International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 3 | ISSN : 2456-3307

Impact of Artificial Intelligence in Software Testing

Dr. A. P. Nirmala[1], Md Shajahan[2], Somnath K[3]

[1]Senior Assistant Professor, Master of Computer Applications, New Horizon College of Engineering, Bengaluru, Karnataka, India

[2]Master of Computer Applications, New Horizon College of Engineering, Bengaluru, Karnataka, India

[3]Master of Computer Applications, New Horizon College of Engineering, Bengaluru, Karnataka, India

ABSTRACT

Since computer's software applications rapidly increased in modern life, it is important to have enough reliability and minimizing the probability of faults in software products. Software testing is a process to find faults in software's products, due to increase software reliability. Because testing process is very costly, automation techniques are needed to reduce these costs and also, increase reliability. In automated testing, the testing phases or part of them performed by intelligent methods, in order to reduce human role in the process. Automatic testing has several advantages such as increase testing speed, quality and reliability, decrease testing resources and costs. In this paper, after explaining software testing phases, we classified methods which can use in automated software testing phases based on previous researches with aim to reach above advantages.

Keywords: AI impact on Software testing, Main Impact on Various Areas, Roles of AI in adapting, The bridge between AI and human testers.

I. INTRODUCTION

The CEO and founder of AppDiff Jason Arbon shared a funny live e.g. with the world, "making a gesture to manually roll down window of a car" would make his kids giggle at him. Well why won't they, they haven't seen the good old days of a Maruti 800 or the era of 1990's. Surely the gap between the years of their father's youth and the fatherhood changed transformed and gave the young ones a new world to breathe in, to which many of them has stated to be the best and the worst phase of science. Jason worked for Google and Microsoft, he is a developer and a tester both, a very write person to start with who can answer the best on How artificial intelligence is impacting software testing. Just the way today's generation finds it funny to roll down on the up gesture of rolling down a window manually Jason states that the coming generation is going to laugh at the notion of testing done with methods being followed today selecting, managing and driving system under test will be all so out of fashion and time consuming for them, why won't they entertain artificial l intelligence which will give them more accuracy and consume less time, more of all it will also cut the manual work which ends getting paid in monthly salary, an advantage and disadvantage for the young ones to see in coming days.

Automatic testing has several advantages such as increase testing speed, quality and reliability, decrease testing resources and costs. In this paper, after explaining software testing phases, we classified methods which can use in automated software testing phases based on previous researches with aim to reach above advantages.

This method classification has performed based on their applications in software testing phases and effects on test automation.

II. SOFTWARE TESTING PHASES

Based on [4] , testing process can divide into four phases which explains in following subsections. With this classification, a framework created to imply testers must consider which problems before moving to next problem and which phase can automate by which methods those mention in section 3.

A. Modeling the software's environment

Usually these interactions performed via interfaces such as human, software, file system and communication interfaces. Methods that can simulate the interfaces may usable for automating this phase.

B. Selecting test scenarios

In this phase, testers must select proper test scenarios and *Test Cases* that covering each line of source code, input sequences and execution paths to ensure all software's modules tested adequately. Because the number of test cases can be very large to execute them all in limited testing time, this is very important to selecting test cases that have higher probability of finding errors. They are some methods that can effectively automate test case selection.

C. Running and evaluating test scenarios

After preparing and selecting test cases, testers must execute them and then, they must evaluate outputs to find if there is a fault. Testers compare the outputs generated by executed test cases and the expected outputs based on defined specifications in analysis phase and system specifications. Automation process requires a method to mapping each input to corresponding output of the entire operational environment and a tool for comparing these outputs. In section 4, an intelligent input/output mapping technique is introduced.

Sometimes expected outputs are not clearly defined. This may duo to uncertainty in software's behavior or lack of complete specification.

D. Measuring testing process

It is very important to identify what is the status of testing process and when the testing process can stop. Testers need quantitative measurement for determine the process status by cognizing the number of bugs in the software and the probability that any of these bugs will be discovered. Some software quality estimation techniques can applicable for automation of this process.

III. AUTOMATED SOFWARE TESTING METHODS CLASSIFICATION

These methods applied for automating a phase or at least some part of a phase in software testing process. As mentioned before, the classification was based on software testing phases and the applications of methods in software testing phase automation. In following, an attempt is made to explain such methods.

