International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2019 IJSRCSEIT | Volume 5 | Issue 4 | ISSN : 2456-3307
DOI : https://doi.org/10.32628/CSEIT195434

Practical Applications of Artificial Intelligence in Software Testing

Mesut Durukal
IOT DS EU TR PLT, Siemens AS, Istanbul, Turkey
mesut.durukal@siemens.com

ABSTRACT

This paper presents the use of artificial intelligence in each software testing stage. In this context, the necessity to use AI (artificial intelligence) in software testing with its effects and outcomes is discussed. Then, practical applications and the advantages are analyzed. The main goal is to make insights about what can be done in different stages of software testing by means of AI.

Keywords: Artificial Intelligence, Machine Learning, Software Testing, Test Automation.

I. INTRODUCTION

Software applications today have very comprehensive features and use cases. Most of them interact with other applications through APIs and connect to various platforms, which results in a remarkably wide scope and complexity.

Another fundamental aspect of testing is time, in addition to scope. When highly dynamic software lifecycles with continuous integration and deployment activities are considered, the need to react quickly against frequent changes can easily be understood.

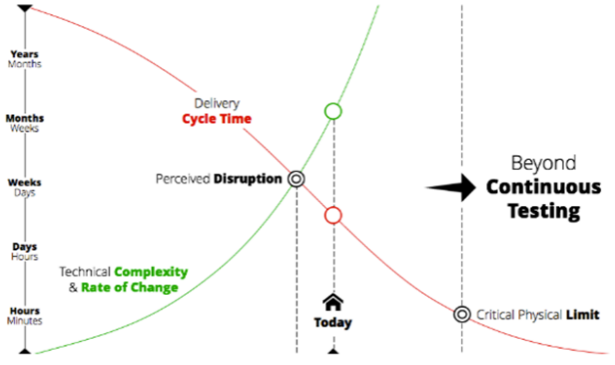

Figure 1. Delivery time versus complexity of products [1].

Products should have competitive features to survive in modern world. On one hand, they should adapt new functionalities and be compatible to new technologies. On the other hand, they should respond to rapid changes not to be old fashioned and be one of the first in the market.

The challenges in product development have some reflections in software testing. Necessity to take quick actions against gaps introduced by complexity and fast changes in testing cycles is unneglectable. In this manner; every possible solution to overcome raising challenges is tried to be applied on testing. Probably; at this point, one of the most exciting candidates is the introduction of machine-based intelligence into testing [2]. AI practices promise for save on time and additional coverage.

Mainly; faster, better, and cheaper solutions are the most promising expectations [4] from usage of AI in testing. Apart from these expectations, testing activities are supposed to be more empirical instead of deterministic situations. The nature of machine learning supports this idea since machine models are continuously improved with observations.

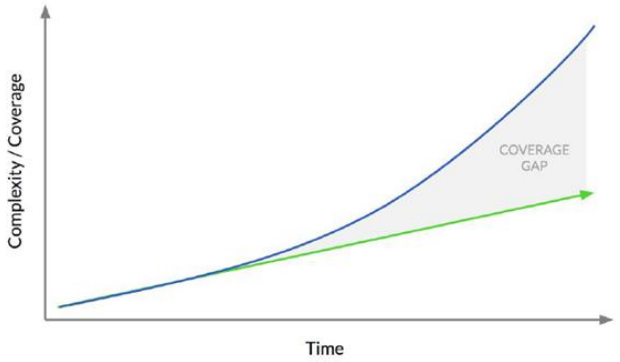

Figure 2. Test coverage gap [3].

To illustrate how big a game changer AI is supposed to be; only the investments, which is estimated as $15.7 trillion in 2030 [5], would be sufficient. Another survey [6] foresees the budgets for AI projects by 2025 as $90BN.

