Testing of Artificial Intelligence in Software

Vishnuvardhan Vadla[1] , Abdul Sohail[2] , G. Pranay[3] , A.Vinuthna[4] , A.Kavya[5] , Shaik Afaq[6]
[1]Assistant Professor in Department of Electrical and Electronics Engineering
[2,3,4,5,6] Student scholar in Department of Computer Science and Engineering
St. Martin's Engineering College, Secunderabad, India

Abstract
Artificial intelligence (AI) is having a big impact on a lot of different fields, like the legal, medical, military, industrial, and artistic fields. AI can manage smart factories, drive autonomous cars, make accurate weather forecasts, detect cancer, and be a personal assistant, among other things. The process of evaluating software to look for anomalous behavior is known as software testing. The process of software testing is arduous, time-consuming, and unpleasant. In order to improve quality and delivery timeliness, automation solutions have been developed to assist in automating certain testing process operations. Automation technologies are losing effectiveness over time as a result of the continuous integration and delivery (CI/CD) pipeline. Because AI can review code for faults and bugs faster than humans and without human interaction, the testing community is looking to AI to fill the void. Our goal in this research is to identify how different aspects of the Software Testing Life Cycle (STLC) are affected by AI technology. The study also seeks to identify and elucidate some of the most significant obstacles that software testers encounter when utilizing AI in testing. Additionally, the report makes some significant predictions about how AI will advance software testing in the future.

Keywords: Artificial Intelligence, Machine Learning, Deep Learning, Software Testing, Software Testing Activities

1. Introduction
With recent progress in automated and digitized data acquisition, efficient machine learning and deep learning algorithms, and high computing infrastructure, Artificial Intelligence (AI) applications are now inflating their foot- print in areas that were previously expected to be only the domain of human experts. The AI-powered tools have al- ready made significant progress in various fields, includ- ing finance, law, medicine, and even arts. In many re- spects, AI is radically surpassing human intelligence and is approaching the domain of human creativity and em- pathy. Examples include AI's spectacular successes in winning Go [1], chess [2], and other board games with humans, and in surpassing humans on fully defined world puzzles. In the domain of NLP, we witnessed how a pow- erful language model like GPT3 wrote news articles that people found hard to distinguish from prose written by hu- mans [3]. We also witnessed DeepMind's protein-folding AI solving a 50-year-old grand challenge of biology [4]. Over the past few decades, there has been substantial sig- nificant growth in the software industry driven by the re- cent advances in AI. Artificial Intelligence is gradually changing the landscape of software engineering in general [5] and software testing in particular [6] both in research and industry as well.
In the last two decades, AI has been found to have made a considerable impact on the way we are approach- ing 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 can help us fill this gap and help us streamline our speeding software delivery process, thereby saving a significant amount of time and effort (and likely a sizeable amount of money too). So far, the use of AI has been very successful in the automation of software testing in some areas. Still, much research

re- mains to be carried out on analyzing, understanding and improving the tested software artefacts in order to learn more and develop better techniques to enable modern soft ware systems.

Our goal in this study is to identify software testing ac- tivities where AI has made a significant impact and greatly enhanced the process within each activity. We also iden- tify AI techniques that have been mostly applied to the process of software testing. Further, we convey the prob- lems identified by the study that the testing community is facing while implementing AI-based solutions to the test- ing problems. We also provide some key areas where AI can potentially help the testing community.

2. Background

Artificial Intelligence Overview: The term artificial intelligence 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 behaviour”. According to John McCarthy, it is “The sci- ence and engineering of making intelligent machines, es- pecially intelligent computer programs” [7]. Here we dis- cuss the main branches of AI that have been mostly ap- plied to software testing.

Artificial Neural Network : Designing artificial intel- ligence based on a biological neural network gives birth to an artificial neural network (ANN) [8]. Like the biological neural network, the ANN is an interconnection of nodes, analogous to neurons. Each neural network has three crit- ical components: node character, network topology, and learning rules. Node character determines how signals are processed by the node. Network topology determines the ways nodes are organized and connected. Learning rules automatically determine how the weights are initialized and adjusted using weight adjustment schemes. This type of network becomes a computational device, which is able to learn through training, consequently improving its per- formance.

