Machine Learning-based Software Testing: Towards a Classification Framework

Mahdi Noorian[1], Ebrahim Bagheri[1,2], and Wheichang Du[1]
University of New Brunswick, Fredericton, Canada[1]
Athabasca University, Edmonton, Canada[2]
m.noorian@unb.ca, ebagheri@athabascau.ca, wdu@unb.ca

***Abstract*—Software Testing (ST) processes attempt to verify and validate the capability of a software system to meet its required attributes and functionality. As software systems become more complex, the need for automated software testing methods emerges. Machine Learning (ML) techniques have shown to be quite useful for this automation process. Various works have been presented in the junction of ML and ST areas. The lack of general guidelines for applying appropriate learning methods for software testing purposes is our major motivation in this current paper. In this paper, we introduce a classification framework which can help to systematically review research work in the ML and ST domains. The proposed framework dimensions are defined using major characteristics of existing software testing and machine learning methods. Our framework can be used to effectively construct a concrete set of guidelines for choosing the most appropriate learning method and applying it to a distinct stage of the software testing life-cycle for automation purposes.**

I. Introduction

Software Testing (ST) is an investigation process which attempts to validate and verify the alignment of a software system's attributes and functionality with its intended goals. Software testing is a labour intensive and costly process and as mentioned in [2], [3], a testing process may require up to 50% of the development resources. Due to this fact, automated testing approaches are desired to reduce this cost and time. Besides, automation can significantly enhance the testing process performance. Hence, for future software systems development, steps need to be taken towards the development of automated testing strategies [4].

Several interesting attempts have already been made for automating the software testing process. Machine Learning (ML) as a sub domain of AI [12] is widely used in various stages of the software development life-cycle [19], especially for automating software testing processes [5]. In [1], [17], evolutionary algorithms have been employed for automating test case generation. In [16], Artificial Neural Networks (ANN) have been used to build a model for estimating the effectiveness of the generated test cases. Briand *et al.* in [8] has proposed a method based on the C4.5 decision tree algorithm for predicting potential bugs in a software system and localizing the bugs in order to reduce the debugging process time. All research works show that the employment of machine learning techniques is a promising approach for automating testing processes. But, still some major questions remain that need to be addressed and further investigated such as:

- What types of machine learning methods can be effective in different aspects of software testing?
- What are the strengths and weaknesses of each learning method in the context of testing?
- How can we determine the proper type of learning method for the stages of a testing process?
- Where are the critical points in a software testing process in which ML can positively contribute?

The general purpose of our ongoing research is to provide a specific set of guidelines for automating software testing processes with the aid of machine leaning techniques. To come up with such set of guidelines, it is required to systematically analyse the current research work and find answers to some of the abovementioned questions.

In the first step, we propose a framework which can be used to classify the current research work in the conjunction of ML and ST. In addition, to support our framework, some works are reviewed. This classification framework can assist ST practitioners to analyse and understand the emerging applications of the ML-ST domain. Such structured classification framework can be useful for defining and constructing a set of guidelines for automating software testing processes.

The reminder of this paper is as follows. In Section II , the proposed classification framework is presented. In addition, the dimensions of the framework are discussed in detail. Section III is devoted to presenting some work in the area of ML and ST. In Section IV we discuss how this set of representative work can be classified according to our proposed framework. We conclude the paper with conclusions and direction for future work in Section V.

II. The Proposed Framework for Classification

During the past decade, several works have been introduced based on machine learning techniques where learning algorithms have been applied to the various stages of software testing. In order to clarify the stand point of ML in the area of ST, we propose a classification framework. In this section, we take a look at the proposed framework and its dimensions.