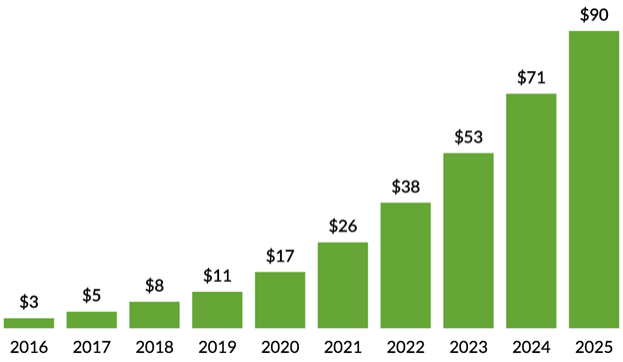

Figure 3. AI Projected Revenue in $ Billion [6].

In this paper, possible AI practices on software testing stages are investigated. Section II describes artificial intelligence working principles. Section III explains several applications, where the outcomes are analyzed in Section IV. Finally, summary of the work is addressed in Section V. The acknowledgement and references close the article.

II. METHODS AND MATERIAL

To reduce manual effort, several automation processes are integrated into projects. However, human testers are still needed for these activities [7]:

- defining testing goals,
- acquiring the knowledge needed to test the system,
- designing and specifying detailed test scenarios,
- writing the test automation scripts,
- executing scenarios that could not be automated,
- analyzing the results to determine threads.

The difference between machines and human should be clarified first to get rid of human effort. Machines are mainly programmed to follow explicit instructions, where humans learn a lot through observation and experience. In this point, machine learning (ML) is the key factor for reducing this gap as much as possible.

Although machine learning is regarded as a subdiscipline of AI, most of the time they have been used for each other. Machine learning is defined by Arthur Samuel in 1959 as "the subfield of computer science that gives computers the ability to learn without being explicitly programmed" just like human beings. If the performance of machine improves with experiences, it means that it is learning [7].

Machine learning algorithms run in two stages: Training and execution. First, machine learns the system; or in other words, it models the system, which is called as training. Then the execution is performed, where machine predicts next steps according to learnt experiences. ML types can be classified as Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning.

A. Supervised Learning

Supervised machine learning algorithms apply what has been learned in the past to new data. They use labeled examples to learn and predict future events. Starting from the analysis of a known training dataset, the algorithm builds a model to make predictions about the output values. System provides targets for any new input after sufficient training.

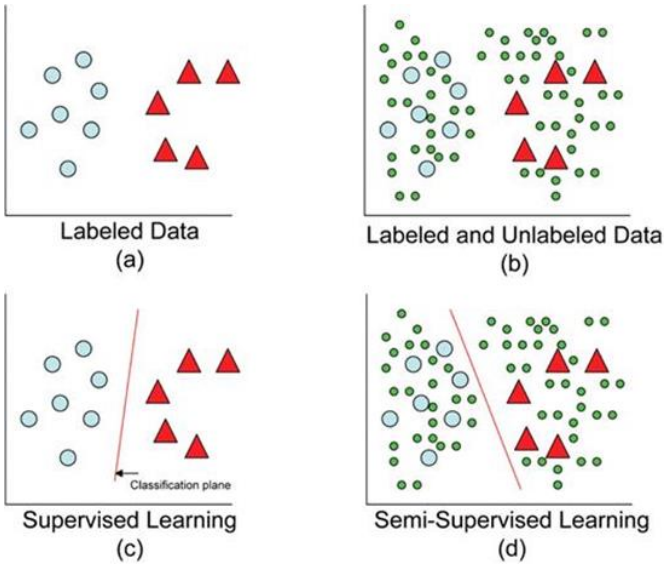

Figure 4. Use of labeling in supervised and semi-supervised learning [8].

B. Unsupervised Learning

Unsupervised ML algorithms are used when the information used to train is neither classified nor labeled. Under these conditions, system builds a model to describe a hidden structure from unlabeled data. The system is not expected to estimate the right output, but it explores the data, draws outcomes from datasets and finally describes hidden structures from unlabeled data.

C. Semi-supervised Learning

Semi-supervised ML algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data.

D. Reinforcement Learning

Reinforcement ML algorithm is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and reward are the most relevant characteristics of reinforcement learning. Simple reward feedback is required for the machine to learn which action is best; which is known as the reinforcement signal.