A. Modeling the Software's Environment (Phase 1)

Since regression testing is a process to retest functionalities of software that remain in new versions, Regression GUI Testing is a process to reevaluate pre-tested parts of the software GUI in modified version of the software. The GUI test designer must regenerate test cases to target these common functionalities, and keeping track of such parts is an expensive and challenging process. So, usually in practice, no regression testing of GUI is performed. Many of GUI test cases from previous software testing process are unusable.

Commonly, a GUI test case contains a reachable initial state, a legal event sequence and expected states. The initial state is used to initialize the GUI to a desired state for specific test case and, an expected state is the state after specific event is executed. Therefore, a modification to the GUI can affect any of these parts and lead to useless of pre-designed test cases.

The GUI regression test cases can divide into two groups: *affected* test cases and *unaffected* test cases. Affected are test cases who should rerun but duo to modifications in GUI, they must design again. Unaffected are test cases that can execute exactly like original software GUI testing process but because they already executed in previous testing process, there is no need to test them again. These unaffected test cases are verified functionalities of the software GUI that do not change in the new version. As mentioned above, redesigning of affected test cases are expensive and challenging.

Memon [10] presents a method to perform GUI regression testing using AI Planner. He presents GUI test cases using *tasks* as pair of initial and goal states. These tasks remain valid in modified GUI, even changes to GUI cause test cases unusable. Each task represents a GUI's functionality. As a result, it is possible to generate affected test cases from these tasks automatically. Also, this technique uses a GUI model to automatically detect changes to the GUI and identify test cases that must rerun.

In this study a *Regression Tester* was designed to determine and regenerate affected test cases. The overview of this regression tester is shown in Figure 2.

One of the inputs is *Original test suits* that generated to test the original GUI. Other inputs are *representations* of original and modified GUIs. Regression Tester determined which test cases are affected, unaffected or must be discarded. Because discarded test cases verified functionalities that not further exist to modified software GUI, they must eliminate from testing process. *Test case selector* partitions the original test suits into (1) unaffected test cases, (2) obsolete tasks test cases, (3) illegal event sequence affected test cases and (4) incorrect expected states affected test cases. Illegal event sequence affected test cases are regenerated by *Planning-based test case regenerator.* But if planner failed to find a plan, the test case marks as discarded because it belongs to absolute tasks. *Expected-state regenerator* is used to regenerate expected state for incorrect expected state test cases and if it fails, test case will discard.

Consequently, this method performed regression testing based on re-planning affected test cases and associating a task with each test case and also create an interface between original and modified GUI to generate test cases. Furthermore, this method automate test case selection phase (the second phase of software testing phases) in regression GUI testing.

B. Selecting the Test Scenarios (Phase 2)

Test case selection is second phase in software testing process. Testers consider in effective test cases. Effective test cases can reveal the majority of software faults. According to [11], an effective test case should:

- ✓ Have a high probability of finding an error
- ✓ Not reevaluate tested sections
- ✓ Be the best of its breed
- ✓ Be neither too complex nor too simple

Each test case is defined by a set of inputs and expected output values. Basically, since the numbers of all test cases are very large in modern software, it is impossible to execute all of them in limited time and resources. Also, because many of test cases evaluate same section and part of the software, there is no need to execute all of them.

Therefore, testers must wisely select effective test cases with higher probability to finding faults. Likewise, if executing a test case does not report any faults, testers must not imagine the software is fault free and reliable. In fact, testers only waste their time in these situations.

So, this is very important to determine and select effective test cases. Automating this process can significantly decrease testing cost and increase testing quality. A good test case reduction approach

introduced in [12]. This research reveals that program's input- output analysis can identify which input attributes mostly affect the value of a specific output. It shows I/O analysis can significantly reduced the number of test cases. An Artificial Neural Networks (ANN) used to automating I/O analysis by identifying important attributes and ranking them. An ANN is a mathematical modeling of human neural networks that can learn from past experience using <input, output> pairs in a training phase and generate outputs for unknown inputs based on previous data. An ANN consists of layers - each layer represented by one or more processing unit called neurons- and connections between them. ANN's can learn by adjusting connections values in the network [6].

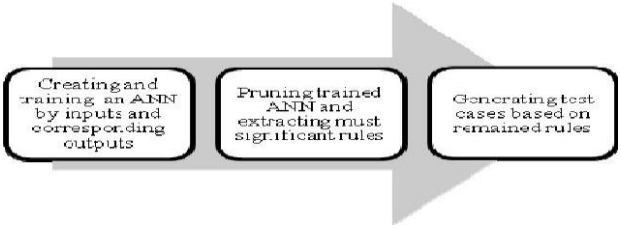

Figure 1. Automated test case generation and reduction.

