JOURNAL OF BASIC SCIENCE AND ENGINEERING

A NEW APPROACH OF SOFTWARE TEST AUTOMATION USING AI

Submitted By

Md Fokrul Islam Khan
M.Sc. in Management Information System
Email :fokrulkhan837@gmail.com
ORCID ID: https://orcid.org/0009-0001-8174-4831

Farhad Uddin Mahmud
Master's in Management Information System
Email:mahmudfarhad89@gmail.com
ORCID ID:https://orcid.org/0009-0008-4243-5818

Arif Hoseen
Master's in Business Analytics
Email: arifhossen4295@gmail.com
ORCID ID: https://orcid.org/0009-0005-4042-8611

Supervised By

Abdul Kadar Muhammad Masum,Ph.D.
Professor ,Dept. of Software Engineering
Daffodil International University,Bangladesh

International American University, Los Angeles, California

ABSTRACT

This essential aim was to give an outline of computerized reasoning's part in software test automation. Man-made consciousness (simulated intelligence) altogether affects software designing, and software testing is no exemption. The aim of software test automation might be nearer than ever utilizing man-made consciousness (computer-based intelligence). Throughout recent years, there has been a slight change in the perspective. From manual testing to robotized testing — where Selenium is recognized as one of the most astonishing test automation instruments — everything about the testing framework has been a wonderful encounter. Software testing ought to in this way consider state of the art testing procedures that depend on careful examination in the ongoing high-speed IT climate. For this reason, the development of artificial intelligence-based testing has demonstrated extremely accommodating. AI and computer-based intelligence calculations might have the option to completely recreate a PC's capacity to learn without the requirement for human mediation (ML). While man-made intelligence and ML

require the production of novel and creative calculations to get to information and advantage from it by distinguishing cases to earn barely enough to get by, these expectations are planned to be completely utilized in software testing.

Keywords:*Software Test Automation, Artificial Intelligence AI, Machine Learning, Software Engineering*

1. INTRODUCTION

Since software testing is fundamental to guaranteeing the trustworthiness, security, quality, and execution of software systems, it assumes a significant part in software designing. Fashioners can recognize and fix any defects or software flaws by administering testing. It centers around by and large handiness and ensures the program fulfils client prerequisites and assumptions. Since man-made reasoning is a wide field, the focal point of this survey is on the subfield of software testing, explicitly ML and DL processes. Software testing is presently confronting various difficulties. The intricacy of software structures increments, making it progressively testing to test each situation genuinely. Moreover, the execution of customary test automation procedures is arduous and unpredictable. What's more, since it requires fast testing, keeping awake until late with facilitated improvement is likewise a test. Artificial intelligence might have the option to assist with these issues by giving refined and powerful testing instruments.

This audit's goal is to distinguish arising patterns and the present status of the field of software testing automation with man-made consciousness. This study looks at and assesses the numerous strategies, methods, and devices utilized in this field. The audit's force comes from the forthcoming advantages of man-made brainpower (computer-based intelligence) in software testing, which could propel the cutting-edge software testing strategies. Simulated intelligence might have the option to further develop testing processes and computerize the testing framework. Software testing might be more open, attainable, and useful with computer-based intelligence. Furthermore, computer-based intelligence can assist with the deficiency of skilled testers. Also, it might help stay aware of the fast headway patterns of shrewd improvement systems. Man-made intelligence can possibly address a couple of software testing difficulties. A portion of these issues incorporate composing test cases manually, improving on tests, examining test results, etc.

Development in our expert and individual lives is continually advancing at an exceptionally high speed, and we ought to be aware of this. All features of life are presently being influenced by the computerized uprising, from expanded reality headgear to home machines. Associations give applications to an overall crowd that are used by thousands, on the off chance that not millions, of individuals. Most of those organizations utilize the speedy and deft conveyance technique, which results in new farewells consequently. To give the most ideal experience to the last client, these activities should be completely tested prior to being conveyed. That is excessively quick for manual testing to stay aware of. Each firm, paying little mind to measure, sees application

and software testing as a central period of the pattern of consistent improvement. A program has a few significant parts that are endorsed by this cycle. Without a doubt, when software creates and new elements are offered, manual testing turns out to be more costly, tedious, and inefficient. To forestall such worries, test automation is progressively being integrated. Test automation motorizes essential cycles and errands thoroughly to support the quality and adequacy of human testers.

