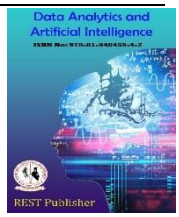
DOI: https://doi.org/10.46632/daai/2/6/4

Data Analytics and Artificial Intelligence

Vol: 2(6), 2022

REST Publisher; ISBN: 978-81-948459-4-2

Website: http://restpublisher.com/book-series/daai/



Future of Software Testing Using Artificial Intelligence

Gaurav Ghanbahadur, *C Kalpana,

S.S.T college of arts and Commerce, Maharashtra, Mumbai, India

*Corresponding Author's mail id: rkalpz@gmail.com

Abstract: *As we know, After the invention of Artificial intelligence, most work has become easy. Artificial Intelligence's short name is 'AI'. So AI is Making impact in many fields for example Medical, Industries, Domestic, Laws, Arts, Defence & etc. Because of this AI is capable of performing many roles like, managing smart factories, creating weather forecasts, detecting disease, Personal assistant and latest autonomous vehicles, etc. Software testing is nothing but abnormal behavior of the software, as per software specification or client's requirements. Software testing is very responsible. It is tedious, laborious and the most time-consuming process. Automation tools have been designed to assist in automating some aspects of the testing process in order to improve quality and timeliness. However, automated tools are becoming less effective in the (CI CD) continuous development and continuous delivery pipeline. So for that Tester decided to use AI to solve this problem. AI is not only finding bugs and errors but also doing work fastly as compared to humans. In this research paper our main agenda is how AI has an impact on Software testing life cycle and Testing activities. So in this research paper identify the biggest challenges for Tester and How the AI is a contribution in Testing life cycle.*

Keyword: *Artificial Intelligence, Machine Learning, Deep Learning, Software Testing, Software Testing Activities*

1. INTRODUCTION

So we all know that Artificial intelligence applications are already involved in many fields, whereas previously expected to be only the domain of human experts. AI tools have already advanced in fields such as finance, law, medicine, and even the arts. In many fields, AI has surpassed human intelligence and is approaching human creativity and empathy. AI's spectacular victories in chess, Ludo king games, and other games are examples. Artificial intelligence is slowly altering the landscape of software engineering in general, and software testing in particular, in both research and industry. AI has been discovered to have had a significant impact on how we approach software testing. Since most organizations have turned to automation testing to bridge the gap between the increasing complexity of deliverable software and the contraction of the delivery cycle, the gap has grown alarmingly. Artificial Intelligence is gradually changing the landscape of software engineering in general and software testing in particular both in research and industry as well.AI has been found to have made a considerable impact on the way we are approaching software testing. Since most of the organizations have turned to automation testing to bridge the gap that exists between the growing complexity of deliverable software and the contraction of the delivery cycle yet the gap is stretching at an alarming pace bringing us closer to a tip-ping point wherein test automation too will fail for us to deliver quality software on time. AI will help to reduce time in testing and fill that gap that's of growing complexity of the delivery cycle why most of the organizations are turned to automation testing. automation tool will fair to the tester to deliver quality software on time. There by saving a significant amount of time and effort. Our focus in this research is to identify software testing life cycle and testing activities, where AI has made impact and greatly enhanced the process within each activity. We also identify AI techniques that have been mostly applied to the process of software testing activities. we convey the problems to identified by the study that the tester is facing while implementing AI-based solutions to the testing problems. We also provide some key areas where AI can potentially help the testing community.

2. OVERVIEW OF ARTIFICIAL INTELLIGENCE

Artificial intelligence this time was coined by John McCarthy in 1955 at a conference organized by the Dartmouth Conference. The term was used to refer to all" programming systems in which the machine is simulating some intelligent human behavior". Accorded by John McCarthy, the science and engineering is making intelligent machines, specially intelligent computer programs. In this study we discuss the main branches of AI that have been mostly applied to software testing life cycle.

Artificial Neural Networks: Artificial neural networks are created by basing artificial intelligence on biological neural networks (ANNs). ANNs are connections of nodes that resemble the connections between neurons in biological brain networks. Three essential elements are present in all neural networks: node properties, network structure, and learning rules. The node's character controls how it manages signals. The arrangement and connectivity of nodes are governed by a network topology. Using a weight adjustment technique, a learning rule automatically decides how to initializing and modify weights. This kind of network develops into a computing system that can learn via practice and enhance its functionality.