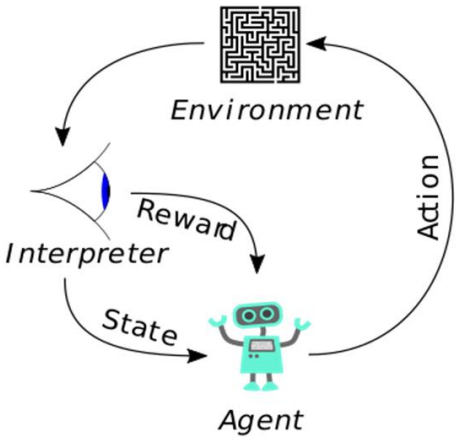

Figure 5. Reinforcement Learning [9].

Lots of applications for supervised (classification, regression, anomaly detection), unsupervised (clustering, association, dimension reduction) or reinforcement learning algorithms are developed in various models like Neural Networks, Support Vector Machines, Principal Component Analysis, k Means, k Nearest Neighbors.

III. APPLICATIONS

AI is utilized in different stages of testing. In this section, all application methodologies are investigated in detail.

Figure 6. Software Testing Stages.

E. *Test Definition*

To ensure product quality, test scenarios are defined and executed to cover use cases. AI improves quality and reduces manual effort in test definition stage in different ways.

Model-Based Test Generation: In this model, machine learns the use cases in the system by observing actions and reactions. In this way, the mandatory parameters and expected inputs are learnt. Similarly, error

messages in negative scenarios are observed. At the end of the learning phase, test cases are generated to verify expected results and behaviors. An example for model-based AI-driven test generation platform called AIST [7], which explores user scenarios to generate test cases.

Test Generation from UML Diagrams: A test generation alternative is analyzing UML diagrams.

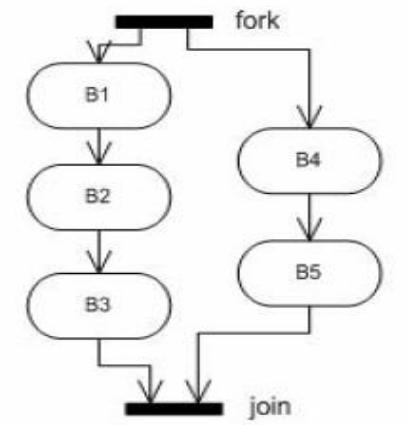

Figure 7. A basic activity diagram [10].

As proposed in a research [10], test cases are generated by covering all the transitions in an activity diagram. In the study, 10 test cases are generated according to the diagram.

TABLE I. TEST SCENARIOS GENERATED FROM ACTIVITY DIAGRAM

Number	Scenario
1	B4 -> B1 -> B2 -> B3 -> B5
2	B4 -> B1 -> B2 -> B5 -> B3
3	B4 -> B1 -> B5 -> B2 -> B3
4	B4 -> B5 -> B1 -> B2 -> B3
5	B1 -> B4 -> B2 -> B5 -> B3
6	B1 -> B4 -> B2 -> B3 -> B5
7	B1 -> B4 -> B5 -> B2 -> B3
8	B1 -> B2 -> B3 -> B4 -> B5
9	B1 -> B2 -> B4 -> B3 -> B5
10	B1 -> B2 -> B4 -> B5 -> B3

F. Implementation

In continuous testing environments, no one would refuse an increase in test implementation speed.

Code Generation: For robots a way to write codes is, first understanding the problem and then applying the solution. When the problem is defined with inputs and outputs, then needed operations are predicted ad related codes are generated.

Consider how DeepCoder [11] does it: If you want to filter numbers smaller than 0, and list them after multiplying with 4 in a reserve sorted order, here is an example IO:

An input-output example:
Input:
`[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]`
Output:
`[-12, -20, -32, -36, -68]`

Figure 8. Input and outputs of the problem [11].

After checking inputs and outputs, according to learnt patterns, DeepCoder predicts needed operations:

Figure 9. Predicted operations [11].