Computerized reasoning (man-made intelligence) in software development is still in its earliest stages. Its degree of autonomy is still far lower than in other created businesses, for example, self-driving systems or voice-enacted control, yet it is as yet moving toward free testing. Man-made brainpower is being used in software testing gadgets to improve on the software development lifecycle for the group dealing with the undertaking. Man-made brainpower (artificial intelligence) can possibly motorize and lessen how much redundant and exhausting assignments that should be done genuinely in software development and testing. Prior to adding any products, we ought to affirm that the test is constantly run with an empty vehicle. This dodges incorrect disclosures and sticks to solid automation prerequisites. The biggest issue with test automation is "support". To stay aware of the rising intricacy of software, we ought to compose more tests. We are overburdened with testing and support accordingly. Finding and examining tests that bomb requires some investment and exertion. Late examination uncovers that testers should give some time in keeping up with their tests.

2. LITERATURE REVIEW

Dai et al. (2018)offer a ground-breaking strategy for dealing with self-recuperating in disseminated automation systems that utilizes issue tree examination (FTA) and cloud processing. The creators perceive the intricacy and significance of adaptation to internal failure in these sorts of systems, and they propose a cloud-based choice support framework (DSS) to help self-mending techniques. Through the combination of FTA with the DSS framework, they empower extensive issue investigation and recognizable proof of hidden drivers, thus actuating computerized response apparatuses. The survey centers around the leaders' meaning of proactive issue the executives and how cloud-based designs could fortify systems.

Myers et al. (2015)"The Art of Software Testing" keeps on being an essential writing in software designing, giving broad bits of knowledge into software testing techniques and systems. The book, presently in its third release, covers an abundance of information amassed over numerous long periods of contribution in the field. It gives specialists fundamental direction for contriving effective testing approaches by covering all parts of software testing, going from imaginative methodology to basic guidelines. The accentuation on deliberate testing draws near, for example, balance allocating, limit regard examination, and state progress testing, which are vital in recognizing plausible imperfections from the get-go in the development lifecycle, is especially appropriate to blame ID.

Huang et al. (2018)by their preliminary examination of covering show constructors, upgrade the exact comprehension of imperfection distinguishing proof abilities. Combinatorial plans known as covering groups are in many cases utilized in software testing to guarantee satisfactory test consideration. The creators assess the reasonableness of a few covering show constructors for tracking down blunders in software systems in their survey. Through the heading of broad investigations and verifiable requests, they give significant bits of knowledge into the attributes and restrictions of various development calculations. Their discoveries not just explain the decision of suitable testing methodologies yet additionally give knowledge into how to work on systems to expand the proficiency of shortcoming recognition.

Singh and Sharma (2015)give a complete outline of electrical automation testing devices determined to help experts and researchers in choosing the best instruments for their testing prerequisites. The overview covers a few points of view like as highlights, capacities, similarity, ease of use, and possibility of an extensive variety of mechanized testing devices explicitly intended for online applications. The creators give important direction on improving testing efficiency and reasonability for web development projects by consolidating pieces of information from different mechanical assemblies.

Narayan (2018)looks at the developing job of man-made brainpower (computer-based intelligence) in software designing and testing, giving light on the progressive prospects of computer-based intelligence driven strategies for upgrading the exactness, efficiency, and flexibility of testing. Using goodies of data from AI, normal language handling, and other man-made consciousness strategies, the paper talks about how artificial intelligence might computerize numerous parts of software testing, like test age, execution, and investigation. Affiliations might speed up the software development lifecycle while holding elevated degrees of significant worth and constancy by using computer-based intelligence to tackle difficulties like test case focusing on, backslide testing, and quirk area.