AI planning : Research on AI planning can be traced back to the logic theorist program designed by Newell and Simon in the 1960s [9]. The task of AI planning is to find a series of effective actions in a given planning domain, to ensure that the initial state in the planning problem can be successfully transferred to the goal state after applying the actions [10][11].

Robotics : Robotics is a branch of AI, that comprises Electrical Engineering, Mechanical Engineering, and Com- puter Science for the design, construction, and applica- tion of robots. An Intelligent Robot is a physically situ- ated Intelligent Agent containing five major components: textiteffectors, perception, control, communications, and power [12]. Effectors are the peripherals of the robot that help it to move and interact with the environment. Per- ception is a set of sensors that provide the robot with the capability to sense the environment. Control is analogous to the central nervous system and is capable of computa- tions that allow the robot to maximize its chances of suc- cess. Communication is how a robot interacts with other agents like humans use language, gestures, and proxemics to interact with each other.

Machine Learning : Machine learning can be broadly defined as computational methods using experience to im- prove performance or to make accurate predictions [13]. Here, experience refers to the past information available to the learner, which typically takes the form of electronic data collected and made available for analysis. This data could be in the form of digitized human-labelled training sets, or other types of information obtained via interaction with the environment [13] [14].

Natural Language Processing (NLP) : Natural Lan- guage Processing (NLP) refers to the AI method of com- municating with an intelligent system using a natural lan- guage such as English. Processing of Natural Language is required when we want an intelligent system to perform as per our instructions, when we want to hear decisions from a dialogue-based clinical expert system, etc.

Fuzzy Logic : Fuzzy logic(FL) is a method of rea- soning that resembles human reasoning. The approach of FL imitates the way of decision-making in humans that in- volves all intermediate possibilities between digital values YES and NO. FL is based on the idea that there is no sharp distinction between the two extremes. FL is a method of reasoning that is applied to make

decisions by means of a number of rules which are combined with each other to produce a result. The rules are fuzzy sets, which are used as a basis for decision-making.

Expert Systems : Expert systems are computer appli- cations developed to solve complex problems in a partic- ular domain, at the level of extraordinary human intelli- gence and expertise. The common features of expert sys- tems can be summarized as follows.

Rules that define the specific problem are formal- ized in the form of computer procedures in the pro- gramming language.

Knowledge Base in the form of a computerized database, stores the problems and solutions for support in the decision-making process.

Inference Engine which processes and evaluates scenarios The problems posed and solutions found are completely transparent to the user. In simple terms, the system func- tions as a large, intelligent "computerized brain".

Software Testing Overview: Software testing is an in- vestigation conducted to provide stakeholders with infor- mation about the quality of the software product or system under test (SUT). Usually, a software development organi- zation expends between 30% to 40% of total project effort on testing [15] and testing consumes more than 50% of the total cost of a project [16]. A higher-quality software is achieved when SUT is failure-free. A failure is detected when the SUT's external behaviour is different from what is expected of the SUT according to its requirements or some other description of the expected behaviour [17].

An important element of the testing activity is the test case. Essentially, a test case specifies in which conditions the SUT must be executed in hopes of finding a failure. When a test case reveals a failure, it is considered suc- cessful (or effective) [18]. The test cases are usually de- rived from either the functional specification, or a design specification, or a requirements specification. A test case specification includes:

- The preconditions, which describe the environment and state of the SUT before the test case is executed.
- The test steps, which describe the actions that should be performed to execute the test case.
- The expected results, which describe the expected results of the executing test case.
- The actual results, which describe the results of the executing test case.

There are different dimensions under which testing has been studied and implemented and these dimensions define the test adequacy criteria, that is, the criterion that defines what constitutes an adequate test [19]. A great number of such criteria have been proposed and investi- gated and considerable research effort has attempted to provide support for the use of one criterion or another. We discuss test adequacy criteria in the following sections.

1) Testing Types: Two main types of testing are

- Manual Testing : In manual testing, testers exe- cute 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 behaviour or bug.
- Automated Testing : is a form of software testing that uses software tools to execute predefined tests. The software tools used for automated testing are often called test automation tools or test automation frameworks. It relieves the tester from the burden of executing the test cases however the process of planning and writing test cases in the form of test scripts still needs to be carried out manually.