AI Planning: AI Planning: AI planning research can be traced back to the 1960s logic theorists' program created by Newell and Simon. The task of AI planning is to find a set of effective actions in a given planning domain that can satisfactorily transform the initial state of the planning problem to the target state after the actions are applied**.**

Robotics: Robotics is a branch of artificial intelligence that combines electrical engineering, mechanical engineering, and computer science to design, manufacture, and apply robots. A physically positioned intelligent agent with five main components: text effectors, perception, control, communication, and power is referred to as an intelligent robot. Effectors are robot peripherals that assist the robot in moving and interacting with its surroundings. Perception is a set of sensors that allows the robot to perceive its surroundings. Controls function similarly to the central nervous system, allowing computations that allow robotic systems to maximize their chances of success. Communication refers to how robots and humans interact with one another through language, gestures, and proxemics.

Machine learning: Machine learning is a computational technique that uses prior experience to improve performance or make accurate predictions. In this context, experience refers to information available to the learner in the past, which is typically collected in the form of electronic data and made available for analysis. This information could take the form of human-labeled digitalized training sets or other types of information garnered from interaction with the environment.

Natural Language Processing (NLP): The term "natural language processing" (NLP) describes AI techniques that allow intelligent computers to converse with one another using natural language, such as English. When you want conversation-based clinical expert systems to make choices or when you want intelligent systems to follow your directions, for instance, natural language processing is required. Fuzzy Logic (FL) is a type of reasoning that resembles human thought.

The method used by FL imitates human decision-making and takes into account all third-party outcomes between the digital values YES and NO. The foundation of FL is the notion that there is no clear demarcation between the two extremes. Using a set of rules that work together to create a result, FL is a way of deductive reasoning.

Expert Systems: Expert Systems are computer programmers made to perform at very high levels of human intellect and experience while solving complicated problems in a particular field. One can list the common characteristics of expert systems.

• Programming languages formalize the rules that characterize a specific problem as computer procedures.

• An inference engine that analyses and processes scenarios; a knowledge base that stores issues and solutions to assist in decision-making in the form of a computerized database. The user is fully informed of any problems and the solutions developed.

To put it simply, the system functions like a powerful, intelligent "computer brain."

Software Testing Overview:

A software testing study is one that is conducted to provide stakeholders with information about the quality of a software product or system that is being tested (SUT). Testing consumes 30% to 40% of a software development organization's total project effort, and it accounts for more than 50% of the total project cost. If the SUT is error-free, the software's quality will improve. An error is detected when the SUT's external behavior differs from what is expected of it based on its requirements or another description of the expected behavior. The test case is a crucial component of a test activity. Typically, test cases are derived from functional specifications, design specifications, or requirements specifications Generally, test cases are derived from functional specifications, design specifications, or requirements specifications. The test case specification includes the following items:

• Preconditions that describe the environment and state of the SUT prior to the execution of test cases.

• Test Steps, which describe the actions that must be taken in order to carry out the Test Case.

• Expected outcomes

Describes the expected outcome of the currently running test case.

• Actual outcomes describing test case execution outcomes
The test is explored and executed along several dimensions, and these dimensions define the test's validity criteria. Criteria defining what constitutes an appropriate test
A number of such criteria have,
Test Types: There are two main types of tests: In this type of testing, a tester assumes the role of an end-user and tests software to identify unexpected behavior or bugs.
Automated Testing: A form of software testing that uses software tools to run predefined tests. Software tools used for automated testing are often referred to as test automation tools or test automation frameworks. This relieves testers from the burden of running test cases, but the process of planning test cases and writing them in the form of test scripts must be done manually.
Manual Testing: In manual testing, testers execute test cases manually without the use of tools or scripts. In this type of testing, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug.
Test Methodology: Three main test methodologies have been identified.
Black-box testing: Black-box testing, also known as functional testing, aims to examine the external behavior of software without dealing with its inner workings.
Black box testing is based on software inputs and outputs.
White-box testing*:* On the other hand, white-box testing, also known as structural testing, creates test cases based on the SUT implementation.
Its purpose is to ensure that all constructs of the SUT (paths, statements, branches, etc.) are executed during test suite execution.
• Grey box Testing: Grey box testing is a testing technique for testing a software product or application using partial knowledge of the internals of the application

3. TESTING PHASES OR TESTING LEVELS

It is carried out at all stages of the software development lifecycle, such as development, release, and production. Unit tests are run during development to test basic software units such as methods and classes. Following unit testing, code is frequently changed as a result of changing software requirements, enhancements, or maintenance work, which can introduce errors into the code and result in obvious failures. To address this, all test levels include a technique known as regression testing. Because his SUT is tested every time a change is implemented, regression testing is the most time-consuming and tedious testing technique. Pre-release requirements testing ensures that the SUT performs all functions in accordance with the software requirements specification. Prior to release, scenario testing is performed by creating scenarios for the SUT, testing the SUT against those scenarios, and looking for unintended behavior of the SUT. Performance testing is a type of testing that evaluates an SUT's speed, ability to respond, and stability under load. In production, alpha testing is performed in a development environment, with a developer acting as a user of her SUT to find bugs. In this testing method, the developer examines her SUT from the user's point of view. The SUT is tested in a user environment during beta testing.The user is interacting with his SUIT here, while the developer is simply observing her SUT and analysing the errors.