Auto-Completion: After most used patterns are learnt, subsequent codes are proposed during implementation. Tabnine [12] is an example application which facilitates test implementation.

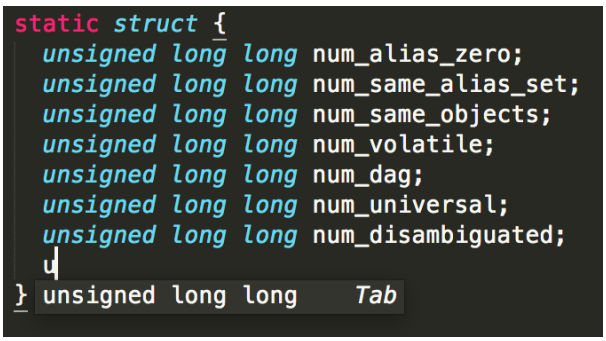

Figure 10. An auto-completion application: Tabnine.

G. Execution

Exploratory Testing: After new feature deployments, AI bots click every new button added to the application to test new functionalities. Additionally, changes on UI are detected and images removed from the application are noticed. Consequently, machine starts to learn about the application and understand relations between the modules. Finally, test cases are generated according to these relations.

Usability and Efficiency Tests: Adam Carmi, co-founder and CTO of Applitools [13], states: "We want to make sure that the UI itself looks right to the user and that each UI element appears in the right color, shape, position, and size." ML algorithms are used in their tool Applitools to perform usability and efficiency tests.

The system is modeled by the machine according to use cases. Parameters for difficult and easy paths are extracted, and new designs are oriented by these trainings.

Execution Analysis: During test executions, AI algorithms learn patterns and user tendencies by collecting data, taking screenshots, downloading the content of web pages and measuring loading times. Then, properties of new features are estimated, and the deviations are detected. For instance, if loading time of a new page is longer than predicted, a warning is raised. Some outlier detection algorithms are applied with Info Fuzzy Network [14].

H. Maintanence and Grouping

Refactoring: According to learnt patterns, some applications like DeepCode [15] proposes solutions against code smells.

Self-Healing: For UI test automation, unusable and broken locators are the most challenging problems. When locators change, tests fail since the codes do not identify the expected module any more. Self-healing algorithms are developed to get rid of this challenge. AnyUI Engine [16] proposes that codes can be refactored by recognizing most stable properties. In this way, codes related to changing parameters are refactored by referring stable ones.

Prioritization: Infinite testing is impossible. In a limited testing time, the most prior test cases should be selected and executed. Priority is decided according to [17]:

- The probability to find an error,
- Uniqueness in terms of scope,
- Complexity or simplicity,
- Fitness for the regression activity.

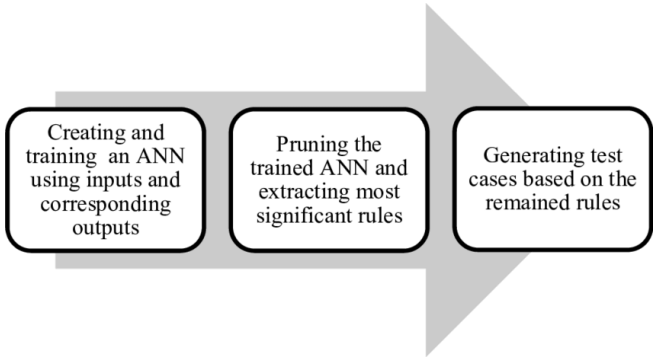

Figure 11. Test prioritization and reduction with ANN [4].

Therefore; for prioritization, test cases are evaluated according to learnings which are collected from the labelled training sets. Test prioritization is developed with various approaches such as ANN [4] and Genetic algorithms [18].

Suite Generation: After a change, at least the regression suite is executed. ML algorithms train the relations between test cases and the features; and decide related test suite for a feature.

It is possible to construct a group of similar tests by observing the coverages of tests during their executions [19]:
Branch Coverage: According to hits to a branch; algorithm calculates the distances of executions to the

target branch and relation between a test and a branch is estimated.