3. SOFTWARE TESTING

One of the main periods of the software development life cycle is software testing, which is likewise a vital method for evaluating the completed item and guarantee quality. Prior to presenting the eventual outcome, it is significant to guarantee that the prerequisites recognized in the examination stage have been met and the thing is sans deformity. In software development, a bug is a mistake that demonstrates the framework isn't creating the ideal outcomes as per the necessities. Testing affects the method involved with creating software. Different cycles can be applied to the software development process to work with testing. One method is prototyping, which includes making an exploratory software model that will be disposed of on assessment. Old software depends on the iterative procedure as opposed to being disposed of. Utilizing structures like the Dynamic Systems Development Method (DSDM), which records best practices for iterative and steady development, is another methodology.

3.1. Categories of Testing

Software engineers must to know about and approach an assortment of elective testing procedures, ideas, and strategies. Fig. 1 sums up the four unmistakable testing characterizations:

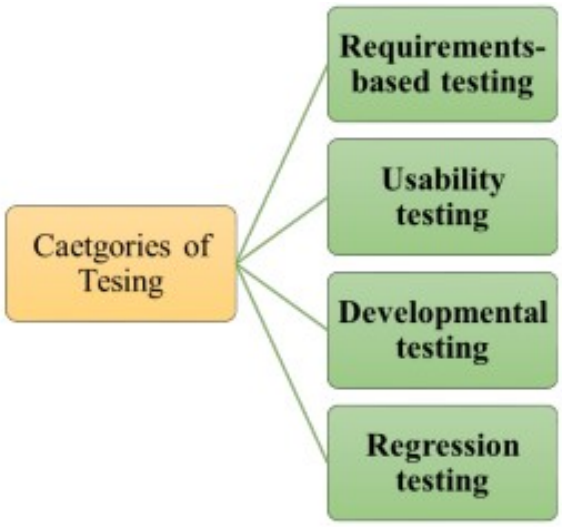

Figure 1: Testing Categories.

3.2. Software Testing Techniques

Test case creation strategies are an important component of approval and verification. White-box testing and discovery testing are the two key tactics. Black box testing's primary goal is to ensure that an execution accurately satisfies all of the client's requirements. The test cases are prepared by examining the system's determination. The decision considers the specifics of the system's anticipated capabilities.

Dim Another technique is box testing, where the application is tested with full admittance to the outside parts of the framework and restricted admittance to within activities. Confirming that the points of interest of a framework execution are right is a significant goal of white box testing. To guarantee that the framework plays out its expected obligations precisely, the test cases are planned by taking a gander at the points of interest of how the framework is being executed.

The five strategies utilized in white-box testing and the four methodologies utilized in disclosure testing are displayed in Fig. 2.

Vol. 21, No. 1, (2024)
ISSN: 1005-0930

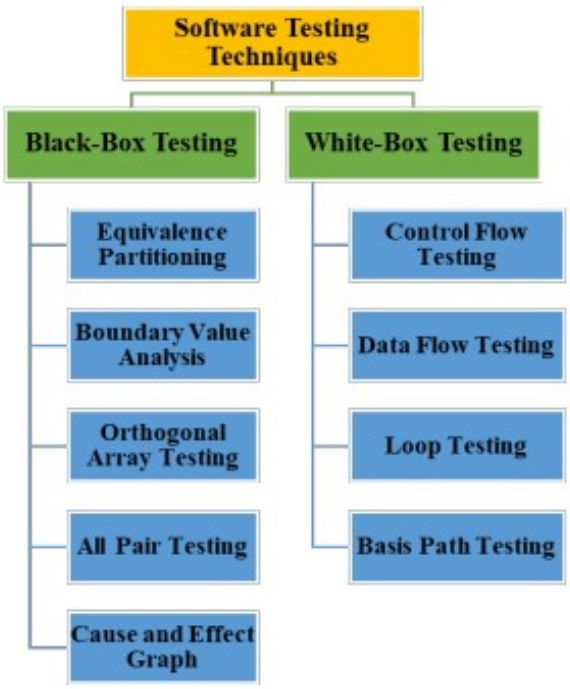

Figure 2: Techniques for White-Box and Black-Box Testing.

