ISSN: 1803-3814 (Printed), 2571-3701 (Online)
https://doi.org/10.13164/mendel.2021.2.068

MENDEL Soft Computing Journal

Robotic Automation of Software Testing From a Machine Learning Viewpoint

Vinod Yadav✉, Raphael Kwaku Botchway, Roman Senkerik, Zuzana Kominkova Oplatkova

Department of Informatics and Artificial Intelligence, Tomas Bata University in Zlin, Zlin, Czech Republic
vyadav@utb.cz✉, botchway@utb.cz, senkerik@utb.cz, oplatkova@utb.cz, ailab@fai.utb.cz

Abstract

The need to scale software test automation while managing the test automation process within a reasonable time frame remains a crucial challenge for software development teams (DevOps). Unlike hardware, the software cannot wear out but can fail to satisfy the functional requirements it is supposed to meet due to the defects observed during system operation. In this era of big data, DevOps teams can deliver better and efficient code by utilizing machine learning (ML) to scan their new codes and identify test coverage gaps. While still in its infancy, the inclusion of ML in software testing is a reality and requirement for coming industry demands. This study introduces the prospects of robot testing and machine learning to manage the test automation process to guarantee software reliability and quality within a reasonable timeframe. Although this paper does not provide any particular demonstration of ML-based technique and numerical results from ML-based algorithms, it describes the motivation, possibilities, tools, components, and examples required for understanding and implementing the robot test automation process approach.

Keywords: *Automation, Big Data, Machine Learning, Robotic Software Testing, Test Automation, Software Reliability.*

Received: 14 November 2021
Accepted: 13 December 2021
Published: 20 December 2021

1 Introduction

Software testing is considered a critical activity that must be performed to guarantee the reliability of every software. This testing process enables the software to be executed under clearly defined conditions to detect errors and facilitate software evaluation. Although a higher number of test cases negatively influences the testing process in terms of cost, minimizing the quantity of these test cases while maintaining software reliability remains a key challenge during software testing. This paper focuses on machine learning (ML)-based automated testing (robot automated testing) [8]. In recent years, there has been a remarkable increase in the need for high-quality software. As a result, testing-related difficulties are becoming increasingly important. The ability to assess software fault-proneness is critical for lowering costs and increasing the overall effectiveness of the testing process [6]. Software engineering's primary purpose is to develop high-quality software. In this regard, software testing is regarded as a crucial component of the software development process. Its primary purpose is to expose flaws by running "excellent test scenarios." An excellent test case is the one that has a high chance of revealing a flaw that has yet to be discovered. To meet the testing goals with the least amount of work and expense, testing approaches and criteria have been offered. Testing criteria are predicates that must be met, and they are often developed using one of three techniques: functional, structural (control and data flow-based criteria), or fault-based (mutation-based testing). These criteria consider a variety of factors while generating test results and can identify a variety of flaws. As a result, a testing approach should combine the criteria in a complementary manner, and the usage of cost-cutting techniques is critical [9]. Software testing is an examination procedure that aims to examine and verify that the properties and functioning of a software system are aligned with its intended goals. Several interesting attempts to automate the software testing process have previously been made. Machine Learning (ML), a subdomain of artificial intelligence, is frequently utilized in many stages of the software development life cycle, particularly to automate software testing processes. [11]. Whenever software developers encounter problems while working on their machines, they create a new branch, fix the problem, commit, push the code with the newly created unit and finally merge the updated code into a master branch for deployment into production. After that, the newly formed branch gets deleted. As developers work on their local machines independently, managing their work becomes difficult. To resolve this challenge, we used a cloud-native application framework called OpenShift. We created dynamic test cases using a robot framework and ML to provide more efficient tests. This is because we can test only a few test cases statically; however, using robot framework and ML, we can have multiple dynamic generated test cases. We can also run automation or robot

testing for message protocols, SSH, HTTP Rest endpoints, JMS, SoapUI API, FTP, XML, JSON data, etc. [13]. In the test scenario, messages that are sent and received are predefined. The message flow is specifically defined by a tester for use cases. Automation Testing refers to using testing tools that avoid manual intervention to discover defects in software. For software testing, several testing frameworks are available, with the most common being Data-Driven Testing Framework (DDTF) and the Test-Driven Framework Development (TDFD) [1]. The tester defines a message flow the use cases will use. The literature review is next to this introduction, followed by a snapshot of robot testing combined with machine learning and our methodology. The discussion and conclusion section brings closure to our work.