A. The Framework's Dimensions

The most important factors that can be used to distinguish the current research work are represented in Fig. 1. In its highest layer, the framework consists of two main categories

TABLE I
TESTING GENERAL ACTIVITY DIMENTION

Sub-dimension	Possible value
A: Test Planning	Testing cost estimation [9]
B: Test Case Management	Test case prioritization [1] Test case design [1] Test case refinement [6] Test case evaluation [16]
C: Debugging	Fault localization [18] Bug prioritization [18] Fault prediction [15]

(Testing and ML). According to the defined categories, six main dimensions and some sub-dimensions are obtainable in the classification framework. The framework's dimensions are as follows:

Dimension 1-Testing approach: According to the older testing terminology [2], we define the *testing approach* dimension with three possible values: black-box, white-box, and gray-box [2]. Based on the black-box approach, testing can be performed using the external description of a software system such as the software specification. In a white-box approach, the internal properties of a software system like source code can be used for testing purposes. The grey-box approach is a combination of the two, which considers both internal and external properties at the same time.

Dimension 2-Testing general activity: In the second dimension, the standard testing process life-cycle [13] inspired us to define the *a) test planning*, *b) test case management*, and *c) debugging* sub-dimensions. With respect to the testing process life-cycle, we found out that these three phases are critical and that automation can effectively assist them in order to reduce the cost and time of the whole testing process. The possible activities that can be automated based on ML in *a*, *b*, and *c* are presented in Table I.

In the *test planning* sub-dimension, *testing cost estimation* can help test managers to predict testing process cost and time and provide good testing plan to manage the testing process efficiently. *Test case management* includes several tasks such as *test case prioritization*, which intends to prioritize the test input space in terms of test case effectiveness; *test case design*, which intends to generate high quality test cases; *test case refinement*, which intends to map the current specification of a software system to the existing test cases in order to reuse the available test cases; and *test case evaluation* which intends to measure the quality of the generated test cases. In the *Debugging* sub-dimension, *fault localization* can help to find the exact location of the program that is defected. In addition, *bug prioritization* intends to prioritize the revealed faults based on their severities; later test engineer can focus on more critical faults accordingly. *Fault prediction* can assist test engineers in the debugging stage, in the sense that potential faults for a given program are predicted.

Dimension 3-Testing level: Software development process includes a range of stages form "requirement analysis" to "implementation" [13]. The development process goes further even after the implementation stage and concentrates on software maintenance. In Dimension 3, we have employed the following widely accepted testing levels in order to classify existing work: *acceptance testing*, *system testing*, *integration testing*, *module testing*, *unit testing*, and *regression testing*, which refer to requirement analysis, architectural design, subsystem design, detailed design, implementation, and maintenance in the software development stages, respectively.

Dimension 4-Learning technique: In the machine learning area, various types of learning methods have been introduced and each of them has its own specific characteristics. We adopt Mitchel [12] learning method classification and define *learning technique* dimension values as follows: Decision Trees (DT), Artificial Neural Networks (ANN), Genetic Algorithms (GA), Bayesian Learning (BL), Instance Base Learning (IBL), Clustering, and Hybrid methods, which can be combination of several other learning methods.

Dimension 5-Learning property: Each learning method has its own specific characteristics. This dimension is devoted to evaluating learning methods properties in various aspects. To do so, three sub-dimensions; *Training data properties*, *Supervision*, *Time generalization* are defined. The *Training data properties* evaluate the properties of a data set. The gathered data for learning process can be small or large in terms of its quantity. Data can be noisy or accurate in terms of errors that might exist in a data set. In addition, learning can be supervised or unsupervised; therefore, the *Supervision* sub-dimension is defined. For a given target function the *time generalization* sub-dimension exposes how its generalization is; which can be either eager (at learning phase) or lazy (at classification phase). In terms of online and offline learning, also *time generalization* shows how a learning system updates its approximation of the target function after the initial training data are used. The last sub-dimension, *automation degree*, is responsible for addressing the learning system in terms of the degree of automation. The designed system can be fully or partially automatic.

Dimension 6-Elements learned: In each learning system for software testing automation, various types of data can be used to build the target function. The training data can be collected in different stages of the software testing process or software development life-cycle. The learning elements could be *software metrics*, *software specification*, *CFG (control flow graph)*, *call graph*, *test case*, *execution data*, *failure reports*, and/or *coverage data*. The *elements learned* dimension is defined to distinguish the type of learning element that being used for an individual learning model in the testing process.