This study modeled the software behavior using ANNs and identified which input has less effect on producing outputs by an ANN pruning algorithm. Pruning an ANN removes unnecessary connections between neurons but retaining significance ones. The removing process deletes unimportant inputs and also decreases the number of test cases. Finally, they generated test cases by remaining most significant inputs. Figure 1 depicts this process.

C. Running and Evaluating Test Scenarios (Phase 3)

As mentioned in section 2, evaluating test results in third phase of software testing phases required software's fault free output. Testers need a method to generate outputs of each input that uses in executed test cases. Then, they can compare this output with the test case execution output and if these outputs are not the same, a fault is detected. This is a place which testers need automatic testing Oracle. The Oracle is a fault free source of expected outputs. Non-automatic testing oracle can be a program specification or the developer knowledge of software's behavior [13]. An Oracle must accept every input specified in software's specification and should always generate a correct result.

Last and his colleges [7, 15] introduced a full automated black-box regression testing method using Info Fuzzy Network (IFN). IFN is an approach developed for knowledge discovery and data mining. The interactions between the input and the target attributes of any type (discrete and continuous) are represented by an information theoretic connectionist network. An IFN represents the functional requirement by an "oblivious" tree-like structure, where each input attribute is associated with a single layer and the leaf nodes corresponds to combinations of input value [7].

The structure of their method is shown in Figure 2. As can be seen in Figure 2, *Random Test Generator* provides test case inputs by means of *Specification of System Inputs*. These specifications contain information about system inputs such as data type and values domain. *Test Bed* executes these inputs on *Legacy Version* (Previous version of the software under test) and receives system outputs. Next, these test cases are used to train and model IFN as automated Oracle. Therefore, this Oracle can be used to detect faults in new software version. This method completely automated software testing's third phase in regression testing.

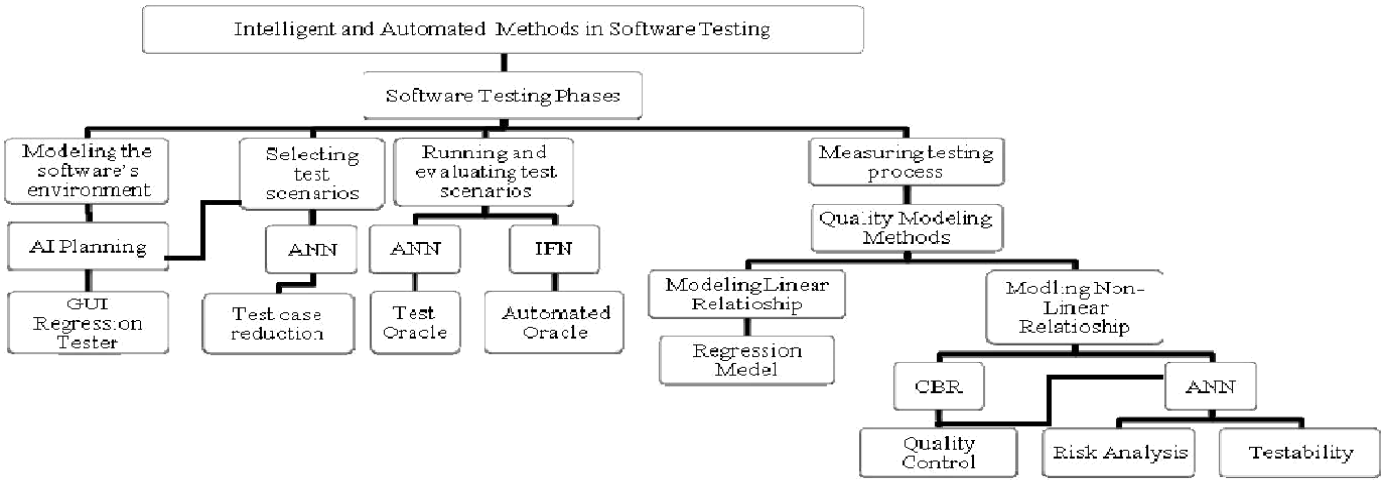

Figure 2. Classification of automated software testing methods.