1.1 Motivation

The motivation behind this presented research and the originality can be formulated as follows. This paper summarizes the most recent research at the intersection of software testing using machine learning, including the most researched themes, the strength of evidence for, and the benefits and limitations of ML algorithms. We anticipate that the outcomes of this systematic mapping will enable researchers to develop more effective ML-based testing methodologies because these research efforts will be based on the most up-to-date information.

This paper represents an extension of preliminary work presented at IEEE ICECET conference[1]. This paper expands the preliminary concepts and theoretical frameworks, related works, presents more code examples and in-depth conclusions.

1.2 Robotic Testing Framework

Robotic Testing Framework (RTF) is a free-to-use online automation testing framework for acquiring, testing and obtaining test-driven based development [2]. It uses many types of tests: keyword, behaviour, and data-driven for writing test cases. Robot Framework (RF) offers automation and database tests via its external libraries. It is considered a generic open-source (including libraries and tools in the ecosystem) automation framework released under Apache License 2.0.

RF performs test automation and robotic process automation (RPA). It is dynamically supported, with many industry-ruling companies using it in their software development. RF is open and extensible, can be merged with any tool to build robust and adaptable automation solutions. Being open-source also means it is free to use with no licensing costs. It has several external libraries such as database, collections, JSON, string, etc. which contains utilities meant for the Robot Framework's usage. It allows us to fire all types of SQL queries in our database library after initiating an action to verify the results. RF-based applications are independent of the operating system.

RTF is using both JPython (JVM) and Iron Python (.Net) with Python as the main framework. First, we need to install all of the essential libraries for robot automation and database testing; after that, we must add them to the robot's setting components and begin the database connection. After connecting the database, we would have access to write all SQL queries in test cases as needed and then develop scripts for machine learning testing. Algorithm 1 shows an example of database connectivity using the robot framework and sample SQL queries.

Algorithm 1 RTF Example

```
* Settings *:
Library RequestsLibrary
Library String
Library JSONLibrary
Library DatabaseLibrary
* Variable *
&{headers}= Content-Type=application/json Authorization=Basic ABCDEF==
* Test Cases *
Sample Database Test (Name of the test case) [Documentation] Employee Details Test.
connect To Database Using Custom Params pymysql database='employee', user='employee', password='employee', host='localhost', port=3305 (database configurations)
${table} = Execute SQL String create table employee (id integer primary key, name varchar (50), address varchar (50)) (SQL query to create a table)
Log ${table}
Should Be Equal As Strings ${table} None
${db_EmployeeId} = Query SELECT id FROM employee WHERE department_id = 101;(Select query)
Log ${ db_EmployeeId } (logging the result retrieving from select query)
Delete All Rows from Table Employee (delete employee from database)
```

1.3 Machine Learning for Robot Test Automation Process

ML is a sub-field of Artificial Intelligence (AI) with the sole aim of emulating data learning activities [12]. It provides methods of identifying current and acquiring future knowledge to improve and realize self-perfection. Before it is possible to describe how ML can support the phases of the robot test automation process, it is essential to understand the basic problem of the aforementioned process. It is crucial to understand why test automation is unstable in the absence of ML. For example, non-ML test data are static. The data are

[1] Yadav, V., Botchway, K. R., Senkerik, R., and Oplatkova, K. Z. Robot Testing from a Machine Learning Perspective. In ICECET 2021 International Conference on Electrical, Computer and Energy Technologies, In press

unable to automatically adjust to modifications that are depending on test results. The following features can be added to test data with ML: automatic scanning the new code, analyzing security issues and identifing test coverage gaps. ML algorithms used for tasks such as decision-making automatically validate and compare the latest datasets based on predefined earlier datasets. The section below demonstrates some real-time applications of robot framework (RF).

In the test scenarios considered here (referred as "our"), software engineers often run their test cases from the beginning. Every time a build is finalized, which, of course, takes time and effort. As a result of adopting this trend, the system will become intelligent. If a developer conducts a test, the results and scenarios will be transferred to other nodes, allowing them to benefit and learn from the outcomes. Furthermore, considering that machine learning is not a panacea for all software-testing challenges, we believe that this article is an important first step toward bringing machine learning to software testing. In essence, the findings of this work have the potential to help practitioners and researchers make better decisions about which machine learning algorithms are most suited to their needs.

