive aspects

7.1.1 Traditional Model

Ease of Setup: Traditional test models are generally easy to set up and rely on previously used testing tools. Therefore, they may require less effort to get started.

Focus on Specific Scenarios: Traditional models focus on executing test scenarios designed and optimized for specific situations. They usually concentrate on a particular expected behavior.

Combination of Manual and Automated Testing: Traditional approaches can combine manual and automated testing processes. This provides flexibility, especially when dealing with user interface tests and handling special cases.

7.1.2 Comparison with Artificial Intelligence (AI) Model

Efficiency and Speed: Artificial intelligence can create test scenarios faster and more effectively by analyzing large datasets. It can enhance efficiency, especially in large and complex software systems.

Learning Capability: AI models have the ability to learn from data over time. This provides valuable insights for understanding the complexity of software and improving future test scenarios.

Automatic Bug Discovery: AI can often automatically detect errors during testing. Its data analysis and pattern recognition capabilities allow it to find errors faster and more effectively than traditional methods.

Coverage of Diverse Test Scenarios: AI may have a broader test coverage compared to traditional models. It can go beyond specific user scenarios and address unexpected situations.

Self-Adaptability: AI can adapt to changes in software and automatically update test scenarios. This is a significant advantage in continuous integration and continuous delivery (CI/CD) processes.

Natural Language Processing (NLP): AI, through its NLP capabilities, can analyze test documents, reports, and error logs. This aids in quickly detecting and analyzing errors.

7.1.3 Conclusion

The choice between traditional and AI-based test models should be made based on the project's needs and requirements. Both approaches have their advantages and strengths, and the one most suitable for the project's requirements is selected. For example, if dealing with large datasets, complex user behaviors, and fast delivery requirements, AI-based test models might be more appealing.

Chapter 8

8.1 Comparison of AI and traditional models on negative aspects

8.1.1 Traditional Model

Flexibility and Adaptation Challenges: Traditional test models often face challenges in quickly adapting to changing software requirements. Flexibility shortcomings are particularly evident in projects with significant changes.

Manual Workload: Manual test processes still hold a significant place in traditional models. This situation can result in time and effort requirements for repetitive manual testing processes.

Challenges in Comprehensive Data Analysis: Working with large datasets and conducting complex data analyses can be more challenging for traditional models. This situation can adversely affect comprehensive error analysis and discovery processes.

Lack of Speed and Efficiency: Traditional models generally require longer test processes. This can be limiting for projects requiring rapid delivery or continuous integration processes.

8.1.2 AI Model:

Training and Algorithmic Complexity: AI-based test models may involve complex algorithms and training processes. This complexity may require expertise to use these models effectively.

Need for Accurate Training Data: AI models require accurate and representative training data. If training data is missing or incorrect, it can negatively impact the model's performance.

Cost: AI-based test processes can be more expensive due to training, development, and maintenance costs. This can pose a challenge for small-scale projects or organizations with limited budgets.

Ethical and Moral Issues: The use of AI can raise ethical and moral concerns, including the selection of accurate training data, prevention of biases, and interpretation of results.

Complexity: AI models tend to be complex, making them less understandable for users or test teams. Additional effort may be required to obtain simplified and understandable results.

Chapter 9

9.1 The impact of AI on test automation in the future

Utilizing artificial intelligence and machine learning is crucial for effective automation. Automating testing processes can provide a faster, more economical, and efficient alternative compared to traditional methods. Allowing AI to handle the scenario-writing process for quality engineers enables the creation of more optimized and robust test scenarios.

According to experts, in the future, many systems and applications will be tested by AI and ML, with the API world playing a pioneering role in this transformation. This shift is expected to facilitate the easier and more effective execution of API tests.

In the manual testing domain, some organizations are still hesitant about embracing AI due to the belief that humans still surpass machines in creativity, exploration, and analytical capabilities. Therefore, while manual tests continue in specific areas, the likelihood of increased implementation of artificial intelligence applications in other areas is high.

In the field of test automation, the establishment of AI-based systems allows organizations to conduct more tests at a lower cost. The resources saved can be redirected towards exploratory testing, contributing to investments in QA innovations. This creates an opportunity for testers to focus on more interesting and valuable tasks, ultimately enhancing software quality.

With the emergence of AI, there is a belief that testers need not worry about the obsolescence of their roles but should instead consider a different perspective on the future of testing. Striking a balance between humans and machines will make test processes more efficient and contribute to the development of high-quality products. Collaboration with artificial intelligence in the testing world is deemed key to success, and testers stand to be among the winners when this collaboration is effectively leveraged.

References

Graham, D., & Van Veenendaal, E. (2007). "Foundations of Software Testing."

Li, K. (2004). "Effective Software Test Automation."

Arbon, J., & Carollo, J. (2021). "AI for Software Testing."

Graham, D., & Fewster, M. (2005). "Experiences of Test Automation: Case Studies of Software Test Automation."

ISTQB Certified Tester AI Testing (CT-AI) Syllabus Version 1.0.

Testeryou - Test Dünyasında Yapay Zekanın Yeri https://testeryou.com/tr/test-dunyasinda-yapay-zekanin-yeri/