IV. AI IMPACT ON SOFTWARE TESTING

Since computer's software applications rapidly increased in modern life, it is important to have enough reliability and minimizing the probability of faults in software products. Software testing is a process to find faults in software's products, due to increase software reliability. Because testing process is very costly, automation techniques are needed to reduce these costs and also, increase reliability [8]. In automated testing, the testing phases or part of them performed by intelligent methods, in order to reduce human role in the process [2].

Here is the table below to listing few e.g. between AI testing

Artificial Intelligence Testing	**Manual Testing**
1)Consumes less time	1)Consumes more time
2)Adapts to changes quickly	2)Need to go through training
3)Will need years to grow to full scale	3)Exists already

Figure 3. AI testing

V. ADVANTAGES AND DISADVANTAGES

While artificial testing has brought lot of accuracy and less consumption of time at the same time it has brought **disadvantages** along with it, here are a few which must be taken under consideration to take better steps to avoid the disaster as well as to make the best of out of what the phase of science is offering us.

While the advantages of Time consumption, Accuracy, human effort going down weigh's the artificial intelligence more, we should not forget that at the same time it is going to cut down millions of jobs worldwide and it is of course a software tool which is prone to hacks. There is no doubt that machines are learning fast, the computers can also generate reports on copious amounts of data. IT giants like Facebook make use of it through which machine analyses about which kind of data will be more interesting to the user and then it is punched in the form of news feed [4].

If we go by the other side of scenario there are still many who believe that manual testing can't go out of fashion as it gives the user high quality experience, no software has been made without bugs in it, manual testing will remain part of testing strategies as creativity opens many doors for users to enjoy and it definitely creates the market for software developing companies [5].

VI. MAIN IMPACT ON VARIOUS AREAS

The bots require very little maintenance and they are also capable of discovering new paths through the product by their own. The instant feedback mechanism is considered to be major impact as it gives an instant report to the developer or the tester which earlier usually would take a long time and a long procedure too. But while showcasing the impacts on positive side we should not also forget that artificial is definitely not human and we can never expect anything but a robotic decision only, one such e.g. comes from Sydney where in a café siege people were desperately trying to leave, as all of them tried booking Uber, the artificial intelligence or the so called algorithm activated price charge which is termed as price surge in this sector was really disheartening and non-human as what can be expected from artificial intelligence which did not bother to take consideration of the crisis people were trapped in. It may help testers to consume time but at the very same point it will give a chance or time for them to look with more creative ideas to rectify and create something new[6].

VII. ROLE OF ARTIFICIAL INTELLIGENCE IN ADAPTING

Machine learning bots are capable of helping with testing especially with end-user experience taking the front seat in testing. When trying to understand the role of bots in software testing, we need to bear in mind the fact that most applications have some similarity, i.e. size of a screen, shopping carts, search boxes and so forth. Bots can be trained to be specialists in a particular area of an app. AI bots can manage tens of thousands of test cases when compared to regression testing which can handle much lesser numbers. They know that AI could help to reduce the level of effort (LOE) while ensuring adherence to built-in standards.

AI bots can tap, type, and swipe through an app just like any living, breathing user, and by continuing to teach the AI how to take that data and apply it in an intelligent way, you have yourself an invaluable tool in a world of mobile applications that require rapid, agile solutions.

VIII. THE BRIDGE BETWEEN AI AND HUMAN TESTERS

But here's the critical piece of data for testers—and maybe more importantly, managers—to understand: AI can, and should, work alongside human testers. This isn't about replacing what we have. With AI, Arbon strongly believes we can make the testers we already have more effective. "More than 80 percent of testing is repetitive. You're often just checking that things work the same way they did yesterday.

This work is solvable by AI and automation," Arbon continued. "That other 20 percent of a tester's time today, the creative, questioning, reasoning part—that is what people should really be doing, and that rarely happens in today's fast moving and agile app teams." "While working along with artificial intelligence, testers in the near future will have a chance to focus on the most interesting and valued aspects of software testing." [3].

IX. CRITICS

While the impacts have been targeted negative by many critics, at the same time they are being looked through positive angle too, many entrepreneurs have stated that the artificial intelligence will make better developers and testers. One area could be seen is through learning different languages also. Artificial intelligence is expected to bring developers together and work better, artificial intelligence is can provide guidance or estimates wherever a complex problem occurs between two different variables and when there is lot of data also available from projects used earlier.