Regarding the issue of forecasting the efficacy of test cases and dealing with its difficulties, we need data to figure out how an effective test case looks like. We make the premise that some effective test cases will be present in the collected data set of inputs for a program whose run-time behavior is also recorded, even if we do not know how to come up with one. It is possible to generate predictions based on the outcomes of an ML algorithm if it can learn from the available test-case data and provided that the program under test did not differ significantly from the version used during data collecting. Although the ML algorithm may not be able to discover all aspects of the test-case assessment process, it can find some underlying structures and patterns in the data. In this case, the algorithm's output is an approximation (i.e., a model). In a broad sense, machine learning algorithms process data to create models. The emerged models contain patterns that allow us to draw conclusions and better identify situations, such as forecasting the effectiveness of test cases [4].

2 Workflow Design

This section describes related works that motivated our presented research and the tools, components, and code examples[2] required for understanding and implementing the described approach. In [8], the authors proposed a software testing framework with Generative Adversarial Networks (GAN) [7], which generates test cases for the software under test to enhance test coverage. GANs use two neural networks, pitting one against the other to learn the underlying distribution of the training data to generate new data instances that resemble training data. GAN consists of a generator and a discriminator. The generator takes random noise and generates fake data. The discriminator checks whether the generated data is accurate or false. The GANs estimate generative models using an adversarial scheme. The main idea in [8] was to utilize the test inputs and the corresponding execution path as inputs of GAN. Paper [5] describes implementing an automation testing framework for testing web applications via the Selenium WebDriver tool. The authors argued that their tool helped in dynamically changing web applications. Since the overall concept is relatively new, it is necessary to introduce the components of a possible ML-based approach. Fig. 1 shows all of the different types of testing that may be done; further, the following subsections contain implementation examples.

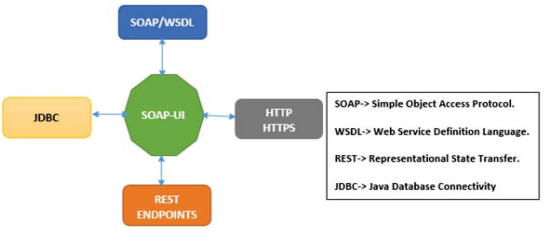

Figure 1: SoapUI supported protocols

2.1 SoapUI

SoapUI is an accessible, open-source, and cross-platform API testing tool [10]. A calm-to-use graphical user interface and enterprise-class qualities allow you to quickly and rapidly build and execute automated practical, regression, compliance, and load tests. It is a functional API testing tool that runs non-functional testing such as performance and security tests. Since the Robot Automation test is different, Robot testing provides functionalities such as SoapUI API test and database test. It can improve the quality of code (clean code) and bug-free application if the requirements are properly defined and managed and the right tool for the correct application matches the application's needs. Please refer to Algorithm 2, for a real time example of SOAP/WSDL with source code.

2.2 REST Endpoints

Rest endpoints testing involves evaluating all rest APIs and requests such as GET, POST, PUT, and DELETE with the proper response status and response body. Algorithm 3 shows an example of calling REST endpoint by GET method and logging the results, whereas Algorithm 4 represents the calling REST endpoint by POST method and comparing the results.

[2]Due to the formatting of algorithm environment here, the codes are not ready for compiling, since double and more special spacing syntax requested by framework was omitted by Journal template. Please contact the corresponding Authors

Algorithm 2 Real time example of SOAP/WSDL

```
* Settings *:
Library SoapLibrary
Library Collections
Library Operating System
Library XML
* Variable *:
${ClientUrl} http://endpoint.com/sampletest.asmx?wsdl
* Test Cases *:
Test read the WSDL file
Create Soap Client ${ClientUrl}
${response} Call SOAP Method With XML request.xml
${result} Get Data From XML By Tag ${response} result
Log ${ result }
```

2.3 HTTP and HTTPS

Requests Library is a Robot Framework module that wraps the well-known Python Requests Library to enable HTTP and HTTPS API testing functionality. Algorithm 5 shows the calling http and https method and comparing the results.