2) Testing Techniques: Three main testing techniques have been identified

Black-box Testing : Black-box testing also known as functional testing aims to study the external be- haviour of software without dwelling on the internal structure of the software. Black-box testing is based on the inputs and the outputs of the software.

- White-box Testing : White-box Testing also known as structural testing, on the other hand, creates test cases based on the SUT implementation. Its pur- pose is to make sure that all structures (e.g., paths, instructions, and branches) of the SUT are exercised during the execution

of the test suite [18].

- Gray-box Testing : Gray box testing is a testing technique to test a software product or application with partial knowledge of the internal structure of the application. The purpose of grey box testing is to search and identify the defects due to improper code structure or improper use of applications.

3) Testing Phases or Testing Levels: Testing is per- formed at all levels of the software development lifecycle including development, release, and production. During development unit testing is carried out to test basic units of software like a method or a class. After unit testing, the basic units combine to form components further test- ing is carried out for testing these components to ensure that the integration has not bought any unintended bugs and the component are working as per the specification. The process of testing at the component level is called Integration testing. Since different teams work on the code simultaneously and there is much reusable and third-party code that is incorporated in the software a further level of testing known as system testing has been identified to test the integrated components from these sources and there- fore to test the system as a whole.

Most often due to the requirements change or addi- tion of functionality to software or due to maintenance the code changes, which may result in bugs crreping in the code apparently resulting in a failure. To tackle this a technique called regression testing is incorporated at all levels of testing. Regression testing is the most cumber- some and time-consuming testing technique as it involves testing the SUT whenever a change is incorporated.

Before the release, requirements testing ensures that the SUT is performing all the functions according to the requirements that have been pre-defined in the software requirement specification document. Scenario testing is carried out before the release where scenarios of the SUT are created and the SUT is tested against these scenarios to look for any unintended behaviour of the SUT. Per- formance testing is a testing measure that evaluates the speed, responsiveness, and stability of a SUT under a work- load.

At production time alpha testing is carried in the de- velopment environment wherein the developer acts as the user of the SUT and tries to identify any failure. In this testing technique, the developer actually looks at the SUT from the perspective of the user. Beta Testing is the testing of SUT in the user environment. Here the user actually in- teracts with the SUT and the developer just watches and analyses the SUT for any failure.

3. Impact of AI on Software Testing

The areas in which AI techniques have proved to be useful in software testing research and practice can be characterized by their applications in the software testing life cycle (STLC). From planning to reporting, AI tech- niques have made a dominant imprint across all the stages of STLC. To study the impact of AI on software testing we have identified testing activities or testing facets for which considerable and significant research has been carried out by applying AI. These testing activities cover most of the STLC.

Test Specification : At the beginning of the software test- ing life cycle, the test cases are written based on the fea- tures and requirements of the software. The test cases are written in a checklist type test specification document to ensure that every requirement of the software is tested. It includes the purpose of a specific test, identifies the re- quired inputs and expected results, provides step-by-step procedures for executing the test and outlines the pass/fail criteria for determining acceptance. Below we mention the work of two seminal papers where AI has been ap- plied to this activity.

Last and Friedman [21] demonstrated the potential use of Info-Fuzzy Networks (IFN) for automated induction of functional requirements from execution data. The in- duced models of tested software were utilized for recov- ering missing and incomplete specifications, designing a minimal set of regression tests, and evaluating the cor- rectness of software outputs when testing new, potentially flawed releases of the system.

Briand et al. [20] proposes a methodology that takes as inputs the test suite (a set of test cases) and test specifi- cations developed using the Category-Partition (CP) strat- egy. Based on the CP specification, test cases are trans- formed into abstract test cases which are tuples of pairs (category, choice) associated with an output equivalence class (instead of raw inputs/outputs). C4.5 is then used to learn rules that relate pairs (category, choice), modelling input properties, to output equivalence classes. These rules are in turn analyzed to determine potential improvements of the test suite (e.g., redundant test cases, need for addi- tional test cases) as well as improvements of the CP spec- ification (e.g., need to add a category or choices).