X. CONCLUSION

Summarizing the entire topic "Impact of Artificial intelligence on software testing" concludes with results to be awaited as this industry is yet to reach places, we do find a question over websites of submitting a unique code and to make the system believe we punch in the word to prove that we are not a robot, the difference between this check and test is one such small world of Artificial intelligence. The result is still awaited as there is not a report or index which show cases the jobs brought down by this tool, the results are still awaited as small scale industries have not touched the tools of artificial intelligence for software testing yet but as predictions are always welcome to avoid the abyss of darkness we must compromise the tools to be under our control as it is us who have the power of creativity,it is us who teach and it is us who make this system, consuming less time might lead us to more profit, accuracy might give us better results but in the end you do need a human touch to start things all over, let it be a test button only, after all this very article will be tested for plagiarism it will certainly be difficult for a human to do it by himself but at the very same time we need the same person to perform the activity of clicking the plagiarism button and submit the report. Impact may change the entire scenario of testing in coming days, but we will always have the upper hand.

In this paper, a classification of automated and intelligent methods has presented which can use in software testing phases. Each phase has introduced and explained based on how it can be totally or partially automated. The methods that used varied between AI methods like ANNs, CBR and AI planning, or statistical methods such as Regression Modeling and PCA. Some of the methods applicable in any type of test and some in special tests like regression testing. Each of these methods has limitations based on the tools they used. For example, ANN models of software cannot be accurate enough if software is non deterministic. Or IFN model can use if application is data oriented. In addition, testers must consider overhead costs of using these methods, and extra knowledge and specialist needed for developing such techniques. On the other hand, resent studies in comparing costs of using and not using these methods show that these automatic approaches have significant effect in reducing testing cost and increasing software quality.

Finally, because each method has affect in special type of test, elimination of human role in testing process cannot be complete yet. Consequently, more researches are needed in order to automate hole testing process.

XI. REFERENCES

[1]. Su, Y.-S. and Huang, C.-Y. Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. Journal of Systems and Software, 80, 4 2007), 606-615.Ding, W. and Marchionini, G. 1997 A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park.

[2]. Myers, G. J. The Art of Software Testing, Second Edition. Wiley, 2004.

[3]. Pressman, R. J. Software Engineering: A Practitioners Approach . Sixth Edition. McGraw-Hill 2005.

[4]. Whittaker, J. A. What is software testing? And why is it so hard? Software, IEEE, 17, 1 2000), 70-79.

[5]. Khoshgoftaar, T. M. and Seliya, N. Three-Group software quality classification modeling using an Automated Reasoning approach. World Scientific, City, 2004.

[6]. Fausett , L. Fundamentals of Neural Networks: Architecture , Algorithms and Applications. Prentice Hall 1994.

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

[8]. Smith, L. I. A tutorial on Principal Components Analysis. City, 2002.

[9]. Stockburger, D. W. Regression Models. Atomic Dog Publishing, City, 2001.

[10]. Memon , A. M. Automated GUI Regression Testing using AI Planning. World Scientific, City, 2004.

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

[12]. Saraph, P., Kandel, A. and Last, M. Test Case Generation and Reduction with Artificial Neural Networks. World Scientific, City, 2004.

[13]. Aggarwal , K. K., Singh, Y., Kaur , A. and Sangwan , O. P. A Neural Net based Approach to Test Oracle. ACM Software Engineering Notes2004).

[14]. Ye, M., Feng, B., Zhu, L. and Lin, Y. Neural networks based automated test oracle for software testing. Springer Verlag, Heidelberg, D-69121, Germany, City, 2006.

[15]. Last, M., Friendman, M. and Kandel, A. Using data mining for automated software testing. International Journal of Software Engineering and Knowledge Engineering, 14, 4 2004), 369-393.

[16]. Khoshgoftaar, T. M., Pandya, A. S. and More, H. B. A neural network approach for predicting software development faults. City, 1992.

[17]. Khoshgoftaar, T. M., Szabo, R. M. and Guasti, P. J. Exploring the behavior of neural network software quality models. Software Engineering Journal, 10, 3 1995, 89-96.

[18]. Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P. and Aud, S. J. A. A. S. J. Application of neural networks to software quality modeling of a very large telecommunications system. Neural Networks, IEEE Transactions on, 8, 4 1997, 902-909.

[19]. Khoshgoftaar, T. M., Allen, E. B. and Xu, Z. Predicting testability of program modules using a neural network. City, 2000.