4. APPLICATION OF AI IN SOFTWARE TESTING

Modern apps need various attributes to satisfy their down to earth and non-utilitarian prerequisites because of their developing intricacy and list of capabilities. Data in regards to the qualities and elements of a framework that should be known before development begins is known as a need. The prerequisites will be carried out in the software framework once the client and the designer have recognized and endorsed them. The framework fashioner's assignment is to change over the prerequisites into an application by utilizing suitable instruments and techniques. The application should be made mistake free, and the originator should get test cases and execute them written down while the application is being coded. The individual responsible for hauling an application through inferno and guaranteeing that everything works out as expected is known as a software tester. Because of their significance in affirming the item's quality, software testers assume a similarly significant part in software designing as creators do. The development group gets a report from the software tester that rundowns the issues that were all found as well as the succession of occasions that added to the blunder. The Creator's liability regarding composing code and making and running test cases is displayed in Fig. 3. The testers will complete the testing position and stick to the test designs.

Vol. 21, No. 1, (2024)
ISSN: 1005-0930

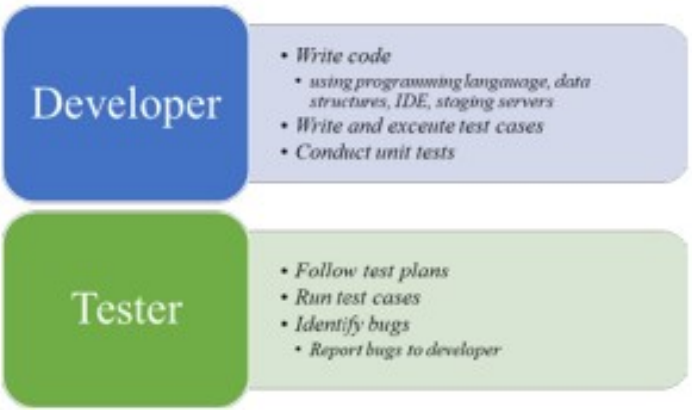

Figure 3: Activities: Testing Software vs. Developing Software.

Software testing is a significant cycle that guarantees the clients' fulfilment with the cultivated framework. Testing shields the framework from any glitches that could affect affiliations' exhibition during the functional stage. In manual testing, the software tester runs the tests on the program to search for mistakes in the framework. Albeit exploratory testing should be possible physically, it is a difficult cycle since HR is involved and test cases are completed by software and human testers. Motorized testing utilizes automation devices to run test cases and is essentially quicker than manual testing as well as a manual technique. Software testing involves simulated intelligence techniques as testing shifts increasingly more toward automation. Viable automation of the testing processes is supported by man-made intelligence techniques. Artificial intelligence systems are assisting with working on the utility of the testing climate. The man-made intelligence calculation works with the cycle and helps software testers in tracking down the greatest number of flaws in the shortest measure of time. The method might be exhibited as exact and trustworthy. The parts of a designer and a tester in manual and man-made intelligence fuelled testing settings are portrayed in Figs. 4 and 5.

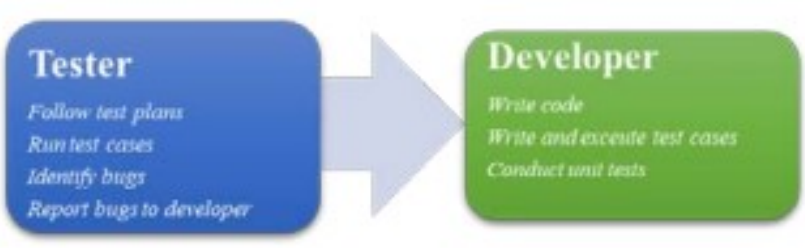

Figure 4: Operates in a Manual Testing Environment.

Vol. 21, No. 1, (2024)
ISSN: 1005-0930

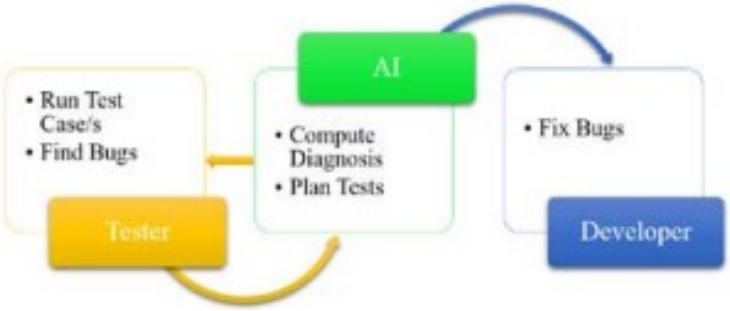

Figure 5: Activities in a Testing Environment Enabled by AI.