4. IMPACT OF AI ON SOFTWARE TESTING

The application of AI techniques in the Software Testing Life Cycle characterizes areas where AI techniques have proven useful in software testing research and practice (STLC). AI techniques have made a strong impression at all stages of the STLC, from planning to reporting. We identified testing activities or testing facets where significant research was done using AI to investigate the impact of AI on software testing. The majority of the STLC has been covered by these test activities. Specifications for the test: Test cases are created at the start of the software testing lifecycle based on the software's characteristics and requirements. To ensure that all software requirements are tested, test cases are written in checklists that include test specifications. Specific test objectives are included, as are required inputs and expected results, step-by-step instructions for performing the test, and pass/fail criteria for acceptance decisions. We will discuss two landmark papers that apply AI to this activity below. Last and Friedman demonstrated how Info-Fuzzy Networks (IFN) can be used to automatically derive functional requirements from running data. When testing new versions of possible bugs in the system, derivative models of the tested software are used to recover missing incomplete specifications, design a minimal set of regression tests, and reduce software errors. We assessed the output's precision. proposes a methodology that takes a test suite (a set of test cases) and a test specification developed using a category partitioning (CP) strategy as input. Test cases are transformed into abstract test cases based on the CP specification. Instead of raw input/output, this is a tuple of (category, choice) pairs associated with the output equivalence class. Then, to model input properties and output equivalence classes, rules that associate pairs (category, choice) are learned. These rules are then analyzed to identify

potential test suite improvements (e.g., redundant test cases, need for additional test cases) and CP specification improvements (e.g adding more categories or choices).
It has to be).

Test Case Refinement: Test case refinement is a deliberate activity performed by testers in order to select the most effective test cases for execution and thus reduce test costs. Two of her AI technologies were identified as being used in this testing activity. Last and Kendell and Last Invented Info-Fuzzy Networks (IFN). Already in use. We have published a new method for automatically reducing black-box combinatorial testing that is based on the automatic identification of input-output relationships from test programmer execution data. Singh. describes a method for generating test cases Partition checking with the Z specification. As input, the learner is given Z's functional specification**.** As output, the approach produces a classification tree describing high-level test cases. The high-level test cases are then further refined by creating their disjunctive normal form.

Test Case Generation: It is the tester's responsibility to create a test suite that meets the test validity criteria after enveloping the test validity criteria. Because manual test set creation is an unmanageable task for complex applications, most testers use techniques to automatically generate test cases. Over the last two decades, there has been a significant increase in interest in using the KI to automate test case generation, and the KI has had a significant impact on this testing effort. Previous research in this area dates back to 1996, when the author used inductive learning methods to generate test cases from a limited set of input-output examples. Given a programmer P and a set of alternative programmers P', the proposed method generates good test cases that distinguish P from all programmers in P'. Present an active learning framework for black box software testing. Active learning uses input/output pairs from a black box to build a model of the model to represent. Then, using this model, generate a new sampling input. An Ant Colony Optimization approach for automatic test sequence generation for state-based software testing is proposed by Li, H., and Lam, C. To achieve the required test coverage, the proposed approach directly uses UML artefacts to generate test sequences. Start creating a method for creating models for web applications based on logged user data. Their method generates statistical models of user sessions automatically and generates test cases from these models. provides an automated method for creating functional conformance tests for Semantic Web Services. The Input, Output, Precondition, Effect (IOPE) paradigm was used to define web service semantics. Their technology generates test cases that can be run through a GUI or by directly calling web services. To avoid his combinatorial explosion problem occurring in the AI- scheduler, he employs a modified AI scheduler. They used this method to generate GUI test cases. The main idea was to generate the planner's first test case and then propose a solution extension method to improve the planner's performance. This paper proposes a method for combining the genetic algorithm (GA) with tab search techniques. An experimental study they conducted revealed that a combination of methods was effective, with test cases generated using the GA methods individually. The main observation is that taboo search prevents the proposed technique from becoming entangled in her local minima. Srivastava and Baby present an algorithm that generates optimal and minimal test sequences for software behavioral specification using ant colony optimization techniques. This paper describes a method for creating test sequences and obtaining complete software coverage. Several test case generation techniques based on memetic algorithms were presented. It is distinct from GA in that it employs a hill-climbing search to find individual local optima in each generation. Ant colony optimization is used by the system to generate tests that affect the state of the GUI. Sadler and Cohen broaden the concept of goal-based interface testing to generate tests for a variety of objectives. They create the Event Flow Slicer, a direct test generation technique that is 92% faster than those used in human performance regression testing and saves time. Specification for the purpose of extracting test cases for testing. By using automated planning to obtain test suites to test common vulnerabilities, they contribute to the use of AI for web application security testing. The planning system generates test cases as a series of actions that progress from an initial to a final state.