Test Case Refinement: Test case refinement is a planned activity that is employed by testers to select the most ef- fective test cases for execution consequently reducing the testing cost. We identified two AI techniques applied to this testing activity.

Info-Fuzzy Networks (IFN) was used by Last and Kan- del [22] and Last et al.[23] who presented a novel ap- proach to automated reduction of combinatorial black-box tests, based on automated identification of input-output re- lationships from execution data of the tested program. Singh et al. [24] details an approach generating test cases from Z specifications for partition testing. The learner receives as input the functional specification in Z. As out- put, the approach produces a classification tree describing high-level test cases. Then the high-level test cases are further refined by generating a disjunctive normal form for them.

Test Case Generation : After devising a test ade- quacy criteria it is the job of testers to formulate a test set that satisfies the test adequacy criteria. Since for complex applications, the job of handcrafting test sets is an unman- ageable task most of the testers use automatic test case generation techniques. In the last two decades, there has been considerable growing interest in applying AI to auto- mate test case generation and AI has impacted this testing activity significantly.

Test Oracle Construction : Software testing is the process of verifying the correct behaviour of the SUT as per the requirements. To highlight this when a program is run with a certain input a mechanism is needed to distin- guish between the correct and incorrect behaviour of the

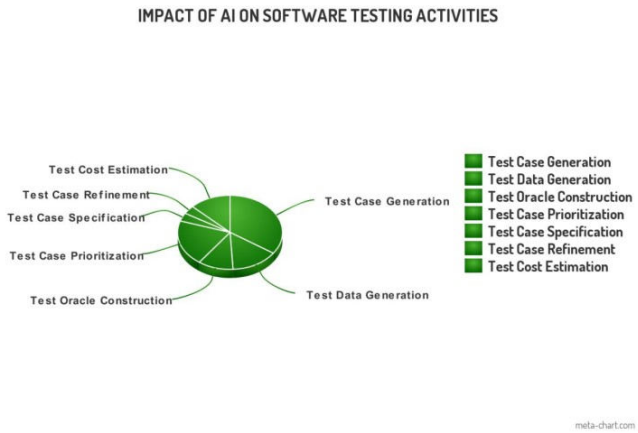

Figure 1: Impact of AI on software Testing Activities

Based on the discovered publications, seven oftware testing activities viz test case generation, test oracle gen- eration, test data generation, test case prioritization, test case specification, test case refinement, and test cost es- timation were identified as activities that have been im- proved significantly by the application of AI techniques. From this study, we can infer that test case generation or test case design activity has been considerably enhanced by the application of AI techniques. Most of the recent research has been carried out around activities like test case generation, test case prioritization, test data gener- ation and test oracle construction. The trivial reason for this is that these activities are more important than other activities in the STLC. We skipped some software testing activities including test harness, testing technique selec- tion, test repairing, change proness etc. from our study as just one or two AI-based studies have been carried for these activities. [Table 1] shows a list of AI techniques that have been applied for

software testing activities.

Also, the most commonly used AI techniques applied to soft testing appear to be solving the problem of optimiza- tion across various software testing activities. Specifi- cally, genetic algorithms, ANN, and reinforcement learn- ing were among the techniques that were used across var- ious testing activities more frequently than others.

4. Problems and Challenges of AI in Software Testing

Considering the lack of industrial expertise and re- search work this section outlines some of the open prob- lems and challenges in the application of AI to software testing.

Test Oracle the biggest challenge : Test oracle prob- lem is a companion of every researcher and practitioner working in the field of software testing. It has been there from the inception of the software testing conundrum and the problem is expected to stay with us for a longer dura- tion or maybe forever. Despite continuous attempts to mit- igate the problem of the test oracle, researchers have been able to solve this problem for a static subset of SUT's. As soon as the dynamic traits of the SUT start to display the previous test oracle derived for the SUT starts to lose ef- fectiveness. In many scenarios, even the documentation from which test oracles are generated is missing in the re- quirements document. To cope with this dynamism and to dream of a documentation-free effective test oracle AI techniques have been employed and these AI techniques have provided a significant initial effort towards realizing this dream.