While testing physically, the tester follows the test plans, executes the test cases, tracks down the imperfections, and informs the architect of them. The tester executes test cases and finds blemishes in the simulated intelligence-based testing climate. The simulated intelligence framework does the investigation and informs the architect to resolve the issues. Subsequently, the computer-based intelligence device makes further tests, and testers keep on running test cases and recognize blunders.

5. TEST MODELING FOR AI FUNCTIONS

To empower quality testing, simulated intelligence software testing, as traditional software testing, additionally needs to characterize test necessities and test models.

We request that students utilize an alternate portrayal model, called a three-layered request choice table (3D Gathering Table), while dealing with student testing projects in CMPE287 class. This model will assist them with driving gathering based test need examination and show for a few erratic convenient applications that are fuelled by computer-based intelligence capacities in object acknowledgment, game plan, and assumption. Our discoveries demonstrate that this model furnishes understudies with a bit by bit, methodical way to deal with fostering their computer-based intelligence capability test models.

The following progressions make up the essential test exhibiting approach for each chosen man-made intelligence ability:

- Step #1: AI function context classification modelling –To show the request discoveries utilizing a setting characterisation model, in some cases called a setting plan tree, a tester should initially recognize and bunch different defining conditions and limits.

GCT= (NCT, ECT, RCT) is a 3-tuple that addresses a setting gathering tree, where NCT is a restricted nonempty assortment of hubs with a hub name. In NCT, there are three distinct sorts of hubs: leaf hubs (NL), halfway hubs (NI), and root hubs (NR). A rudimentary model is displayed in Figure 6. The ECT is comprised of a few edges, every one of which joins two hubs in the tree (GCT) and determines different order semantic connections between them. As its constituent

Vol. 21, No. 1, (2024)
ISSN: 1005-0930

parts, these semantic relations are reviewed for RCT. Semantic relations come in four assortments: AND, XOR, SELET-1, and SELECTM. The definite portrayals are shown in the table underneath.

Table 1: Semantic Relations in Node-Parent (NP) Structures

Semantic Relation	**Description**
AND	- When all of its child nodes need to be included, NP and its n child nodes have an AND relation.
XOR	- If just one of NP's two child nodes could be chosen, then NP and its two child nodes would have an XOR relation.
SELECT-1	- When just one of NP's child nodes is eligible for selection, the relationship between NP and its child nodes is SELECT-1.
SELECT-M	- If and only if m of n child nodes might be chosen, then NP and its n child nodes have a SELECT-M relation.

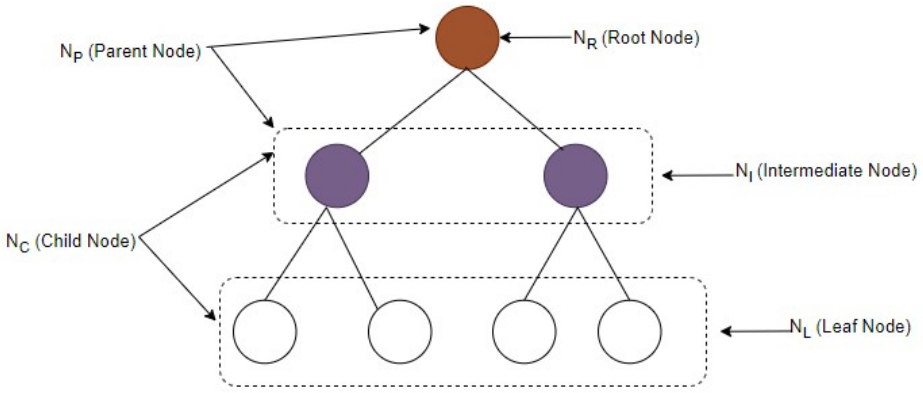

Figure 6: An example tree with notations for the nodes.