Test Data Generation*:* A software testing activity or process that generates test inputs and data based on logical test cases and test scenarios is known as test data generation. The test coverage of the SUT is determined by the quality of the test data. Jones began preliminary work to apply AI to this activity. We used GA to generate the test set by searching the test data's input domain, ensuring that each branch of the code was executed. A systematic mapping study on the use of GA techniques to test data-generating activities was provided. The results demonstrated that the genetic algorithm could successfully generate simple test data but not complex test data such as images, videos, audio, and 3D models (three-dimensional models). The learner is given a series of actions extracted from the app's GUI as input. The output can be viewed as a model of the application's user interface. presented a tool for generating test data for programmer evaluation from a corpus of sample tests as initial input The corpus is clustered, and RNNs are used to learn generative models that can generate new test data using sequences. Propose a memetic algorithm for optimizing ant colonies for the generation of structural test data. To improve the local train ant search function, we employ the evolution strategy. Proposes a machine learning method as a metaheuristic approximation to model programmer behavior that is difficult to test with traditional approaches, overcoming current state-of-the-art limitations. when the path-exploding problem occurs.

5. PROBLEMS AND CHALLENGES OF AI IN SOFTWARE TESTING

Given the lack of industry expertise and research activity, this section outlines some open problems and challenges in applying AI to software testing.

Test Oracle's Biggest Challenge: The test oracle problem affects every software testing researcher and his practitioner colleague. It's been around since the beginning of the software testing puzzle, and it could last much longer, if not forever. Despite repeated attempts to mitigate the test oracle issue, the researcher was able to solve the issue for a static subset of her SUIT. Previous test oracles derived against the SUT begin to lose validity as soon as the SUT's dynamic properties emerge. In many cases, the requirements document lacks a document from which test oracles are generated. Effective Testing Without Documents AI technology was used to address an oracle's dynamic dream. These AI techniques have taken a lot of work.

Data availability: Every software testing researcher and practitioner fellow faces the test oracle problem. It's been around since the beginning of the software testing puzzle and could last much longer, if not forever. Despite repeated attempts to mitigate the test oracle problem, the researcher was able to solve the problem for a static subset of her SUIT. Previous test oracles derived against the SUT begin to lose their validity as soon as the SUT's dynamic properties emerge. In many cases, the requirements document does not include a document from which test oracles are generated. Effective Testing Without Documents AI technology was used to address this dynamic oracle dream.

Adaptability to data: AI models are heavily reliant on the data on which they are trained and tested. Collecting a robust dataset from real-world scenarios and using that data to train a generalized model that fits that data is a critical step in creating an AI model. Such models are based on the belief that future data and historical data (the data used to train the model) are drawn from the same distribution. However, this is not always the case. Because most data show significant differences over time. Customers discovered that their shopping habits change with the seasons. The difficult task is determining the best time to recalibrate and automating the recalibration process. The challenging task, however, is for the to recognize the ideal time to recalibrate and automate the recalibration process.

Test Data Identification: Before being put into production, all AI models must be thoroughly 9 tested. Testing a model is similar to a black-box method in that no structural or logical information about the model is required. Requires a fair amount of knowledge and understanding of test data. Again, selecting test data from the same distribution can result in a biased model. The issue is with test set coverage. H. "Will they be tested with a larger data distribution?" you might ask. In the realm of software testing, locating such a coverage-based test dataset is a difficult task.

Identifying Test Data: Before going into production, every AI model must be thoroughly tested. Model testing is a black-box technique in which structural or logical information about the model is not required. Comprehensive knowledge and understanding of the testing data is required. Again, selecting testing data from the same distribution can lead to bias in the model. The issue is with test set coverage, or asking the question "Is the model tested over a larger distribution of data?" Identifying such coverage-based test datasets is a difficult task in the field of software testing.