Fig. 1 shows the framework dimensions, their sub-dimensions and some possible values for them. The main dimensions are shown with ellipse and numbered from 1 to 6. The sub-dimensions are represented with ellipse as well. In addition, the possible values of each dimension are shown with rectangles.

III. THE EXAMPLES OF ML IN SOFTWARE TESTING

In this section, some research work in the area of ML and ST has been selected for supporting the proposed classification framework. In Section IV, we discuss how this work can be classified with proposed framework. Note that, Dimension

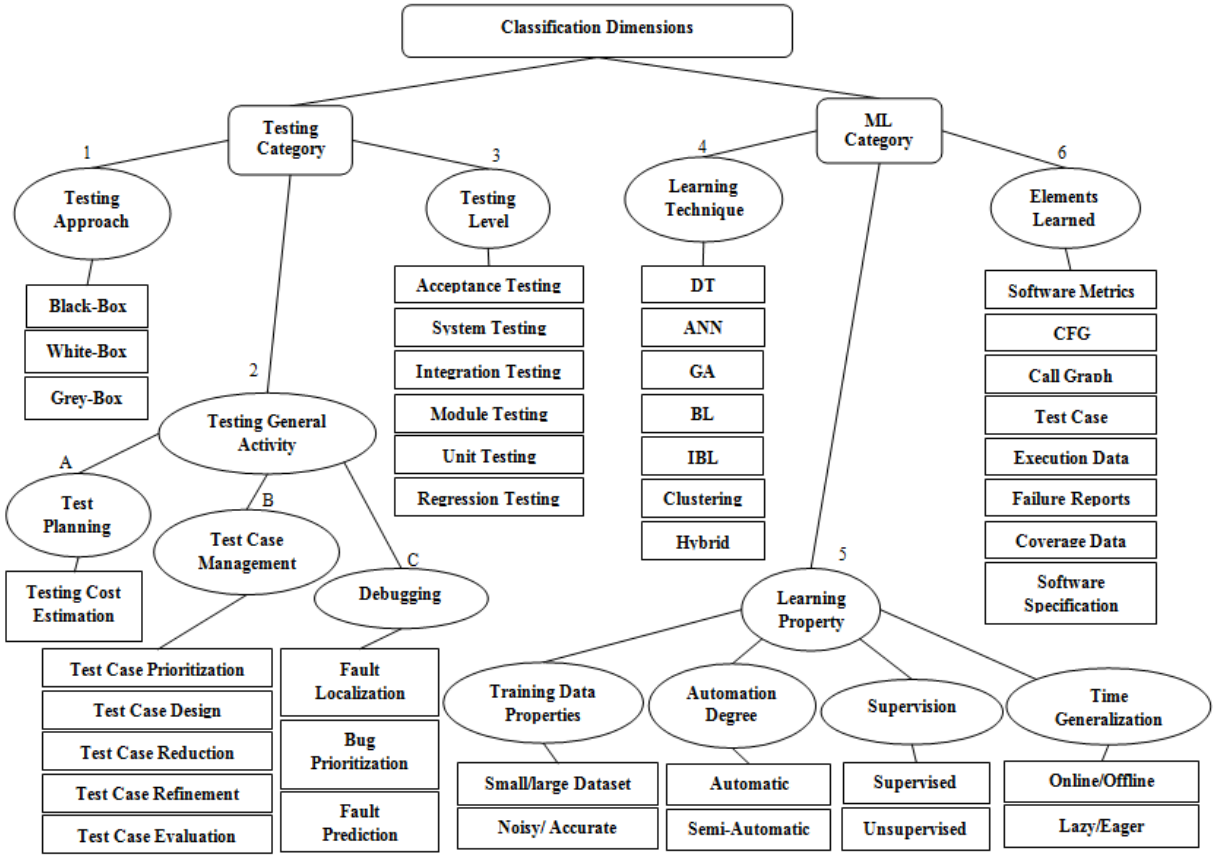

Fig. 1. Classification framework dimensions.