Algorithm 3 Calling REST endpoint by GET method and logging the results

```
* Settings *:
Library RequestsLibrary
Library String
Library JSONLibrary
Library DatabaseLibrary
* Variable *
&{headers}= Content-Type=application/json Authorization=Basic ABCDEF==
${r_Id}= 101
${r_Name}= AXY
${r_Address}= TGM Zlin CZ
* Test Cases *:
${response}= GET http://endpoint.com/sampletest expected_status=200 ${headers}
Should Be Equal ${result} ${ response.text}
Log SampleTest Body is: ${response.text}
```

3 Methodology Design

When the robot automated software testing is executed, it is crucial to create a list of requirements of test cases. The Robot Framework and Machine Learning are the two main tools discussed in this research paper. Recently, several companies have been utilizing robot automated testing tools for various applications because of their advantages compared to other testing tools. This field of applied computer science presents five test automation tools used by other researchers in

Algorithm 4 Calling REST endpoint by POST method and comparing the results

```
* Settings *:
Library RequestsLibrary
Library String
Library JSONLibrary
Library DatabaseLibrary
* Variable *:
&{headers}= Content-Type=application/json Authorization=Basic ABCDEF==
${data}= {"id": "101", "name": "AXY", "address": "TGM Zlin CZ"}
${result}= OK
* Test Cases *:
${added_employee_response}= POST http://endpoint.com/addemployee expected_status=200 ${headers} ${data}
${employee_response_body} convert to string ${added_employee_response.text}
Log ${employee_response_body}
${response}= Evaluate json.loads("""${employee_response_body}""") json
$id }= Get Value From Json ${ employee_response_body $..id
${name}= Get Value From Json ${ employee_response_body } $..name
${address}= Get Value From Json ${ employee_response_body } $.. address
Should Be Equal ${id} ${r_Id}
Should Be Equal ${name} ${r_Name}
Should Be Equal ${address} ${r_Address}
```

Algorithm 5 Calling http and https method and comparing the results

```
* Settings *:
Library RequestsLibrary
Library String
Library JSONLibrary
Library DatabaseLibrary
Library OperatingSystem
Library Collections
* Variable *:
&{headers}= Content-Type=application/json Authorization=Basic ABCDEF==
${result}= OK
${results}= OK
* Test Cases *:
${http_response}= GET http://endpoint.com/httptest expected_status=200 ${headers}
${https_responses}= GET https://endpoint.com/httpstest expected_status=200 ${headers}
Should Be Equal ${result} ${http_response.text}
Should Be Equal ${results} ${https_response.text}
Log SampleTest Body is: ${ http_response.text}
Log SampleTest Body is: ${ https_response.text}
```

the last few years. Fig. 1 shows the possible types of tests done using the SoapUI API in our work. Fig. 2 depicts the developed and presented pipeline concept. Whenever we encounter any problem in our test case, our robot test directly points out and displays the location and error message on our HTML page (please, see Fig. 3).

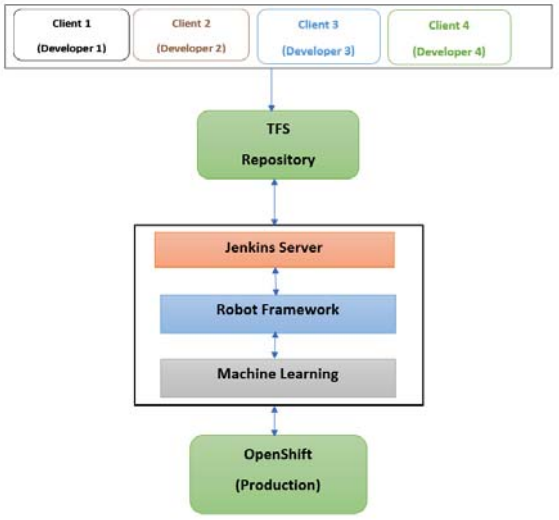

Figure 2: Conceptual study framework

Figure 3: Sample test log and test execution log

3.1 Team Foundation Server

Team Foundation Server (TFS) is a product of Microsoft, now called Azure DevOps Server. The lightweight operations actualized by the tools and environment can authorize development teams to achieve the target without affecting productivity. It is somehow similar to GitLab, where we create as many repositories as we want and integrate them with production into OpenShift.

3.2 Jenkins Server

Jenkins server, written in Java is a free-to-use online automation tool with plugins designed for integration purposes [11]. Jenkins obtained a build for a single exchange made in the TFS repository. Once the code is ready, we immediately deploy and test it on the testing server. Accordingly, development teams are given updates on build and test results.

3.3 OpenShift

The company Red Hat develops OpenShift. It is a family of containerization software products. The OpenShift has developer-oriented views which are oriented around working with application resources within a namespace. It also provides a Command Line Interface (CLI) that supports a superset of the actions similarly as provided by Kubernetes CLI.

3.4 ML in Test Automation

There are many ways of implementing ML-based techniques within the designed pipeline. For example, decision-making algorithms automatically validate and compare specific releases based on predetermined data sets and acceptance criteria. Regression algorithms, like classification algorithms, learn from past data and provide us with a value as an output. Future research aims to implement several ML techniques and investigate their properties regarding the quality of results, efficiency, and usability.