- Step #2: Man-made intelligence capacity input request appearing - to perceive and bunch different snippets of data as per their order classes and their sub-classes, a tester should zero in on input portrayal in this step. At the point when an artificial intelligence-based limit perceives different data media designs (like text, sound, picture, and video), every one ought to be inspected and organized. We utilized a data portrayal model (likewise called a data gathering tree) as our examination and test model to manage and deal with different data classes and their

Vol. 21, No. 1, (2024)
ISSN: 1005-0930

sub-class utilizing a class approach to effectively support a tester to lead input request. An illustration of a data game plan tree made for Home Kit is displayed in Figure 7.

- Step #3: Grouping and showing AI capability results: In this stage, a tester focuses on characterizing various AI capacity yields, such as texts, sounds, videos, and images, or events (or activities). This stage results in the creation of a result order tree model, similar to info arrangement.

- Step #4: Go with a 3D portrayal choice table. In this step, a tester makes a 3D request choice table, which distinguishes three layered mappings between disjoint gathered inputs, disjoint gathered setting conditions, and disjoint arranged yields. This is finished for each under-test man-made intelligence ability feature.

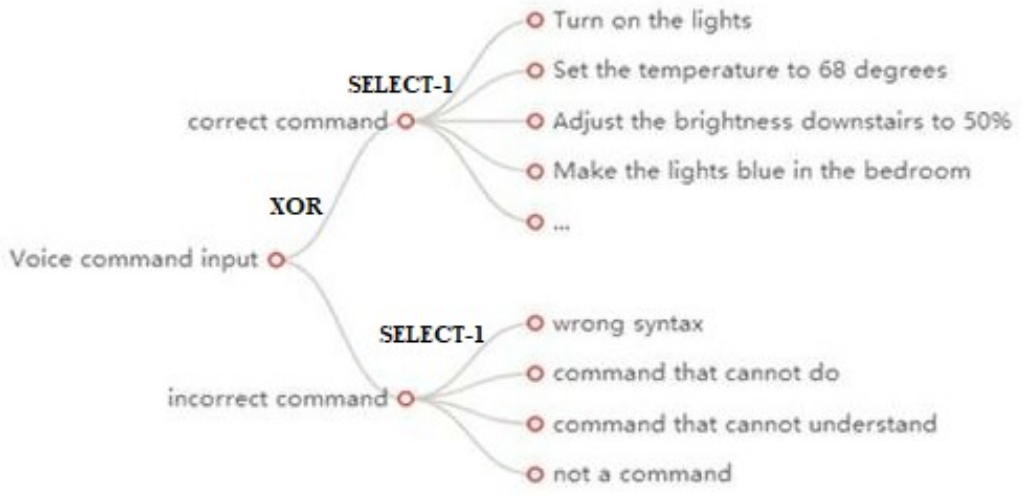

Figure 7: The Voice-Command Function Context Classification Tree Model.

Ordinary decision tables have been utilized in the past as a viable testing technique to approve a framework's business leads and prompt terrible ways of behaving. Normal decision tables handle confounded business rules under various situations. Every decision table thinks about action stubs and action entries and incorporates segments and condition nails. A decision table's segments each act as norms for concluding the circumstances under which the activities recorded in the action stub will happen. A decision table is a methodical methodology that catches different data blends and the related framework ways of behaving (or yields) in an even design. It is likewise often alluded to as an explanation influence table since it catches circumstances and end results for further developed test consideration. Decision tables are able to do precisely portraying muddled issues in every single imaginable circumstance, being succinct, and not missing anything. A whole test case course of action can be arranged utilizing the decision table.

The concept of regular decision tables influences the suggested 3D order decision table for any AI capacity. Its main objective is to use a three-layered plain view to illustrate any given AI

characterization capability. For each AI system capacity, there are three perspectives included in each 3D characterisation decision table:

a) AI function context classification view,where the connected disjoint condition choice is handled as clear setting order rules, and distinct different setting conditions are recorded as setting hits.

b) AI function input classification view,when distinct sources of information are identified and recorded as characterized input hits, and linked disjoint conclusions are handled as specific information characterisation rules.

c) AI function outcome classification view,where the recognized unmistakable man-made intelligence capacity yields (occasions or results) are reported as grouped yield nails, and the connected disjoint decisions are taken care of as particular outcome plan rules.