Line Coverage: Distances are calculated with covered lines in the code excluding the comments.

Exception Coverage: Exception coverage is a kind of reinforcement learning and aims as much exceptions as possible. Tests which throw more exceptions are rewarded.

Method Coverage: Method coverage approach applies the same algorithms over methods. Tests are evaluated according to whether they call methods or not.

I. Bug Handling

Classification: Bug classification provides hints about the weaknesses of the product. For example, if bugs mostly heap together on a feature, some actions can be taken accordingly. In this case, related tests are prioritized to investigate the feature deeper.

Addressing: For big teams, it is not easy to know every assignee for all features. In such cases, the address can be proposed by the machine according to previously addressed bugs. Correctly assigned bugs are labeled and the system is trained by the machine. Then addresses for next bugs are estimated.

Scoring: Scoring of the bugs are very important since they are handled according to their severities. First bugs with highest severity levels are fixed, then others are handled in order. If a critical bug is not scored with a high severity, it may be postponed since it is not regarded as a priority. Therefore, the fix is not done in time, which leads to additional costs.

AI is applicable in all stages of software testing cycles. Summary of application areas with algorithms is as follows.

TABLE II. AI APPLICATIONS IN TESTING

Stage	Application	Platform/Algorithm
Definition	Test Case Generation	AIST [7] Object Recognition: ANN Text Generation: Random Forest, kNN, SVM, Bayesian Networks UML-Based Generation
Implementation	Code Generation	DeepCoder [11]: NN
	Auto Completion	TabNine [12]: NLP
Execution	Exploratory Testing	Applitools [13] Visual AI (AI powered cognitive vision
	Usability, Efficiency	
	Execution Analysis	
Maintenance	Refactoring	DeepCode [15]
	Self-Healing	AnyUI Engine 3.0 [16] Pattern-Based Recognition
	Prioritization	ANN, GA
	Suite Generation	Search-Based Modelling
Bug Handling	Bug Addressing	Naïve Bayes, K-Means clustering
	Bug Classification	
	Bug Scoring	

IV. DISCUSSION

Usage of AI in testing activities has lots of advantages. Firstly, test coverage is improved by means of AI. Machines learn system patterns and tests are generated automatically.

In addition, AI applications provide extra speed in all stages of testing. Compared to humans, machines decide much faster. At least for the rough estimations, AI results can provide a quick feedback.

In the same context, manual effort is obviously reduced. Instead of manual tasks, machines work for defining, executing and maintaining tests. Moreover, outliers are detected by algorithms during. In this way, the risk to miss bugs are minimized and cost is reduced with early fixes.

Advantages of AI applications in testing are unneglectable; however, potential risks should not be ignored. Performance, security, control and social risks can be faced in failure cases [20]. Error cases can result in misleading actions, even security risks or fatal consequences [21]. Furthermore; if AI goes out of control, or is abused by people, some ethical and social concerns can arise. In short, it can be concluded that AI is a safe and beneficial tool only when it is under control.

V. CONCLUSION AND FUTURE WORK

Rapidly improving software world grows a great rivalry and create a pressure on stakeholders in terms of time, cost and scope. As other development processes, these challenges are faced during testing cycles as well. Thus; anything that can help to achieve this, is welcome to be adopted into processes. In this respect, AI is probably the most promising discipline to improve testing by making better and faster decisions.

Even though at least for now; it is assumed that Artificial Intelligence can never fully replace human beings, it can easily be seen that it is already equaling or surpassing humans in several tasks such as playing games, driving cars and providing recommendations. As far as these advances are concerned, the goal is to make use of AI in testing as much as possible.

There are possible applications which can be applied in all testing stages. AI algorithms provides a remarkable benefit on testing activities. It contributes with test coverage improvement, manual effort reduction, better conclusion and addressing.

As a future work, it is aimed to increase the artificial “intelligence” and decrease the necessity for manual effort. To achieve this, more application areas are supposed to found. However, another aspect of the issue is to measure how beneficial the applications are. In other words, it is expected to be able to measure how intelligent the system is.