2 has been chosen as the main dimension for classification purposes. We have tried to select and present at least one work according to sub-dimensions *a*, *b*, and *c*. In the following three subsections, we have chosen and briefly introduced three interesting works (due to space limitation) that have employed machine learning techniques for test planning, test case management, and debugging, respectively. Later, we will show how these three works that fall under different aspects of software testing can be effectively described using our proposed classification framework. In our future longer publication, we will comprehensively classify the relevant related literature within our proposed framework. The following three works are only introduced to show the potential of our framework.

A. Test planning

In [9], machine learning techniques have been used to identify the main factors of software testing where they have been employed to predict the testing process time. Understanding the important factors of software under the test can help test managers to prepare a good test plan. The developed methodology consists of four phases; 1) Database formation; 2) Data collection; 3) Classification of software and 4) Analysing the results. In the first phase, databases of 25 software systems were constructed. The sample software were collected from various sources with different attributes. The second phase is devoted to real data extraction form software systems. According to the predefined factors and for a given software system, values were extracted for each individual factor. The list of determined factors included code complexity measures, measures of programmer and tester experience, testing time and a number of other attributes. The complete attributes list includes twenty-seven attributes. In the third phase, COBWEB/3 [11] was used to classify the software systems and build a model to predict the cost of testing for the new software system. COBWEB/3 builds a decision tree. In the last phase, the classification results need to be analyzed to get specific time prediction.

B. Test Case Management

In the context of software evolution, sometimes the available test cases need to be refined to address the current test requirements of the software system. The process of test case refinement is called *test case re-engineering* [6]. In [6], [7], a semi-automatic methodology based on a machine learning approach is provided to explore the limitations and find the possible redundancies of the available test cases and then refine them for future usage. As illustrated in Fig. 2, the MELBA (MachinE Learning based refinement of BlAck-box test specification) methodology is an iterative process and it consists of five main activities.

In the Activity 1, test cases are transformed into the abstract level using the Category-Partition (CP) strategy [14], so they would be ready to be used by the machine learning method in Activity 2. The CP method helps to model the input domain

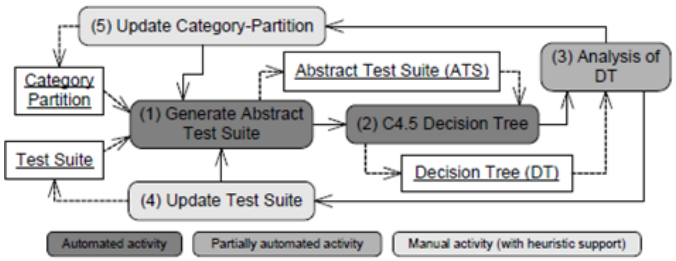

Fig. 2. Overview of the MELBA process [6].

and as discussed in [14], CP requires defining the sets of categories and choices. The categories are the properties of the input and environment parameters of the software system and the choices are the possible values for each category. The abstraction process is conducted using the defined categories and choices. An abstract test case shows an output equivalence class and the pairs of (category, choice). In Activity 2, the machine learning algorithm (C4.5 decision tree) is used to learn the rules for classifying the abstracted test cases. Activity 3 is the important part of MELBA, which is devoted to analysing the learnt rules. The results of the analyzed tree could lead to determine the problems that cause the redundant test cases or the needs for adding new test cases (Activity 4). In addition, the learnt rules may indicate that CP needs to be updated (Activity 5). This iterative process will continue until no more problems could be identified in the trees.

C. Debugging

In software debugging, fault localization refers to process of finding the exact locations of the program that are defected and contain faults. In [18], the Back-Propagation (BP) neural network method [10] is used to effectively pinpoint the program faults. According to proposed approach, first the BP neural network is trained with both the coverage data and the execution result, then the trained network is used to predict and rank the suspicious statements of the code, in terms of its potential to have faults. The proposed method for fault localization consists of the following steps:

1) Building up the dataset: In order to create dataset, it is required to run the program with several test cases and collect the coverage data for each execution. Beside this, the execution results are also need to be collected. This coverage data (coverage vectors) and its execution results can be used for training phase.