Exhaustive search space leads to loss of generality: Most optimization problems in search-based software testing necessitate an exhaustive search for solutions or goals by AI algorithms. Suboptimal search strategies have been identified and implemented in the past, but they only work for certain types of problems.

Exploitation of Multi Core Computation: Many AI techniques are computationally expensive, which makes them potentially incompatible with large-scale problems encountered by software testers. Graphical Processing Units (GPU) and Tensor Processing Units (TPU) have been incorporated at scale for these techniques due to recent advancements in computing infrastructure. respects for AI in Software Testing Many businesses have begun to invest in AI-assisted software testing techniques in recent years. These artificial intelligence systems provide an alternative to traditional testing methods. AI systems are still in their infancy, but the potential benefits are too great to overlook. Here are some excerpts from research and software testing industry experts that we believe will help software testers in the future. AI systems can reliably perform time-consuming routine tasks. This allows software testers to spend more time troubleshooting the most difficult problems. • Simulated Testing - The ability to programmer artificial intelligence systems and test application code is extremely useful. It offers a realistic simulation of situations that a software tester might encounter. This improves the test's accuracy because all possible scenarios can be identified and reproduced. • To create self-healing systems, the next generation of artificial intelligence in software testing includes self-correcting tools that can instantly identify and fix vulnerabilities without requiring human intervention. • Using artificial intelligence in software testing can help software companies and testers save money. This is already taking place. We believe it is normal for organizations and other user groups to use AI to automate the testing process, while testers concentrate on system exploratory testing. • Predictive AI analytics are critical in identifying all possible test cases and making software products more robust, reliable, and exceeding customer expectations. • AI is expected to perform all tasks in the STLC that require human intelligence, from planning to execution to reporting.

6. CONCLUSION

The software testing community's desire for AI is evidenced by the rapid growth of interest in topics where AI has been applied to software testing over the last two decades. This is the result of AI providing an efficient solution to a long-standing problem in his testing community. AI is already being embraced as a promising solution to many issues confronting testers worldwide. We investigated the effect of his KI on all phases of STLC in this paper. We identified seven software testing activities that benefit the most from AI techniques. GA, Reinforcement Learning, and ANN are some of the most widely used AI techniques. We identified the issues and difficulties that researchers and testers face when applying his AI techniques to software testing. We also discussed how AI could shape the software testing space in the future.

REFERANCE

[1]. Pressman, R. S."Software engineering: A practitioner's approach," New York: McGraw-Hill. 1987.

[2]. Ramler, R., & Wolfmaier, K,"Economic perspectives in test automation: balancing automated and manual testing with opportunity cost. In Proceedings of the 2006 international workshop on Automation of software test", (pp. 85-91). ACM, 2006, May.

[3]. P. Ammann and J. Offutt,"Introduction to Software Testing, 2nd ed,". Cam- bridge, U.K.: Cambridge Univ. Press, 2016.997.

[4]. Vinicius H. S. Durelli , Rafael S. Durelli , Simone S. Borges, Andre T. Endo, Marcelo M. Eler , Diego R. C. Dias , and Marcelo P. Guimaraes, "Machine Learning Applied to Software Testing: A Systematic Mapping Study,". IEEE TRANSACTIONS ON REALIABILITY. 2019.

[5]. C. Koch, "How the computer beat the Go master," Scientific American, vol. 19, 2016.

[6]. F.-H. Hsu, Behind Deep Blue: Building the Computer that Defeated the World Chess Champion. Princeton, NJ: Princeton Univ. Press, 2004

[7]. Brown, T. B. Preprint at https://arxiv.org/abs/2005.14165 (2020).

[8]. Choi, W., Necula, G., & Sen, K. (2013). Guided GUI testing of android apps with minimal restart and approximate learning. OOPSLA 2013, 623–640.

[9]. Paduraru, C. and Marius-Constantin Melemciuc. "An Automatic Test Data Generation Tool using Machine Learning." ICSOFT (2018).

[10]. Sharifipour, H., Shakeri, M., & Haghighi, H. (2018). Structural test data generation using a memetic ant colony optimization based on evolution strategies. Swarm Evol. Comput. 40, 76–91.

[11]. Jan Cegin," Machine learning based test data generation safety-critical software, Proceedings of the 28th ACT Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering November 2020 Pages 1678–1681https://doi.org/10.1145/3368089.3418538

[12]. Liu, P., Zhang, X., Pistoia, M., Zheng, Y., Marques, M., & Zeng, L. (2017). Automatic Text Input Generation for Mobile Testing. ICSE 2017, 643–653.