VI. ACKNOWLEDGEMENT

I would like to thank my team mates for their precious help. Additionally, I am very grateful to my manager, Mr. Kamil Yıldırgan who has always supported and encouraged me to prepare this paper. Finally, I appreciate Ms. Elif Yilal, Ms. Buse Ozarslan and Ms. Eylul Akar who have reviewed the paper and guided me for the improvements.

VII. REFERENCES

[1] W. Platz, "What’s beyond continuous testing? AI," SD Times, 2017.

[2] W. Murray, P. Karuppiah, C. Stancombe," On the way to smart, intelligent, and cognitive QA," World Quality Report 2017-18, 9th edition, 2017.

[3] J. Arbon, "AI for Software Testing," Pacific NW Software Quality Conference, 2017.

[4] Dr. A. P. Nirmala, Md Shajahan, Somnath K, "Impact of Artificial Intelligence in Software

Testing," International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 3, Issue 3, pp.1519-1526, 2018.

[5] "Sizing the prize," PwC, World Economic Forum, Dalian, 2017.

[6] "Which Industries Are Investing in Artificial Intelligence?," Splunk, Priceonomics Data Studio, 2018.

[7] T. King, "AI Driven Testing: A New Era of Test Automation," Japan Symposium on Software Testing JaSST, 2019

[8] A. R. Shah, C. S. Oehmen, B. Webb-Robertson, "SVM-HUSTLE—an iterative semi-supervised machine learning approach for pairwise protein remote homology detection," Bioinformatics, Volume 24, Issue 6, 15 March 2008, pp. 783–790, doi: 10.1093/bioinformatics/btn028

[9] Reinforcement learning [Online] Available from: https://en.wikipedia.org/wiki/Reinforcement_learning/ 2019.07.19

[10] H. Kim, S. Kang, J. Baik, I. Ko, "Test Cases Generation from UML Activity Diagrams," Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007, doi: 10.1109/SNPD.2007.189

[11] M. Balog, A. Gaunt, M. Brockschmidt, S. Nowozin, D. Tarlow, "DeepCoder: Learning to Write Programs," Proceedings of ICLR'17, March 2017

[12] Tabnine [Online] Available from: https://tabnine.com/ 2019.07.19

[13] Applitools [Online] Available from: https://applitools.com/ 2019.07.19

[14] M. Last, M. Freidman, "Black-Box Testing with Info-Fuzzy Networks," World Scientific, City, 2004.

[15] DeepCode [Online] Available from: https://www.deepcode.ai/ 2019.07.19

[16] I. Philipp, "AI in Software Testing: A Reality Check," Tricentis, 2018.

[17] P. Saraph, M. Last, A. Kandell, "Test case generation and reduction by automated input-output analysis," Institute of Electrical and Electronics Engineers Inc., City, 2003.

[18] S. Dhawan, K. S. Handa, R. Kumar, "Optimization of software testing using genetic algorithms," In Proceedings of the 11th WSEAS international conference on Mathematical and computational methods in science and engineering (MACMESE'09), World Scientific and Engineering Academy and Society (WSEAS), pp. 108-112, 2009.

[19] J. M. Rojas, J. Campos, M. Vivanti, G. Fraser, A. Arcuri, "Combining multiple coverage criteria in search-based unit test generation," Springer International Publishing, Search-Based Software Engineering, volume 9275 of Lecture Notes in Computer Science, pp. 93–108, 2015.

[20] Dr. A. S. Rao, "Responsible AI & National AI Strategies," 4th International Initiatives, Europian Union Commission.

[21] S. Levin, J. C. Wong, "Self-driving Uber kills Arizona woman in first fatal crash involving," The Guardian, March. 19, 2018.

Cite this article as :

Mesut Durukal, "Practical Applications of Artificial Intelligence in Software Testing", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 5 Issue 4, pp. 198-205, July-August 2019. Available at doi : https://doi.org/10.32628/CSEIT195434
Journal URL : http://ijsrcseit.com/CSEIT195434