2) Training of the BP neural network: In this stage BP neural network is built with k input-layer neurons, three hidden-layer neurons, and one output-layer neuron. The sigmoid function is used as the transfer function. In order to perform training process, coverage vector of each test cases $t_1...t_n$ is used as input data and their corresponding $r_1...r_n$ testing result is used as expected output.

3) Prioritizing the defected statements in program: the trained network can be used for prioritization purpose. Assume, we have a set of virtual test cases $v_1...v_n$ which they cover statements $s_1...s_n$ respectively. It means that, the execution of virtual test case v_i *(i=1...m)* covers only s_i statement. With respect to this assumption, we can say that if the execution of test case v_i fails, then the probability that s_i is defected is high. Here, the trained network can be used to predict the expected output for each virtual test case (r_{vi}). The value of r_{vi} is between 0 and 1. The larger the value of r_{vi}, the more probable s_i is defected. Hence, the statements $s_1...s_m$ can be ranked based on $r_{v1}...r_{vm}$ values in descending order. At the end, the bugs can be identified by examining statements one by one form top of the list.

IV. Classification based on Proposed Framework

In Section II, we introduced a classification framework. There, we discussed the framework's dimensions, sub-dimensions, and some possible values of them. This framework can be used to classify and highlight the main features that need to be considered in the junction of ML and ST area. Following that, in Section III, we presented some representative work in the ML and ST domains.

In this section, we show how the proposed framework can be applied on the three sample presented works. Table II summarizes the results of this classification for each individual work that we have reviewed. In this table, the columns show the dimensions and the related sub-dimensions of the framework and the rows represent the research work that we reviewed. In addition, each cell in the junction of a column and a row indicates the value(s) of that dimension for the corresponding research work.

To clarify, from Table II it can be understood that the automation in [18] has been done in the *debugging* stage of testing process and the *task type* was *fault localization*, which *automatically* finds the suspicious statements in the source codes. In addition, in terms of testing level, this work has been conducted at the *unit testing* level. With respect to the *testing approach* dimension, this work was carried out based on the *white-box* approach, because the statements of the source code are considered for testing purposes. From the ML perspective it can be derived that this work has benefited from the *ANN* learning algorithm which is a *supervised* learning method. Furthermore, the elements which were used as input in the learning process were *coverage data*.

For the work presented in [9], automation has been performed in the *test planning* stage whose *task type* is *testing cost estimation*. The *DT (Decision Tree)* type learning algorithm was used to build a model for estimating testing process cost. From the *automation degree* point of view, this work proposed a *semi-automatic approach*, since in some parts of this method we can see the need for test engineer's intervention. Furthermore, *software metrics* have been employed as input for the learning process. In this work, the learning task has been conducted using a *small size* of dataset, which can considerably decrease the learning process time.

Finally, in [6], [7], *testing general activity* is addressed by *test case management* value, which means, for automation purposes the learning method was applied in the *test management* stage in the software testing life-cycle. The *task type* in this work was *test case refinement*. The proposed method in

TABLE II
CLASSIFICATION OF SOME RESEARCH WORK IN ML AND ST BASED ON PROPOSED FRAMEWORK

Research Work	Testing Category				ML Category					
	Testing General Activity	*Task Type*	*Testing Level*	*Testing Approach*	*Learning Technique*	*Learning Property*				*Elements Learned*
						Training data properties	*Automation Degree*	*Supervision*	*Time generalization*	
[9]	Test planning	Testing cost prediction	-	-	DT	Small dataset	Semi-automated	Unsupervised	Offline/Eager	Software metric
[6][7]	Test case management	Test case refinement	Regression testing	Black-box	DT	Small dataset	Semi-automated	Supervised	Offline/Eager	Test Case/software specification
[18]	Debugging	Fault localization	Unit testing	White-box	ANN	-	Automatic	Supervised	Offline/Eager	Coverage data

this work, in terms of testing level, can be categorised in the *regression testing* level, because this work focused on reusing the existing test cases for new updated version of the current software system. With regards to the *testing approach*, this work used the *black-box* method, since it considers only the software specification for testing purposes. From the machine leaning point of view, this work employed a *semi-automatic* approach using *DT*- based learning technique. The dataset size was quite small and the elements which have been used to build dataset were *test cases* and *software specification documents*.