6. CONCLUSION

This study gives points of view on the endorsement of artificial intelligence software, including test focuses and includes, educational activity ideas, endorsement association, and test exhibiting for assessing artificial intelligence characterisation capacities. Also, the vital classifications of artificial intelligence software testing and current methods for endorsement are checked on and examined. Besides, the text additionally examines. At last, the crucial difficulties, issues, and prerequisites are introduced. It takes a great deal of work to coordinate every one of the necessities and give excellent software to the clients in the distributed period. This will not be simple assuming the yearly testing approach is utilized. Software testing should be possible with an assortment of automation gadgets. Man-made consciousness processes fundamentally affect software testing as well as different phases of the software development life cycle. A fitting answer for software testing activity to deliver a disfigurement free software program is the utilization of computer-based intelligence in test automation.

REFERENCES

1. *Aleem, L. F. Capretz, F. Ahmed, "Benchmarking Machine Learning Techniques for Software Defect Detection", Int. J. Softw. Eng. Appl., vol. 6, no. 3, pp. 11-23, 2015.*
2. *C. Jordan, F. Maurer, S. Lowenberg and J. Provost, "Framework for Flexible, Adaptive Support of Test Management by Means of Software Agents", IEEE Robotics and Automation Letters, vol. 4, no. 3, pp. 2754-2761, 2019.*
3. *Dai, Wenbin, et al. "A Cloud-Based Decision Support System for Self-Healing in Distributed Automation Systems Using Fault Tree Analysis." IEEE Transactions on Industrial Informatics 14.3 (2018): 989-1000.*
4. *Glenford J. Myers, Corey Sandler, Tom Badgett, The Art of Software Testing, 3rd Edition, 2015.*

5. *Huang, Rubing, et al. "Poster: An Experimental Analysis of Fault Detection Capabilities of Covering Array Constructors." 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). IEEE, 2018.*
6. *Jagdish Singh, Monika Sharma." A Comprehensive Review of Webbased Automation Testing Tools". International Journal of Innovative Research in Computer and Communication Engineering, Vol. 3(10), October 2015.*
7. *Mariani, L., Hao, D., Subramanyan, R., Zhu, H.: The central role of test automation in software quality assurance. Software Quality Journal 25(3), 797–802 (2017). doi:10.1007/s11219-017-9383-5.*
8. *Narayan, Vaibhav, The Role of AI in Software Engineering and Testing (June 22, 2018). International Journal of Technical Research and Applications, 2018, Available at SSRN: https://ssrn.com/abstract= 3633525.*
9. *P. Kumudha, R. Venkatesan, Cost-Sensitive Radial Basis Function Neural Network Classifier for Software Defect Prediction, vol. 2016, 2016.*
10. *R. Malhotra; Malhotra, Ruchika, "A systematic review of machine learning techniques for software fault prediction", Applied Soft Computing Applied soft computing, 2015, Vol.27, p.504-518*
11. *Shreejith, Shanker, Bezborah Anshuman, and Suhaib A. Fahmy. "Accelerated artificial neural networks on FPGA for fault detection in automotive systems." 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2016.*
12. *Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y. and Zhai, C., 2014. Bug characteristics in open source software. Empirical software engineering, 19, pp.1665-1705.*
13. *Van De Ven, T. et al., 2018. World Quality Report 2018-19 - Artificial Intelligence - World Quality Report 2018-19 Findings: [Online] Available at: https://www.sogeti.com/globalassets/global/wqr201819/wqr-201819_secured.pdf.*
14. *W. B. Langdon, S. Yoo, M. Harman, "Inferring automatic test oracles[C]", 2017 IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST). IEEE, 2017:5-6.*
15. *Z. He, F. Peters, T. Menzies, Y. Yang, "Learning from open-source projects: An empirical study on defect prediction", Int. Symp. Empir. Softw. Eng. Meas., pp. 45-54, 2013.*

******