Availability of Data : Any model in AI must be trained and tested before being deployed in production. The ef- ficiency and effectiveness of a model are highly corre- lated to the amount of data with which a model is trained and tested. Acquiring data for building AI models in the domain of software testing is a challenging task because software testing unlike other fields of study is not fully automated yet. Apparently, a lot of testing is still being carried manually and it is difficult to capture data when testing is manual. This exhumes as a bottleneck to data acquisition for training AI models in the future.

Adaptiveness to data : AI models are highly depen- dant on the data with which they are trained and tested An important phase in the production of an AI model is the collection of robust datasets from real-world scenarios and the use of that data to train a model generalized to fit that data. Such a model assumes future data and historic data (data with which the model is trained) to be from the same distribution. However, it is often not the case as most data has higher disparity over time e.g. learning customers shopping behaviour is dependant on seasons. AI models are evaluated for generalizations by testing the model on a particular subset from the data (test set) which is from the same time distribution. Over time less promis- ing outcomes from such models are witnessed. Some AI techniques allow the model to be readjusted to adapt to the changes in the new data. However, the challenging task is to detect this ideal time to readjust and even automate the readjustment process.

Identifying Test Data : Every AI model must be tested horoughly before being put to production. Model test- ing is like a black-box technique where the structural or logical information regarding the model is not a necessity. Rather comprehensive information and understanding re- garding the testing data is required. Again the selection of testing data from the same distribution can incur issues resulting in a biased model. The problem is with the cov- erage of the test set i:e asking the question "Is the model tested over a larger distribution of data?" Identification of such coverage-based test datasets is a challenging task in the domain of software testing.

Exhaustive search space leads to generality loss : For most of the optimization problems in search-based soft- ware testing, the AI algorithm has to exhaustively search for the solution or the goal. Although sub-optimal search strategies have been identified and implemented so far they work for a particular class of problems. To include more general solutions to a variety of problems the inputs of the whole problem domain are to be included, consequently making the input space more exhaustive. Versatile and broadly capable AI methodologies need to be identified to cope with this generality loss.

Exploitation of Multicore Computation : A lot of AI techniques are highly computationally expensive making them potentially incompatible with large-scale problems faced by software testers. With the recent advancements in computing infrastructure, Graphical Processing Units (GPU) and Tensor Processing Units (TPU) have been in- corporated at scale for these techniques. More work is re- quired to fully exploit the enormous potential of the rapidly increasing number of processors available. Since such high computational devices are expensive more work needs to be carried out towards designing techniques that require less computation and still match the performance of high computing devices.

5. Prospects of AI in Software Testing

In the past few years, many companies have begun to invest in AI-powered software testing technologies. These AI systems offer an alternative to traditional testing pro- cesses. While AI systems are still relatively new, the po- tential gains are simply too great to ignore. Here are some excerpts from our study and from software testing indus- try experts where we expect these technologies to poten- tially help software testers in the future:• Collaborating with people who are geographically spread out can be difficult. This is where AI sys- tems can be relied upon to carry out routine, labour- intensive tasks. This frees up more productive time for software testers to spend on addressing the most complicated issues.

- Simulated testing - The ability to program AI sys- tems to test application code is incredibly useful. It offers a realistic simulation of a situation that a soft- ware tester might face. This also improves the accu- racy of tests because they can identify and replicate all possible scenarios.
- The next generation of artificial intelligence in soft- ware testing will include self-modifying tools that can instantly identify and fix vulnerabilities without any human intervention creating self-healing sys- tems.
- With artificial intelligence in software testing, soft- ware companies, and testers can reduce their costs by a great degree, which is already happening. We think it will normal to see organizations and other user groups automate their testing process using AI while testers focus on the exploratory testing of sys- tems.
- The AI predictive analytics will play a major role in discovering all possible test cases and will make the software products more robust, reliable and will exceed customer expectations.
- AI is operating at all levels of testing from planning to execution to reporting and it is expected to take over all the tasks in the STLC which require human intelligence. This in turn will free the tester from the job of various time-consuming testing strategies like regression testing and smoke testing etc.
- AI incorporated in testing will provide a high ROI because these systems ensure that the time allocated to deliver the product is spent on polishing its fea- tures rather than on testing and debugging technical defects.
- AI-pwered automation tools will help to increase the level of autonomy in software testing and hence help to deliver higher quality software. AI-related technologies are helping to bridge the gap between human and machine-driven testing capabilities.
    - AI is expected to impact testing in all the software product areas including mobile applications, web applications, IoT, embedded systems, database ap- plications, gaming industry, real-time applications, critical software applications to name a few.
- With more data being acquired and stored, AI can enhance the software testing capabilities which are somewhat restricted today due to the non-availability of data.