Table II and the above classification show how our proposed framework is useful in categorizing and explaining work in the intersection of ML and ST.

V. Conclusion and Future work

Testing is critical task in the software development process and considerably imposes cost and time restrictions on the development process. Therefore, automating the testing process can significantly increase testing process performance. In the context of software testing, different types of data can be collected in various stages of testing. Machine learning techniques can be used to find patterns in this data and use them for automation purposes.

In this paper, we have introduced a framework for classifying the current research works in ML and ST. Some sample works of ML in ST have been reviewed and appropriately classified according to the proposed framework. The proposed framework provides us with the opportunity to systematically investigate and extract the prominent information from existing research works in ML and ST.

Considering the introduced framework, our main attempt for future research is reviewing and classifying all current work in the area of ML and ST. The collected information from classification process will assist us to construct the sets of concrete guidelines for applying ML methods in the ST process. These guidelines will help test engineers to choose the most efficient and appropriate learning method for automation of a target testing stage. Our initial probe has shown that our classification framework is quite strong in providing the means to capture various aspects of work in ML and ST.

References

[1] Moataz A. Ahmed and Irman Hermadi. Ga-based multiple paths test data generator. *Comput. Oper. Res.*, 35:3107–3124, October 2008.

[2] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2008.

[3] Boris Beizer. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[4] Antonia Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, FOSE '07, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society.

[5] Lionel C. Briand. Novel applications of machine learning in software testing. *Quality Software, International Conference on*, 0:3–10, 2008.

[6] Lionel C. Briand, Yvan Labiche, and Zaheer Bawar. Using machine learning to refine black-box test specifications and test suites. In *Proceedings of the 2008 The Eighth International Conference on Quality Software*, Washington, DC, USA, 2008. IEEE Computer Society.

[7] Lionel C. Briand, Yvan Labiche, Zaheer Bawar, and Nadia Traldi Spido. Using machine learning to refine category-partition test specifications and test suites. *Inf. Softw. Technol.*, 51, November 2009.

[8] Lionel C. Briand, Yvan Labiche, and Xuetao Liu. Using machine learning to support debugging with tarantula. In *Proceedings of the The 18th IEEE International Symposium on Software Reliability*, ISSRE '07, pages 137–146, Washington, DC, USA, 2007. IEEE Computer Society.

[9] Thomas J. Cheatham, Jungsoon P. Yoo, and Nancy J. Wahl. Software testing: a machine learning experiment. In *Proceedings of the 1995 ACM 23rd annual conference on Computer science*, CSC '95, pages 135–141, New York, NY, USA, 1995. ACM.

[10] Laurene Fausett, editor. *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[11] Kathi Een Mckumcr, Kevin Thompson, Uncl As, Kathleen Mckusick, and Kevin Thompson. Cobweb/3: A portable implementation, 1990.

[12] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[13] Glenford J. Myers and Corey Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004.

[14] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating fuctional tests. *Commun. ACM*, 31:676–686, June 1988.

[15] Susan A. Sherer. Software fault prediction. *Journal of Systems and Software*, 29(2):97 – 105, 1995.

[16] A. von Mayrhauser, C. Anderson, and R. Mraz. Using a neural network to predict test case effectiveness. In *Aerospace Applications Conference, 1995. Proceedings., 1995 IEEE*, number 0, pages 77 –91 vol.2, February 1995.

[17] Joachim Wegener, Andre Baresel, and Harmen Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43(14):841 – 854, 2001.

[18] W. Eric Wong and Yu Qi. Bp neural network-based effective fault localization. *International Journal of Software Engineering and Knowledge Engineering*, 19(4):573–597, 2009.

[19] Du Zhang and Jeffrey Tsai. Machine learning and software engineering. *Software Quality Journal*, 11:87–119, 2003. 10.1023/A:1023760326768.