4 Discussion and Conclusion

For software companies it will be necessary to embrace robot automation testing-based ML to optimize tests execution time and effort by automating the whole software testing process, leading to an intelligent system. When we mention ML within the DevOps pipeline, it is essential to consider how ML can monitor ongoing Continuous Integration (CI) builds and point out trends within build-acceptance testing, API testing, and other testing areas. ML models can investigate the entire CI pipeline for failed builds. CI builds are often not appropriately reviewed and repeatedly die without attention. With ML entering this process, the immediate value is a shorter cycle and more stable builds, translating into faster feedback for developers and cost savings.

Machine learning applications in software testing, determining the role of the human in supplying enough input information and leveraging the models is critical. Automation testing, for example, is commonly performed by testers with extensive domain knowledge by finding execution situations of interest. This is, for example, the purpose of Category-Partition. However, determining failure criteria based on the execution results of a test suite is difficult for any person. Therefore, our fault localization technique, as stated above, was built on the assumption that testers would submit

Category-Partition specifications. This professional input is most likely what will make this technique scalable and viable [3].

In this work, we have presented a design of conceptual framework for RF and ML and performed functional testing of a web application developed with JAVA11, connection to machine learning algorithms, and Robot Framework 4.0.2 [9]. Robot framework automated database testing can help software developers evaluate database-based web applications, write all SQL queries as required in test cases, and reduce the effort to perform this task compared to manual or semi-automated tests. Additionally, ML provided the robot with automated tests cases, which generated similar test cases. The software testing tool can be selected based on the application needed to be tested, budget, usage, and efficiency. Jenkins compiles the program codes and prompts the developer when the violation occurs. In this paper, we have also shown how the adoption of database testing can help a company achieve better results, high performance, and reduce test execution time.

As already mentioned, this paper does not provide any particular demonstration of ML-based algorithms on case studies test suite with analysis of numerical results and comparisons between other (non) ML-based approaches. The main aim of this work is to clearly describe the motivation, possibilities, protocols, components, and code examples required for understanding and implementing the robot test automation process approach.

Acknowledgement: This work supported by the Internal Grant Agency of Tomas Bata University under the project no. IGA/CebiaTech/2021/001, and further by the resources of A.I.Lab at the Faculty of Applied Informatics, Tomas Bata University in Zlin.

References

[1] Beck, K. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[2] Bihlmaier, A., and Wörn, H. Robot unit testing. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots* (2014), Springer, pp. 255–266.

[3] Briand, L. C. Novel applications of machine learning in software testing. In *2008 The Eighth International Conference on Quality Software* (2008), IEEE, pp. 3–10.

[4] Durelli, V. H., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R., and Guimarães, M. P. Machine learning applied to software testing: A systematic mapping study. *IEEE Transactions on Reliability 68*, 3 (2019), 1189–1212.

[5] Gojare, S., Joshi, R., and Gaigaware, D. Analysis and design of selenium webdriver automation testing framework. *Procedia Computer Science 50* (2015), 341–346.

[6] Gondra, I. Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software 81*, 2 (2008), 186–195.

[7] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems 27* (2014).

[8] Guo, X. Towards automated software testing with generative adversarial networks. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)* (2021), IEEE, pp. 21–22.

[9] Lenz, A. R., Pozo, A., and Vergilio, S. R. Linking software testing results with a machine learning approach. *Engineering Applications of Artificial Intelligence 26*, 5-6 (2013), 1631–1640.

[10] Marale, P. S., and Chandavale, A. A. Implementation of rest api automation for interaction center. In *Intelligent Computing and Information and Communication*. Springer, 2018, pp. 273–277.

[11] Noorian, M., Bagheri, E., and Du, W. Machine learning-based software testing: Towards a classification framework. In *SEKE 2011: The Twenty-Third International Conference on Software Engineering and Knowledge Engineering* (2011), pp. 225–229.

[12] Shi, C., Wu, C., Han, X., Xie, Y., and Li, Z. Machine learning under big data. In *6th International Conference on Electronic, Mechanical, Information and Management Society* (2016), Atlantis Press, pp. 301–305.

[13] Stouky, A., Jaoujane, B., Daoudi, R., and Chaoui, H. Improving software automation testing using jenkins, and machine learning under big data. In *International Conference on Big Data Technologies and Applications* (2017), Springer, pp. 87–96.