6. Conclusion

In the last two decades, the rapid growth of interest in topics where AI has been applied to software testing is a testimony to the appetite the software testing community has for AI. This is a consequence of AI providing efficient solutions to the problems faced by the testing community for

a long time. AI has already been accepted as a promis- ing solution to many problems faced by testers all around the globe. In this paper, we studied the impact of AI across all stages of the STLC. We identified seven software test- ing activities that were most enhanced by AI techniques. GA's, Reinforcement Learning, and ANN were among the most widely used techniques from the domain of AI. We identified problems and challenges researchers and testers face while applying AI techniques to software testing. We also provided a future prospect into how AI can shape the software testing domain.

References

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

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

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

neering. In 1st International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2012), Zurich, Switzerland, 2012. protein-folding-ai-solved-biology-science-drugs-disease/

[5] M. Harman. The role of artificial intelligence in software engi-

[6] Hourani, H., Hammad, A., Lafi, M. (2019, April). The Impact of Artificial Intelligence on Software Testing. In 2019 IEEE Jor- dan International Joint Conference on Electrical Engineering and Information Technology (JEEIT) (pp. 565-570). IEEE.

[7] J. McCarthy, "Programs with common sense," in Proceedings of the Symposium on Mechanisation of Thought Processes, vol. 1. London: Her Majesty's Stationery Office, 1958, pp. 77–84.

[8] Zou J., Han Y., So SS. (2008) Overview of Artificial Neu- ral Networks. In: Livingstone D.J. (eds) Artificial Neural Net- works. Methods in Molecular Biology™, vol 458. Humana Press. https://doi.org/10.1007/978-1-60327-101-12

[9] Newell, A., and Simon, H. A. 1963. GPS: A Program That Sim- ulates Human Thought. In Computers and Thought, eds. E. A. Feigenbaum and J. Feldman. New York: McGraw-Hill. [GPS]

[10] Jiao, Z.; Yao, P.; Zhang, J.; Wan, L.; Wang, X. Capability Con- struction of C4ISR Based on AI Planning. IEEE Access 2019, 7, 31997–32008.

[11] James Hendler, Austin Tate, and Mark Drummond, "AI Plan- ning: Systems and Techniques," AI Magazine Volume 11 Num- ber 2 (1990)

[12] Robin R Murphy, "Introduction to AI Robotics Second Edition, " The MIT Press, Cambridge, Massachusetts, London England, 2019.

[13] M. Mohri, A. Rostamizadeh, and A. Talwalkar, Foundations of Machine Learning. Cambridge, MA, USA: MIT Press, 2012. [14] P. Louridas and C. Ebert, "Machine learning," IEEE Softw., vol. 33, no. 5, pp. 110–115, Sep./Oct. 2016.

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

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

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

[18] Vinicius H. S. Durelli , Rafael S. Durelli , Simone S. Borges, An- dre 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 RE- LIABILITY. 2019.

[19] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test cov- erage and adequacy," ACM Comput. Surveys, vol. 29, no. 4, pp.
366–427, 1997.

[20] INDI,2014. L. C. Briand, Y. Labiche, and Z. Bawar, "Using Ma- chine Learning to Refine Black-Box Test Specifications and Test Suites," 2008 The Eighth International Conference on Quality Software, 2008.

[21] Mark Last and Menahem Friedman, "BLACK-BOX TEST- ING WITH INFO-FUZZY NETWORKS," Artificial Intelligence Methods in Software Testing, pp. 1-20 (2004)

[22] Last M., Kandel A. (2003) Automated Test Reduction Using an Info-Fuzzy Network. In: Khoshgoftaar T.M. (eds) Software Engineering with Computational Intelligence. The Springer In- ternational Series in Engineering and Computer Science, vol
731. Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-0429-09