TEST AUTOMATION FOR CONTINUOUS INTEGRATION IN SOFTWARE DEVELOPMENT

Tudor-Alin NIȚESCU[1], Andreea-Iulia CONCEA-PRISĂCARU[2], Valentin SGÂRCIU[3]

Software application testing represents one of the crucial steps in an application release lifecycle, and it usually consists in performing different sets and types of tests, after every update on the code of the application. Therefore, we can notice the presence of manual and repetitive testing tasks, and we can automate them using performant tools. Moreover, the set of automated tests can be included in the continuous integration flows, where the software engineers usually define a set of tasks to run automatically on every change on the application codebase. By doing this, the set of automated tests can be able to detect the eventual problems in the new changes on the code as early as possible, minimizing the faulty code changes that could get released for a new version of the software application. Besides that, test automation can save precious time of the software engineers team, if the set of tests is defined correctly and scheduled to run properly for each new code change.

Keywords: functional testing, unit testing, test automation, continuous integration, software development, pipeline, automated build, Azure

1. Introduction

Continuous integration, also known per its abbreviation as CI, represents a software development practice where the software engineers of an application regularly add their code to a centralized location, named repository. A code repository represents the location in which all the code base of an application resides, so each engineer working on the application can access it, modify it, improve it and so on.

Continuous integration is, most of the times, related to the build stage of an application, where the engineers create a new version of it, called release. The release can contain either some new functionalities for the software application, or just some fixes related to wrong working functionalities found within it (also known as bugs). But in order to avoid, or at least try to reduce the impact of a bug

[1] PhD student., Dept. of Automation and Industrial Informatics, University POLITEHNICA of Bucharest, Romania, e-mail: tudor_alin.nitescu@stud.acs.upb.ro
[2] PhD student., Dept. of Automation and Industrial Informatics, University POLITEHNICA of Bucharest, Romania, e-mail: andreea.concea@stud.acs.upb.ro
[3] Prof., Dept. of Automation and Industrial Informatics, University POLITEHNICA of Bucharest, Romania, e-mail: valentin.sgarciu@upb.ro

as much as possible in a software project (avoid application downtimes, malfunctioning features resulting in unintended actions in the software application and so on), the CI practices come in handy for a variety of situations: find and address bugs quicker, improve software quality, reduce the required validation time for a new release, smoothen the release process etc.

In order to apply this software principle, there are two main components needed: one is the automation component (which can be chosen from a variety of solutions found on the software market and adapted to the needs of the project) and the other one is the cultural component (this is more related to the willingness of the software engineers from a project to learn the continuous integration principles and apply them as much as possible) [1].

The success of CI should be accredited to the variety of strong tools that can be used in the present, which, when configured and used properly, can considerably automate most of the required steps to inspect, integrate, and test the source code changes of a software application in a transparent and straightforward manner. The usage of CI tools not only accelerates the release process of a software application, but also helps in avoiding undesirable events such as faulty code in new application versions, helping with the detection and also with the prevention of them.

However, the advantages of the CI tools usage within a software development team are only present and useful when the respective team adheres to the best practices of this development practice. Unless the development team is willing to change their development culture, the automation tools may produce no benefits to them, and it will even harden their workload. For example, when integrating a suite of automated test in an application build, the engineers must ensure that the test suite is appropriate, which is a thing that none of the CI tools can take care of, but only the ones designing the tests to be integrated in the automated builds can [2].

Functional testing is a very important step of the software development process, and it shouldn’t be neglected, skipped or performed unproperly. It can help us prevent, detect and fix possible issues that might appear in the development phase. This process involves both manual testing and automated testing and it can save us a lot of time if it is designed properly. Usually, in this phase, the testers and/or the developers test all the possible business flows of the application, including both positive and negative scenarios. Depending on the run frequency of the tests, the team can determine which flows can be automated, by doing an analysis of the most recurrent manual steps performed during the testing phases.

In this paper, we will demonstrate the best practices of test automations for continuous integration of software, how to design the CI for software applications and which tools are the most powerful to use for automations.

2. State of the art

The need of automation in the testing field came as a result of the many repetitive and recurrent tasks performed in the testing stage of the software development life cycle. Nowadays, the trend for test automation is constantly increasing, as on the market there are a lot of software tools and applications available for this purpose. The trend was followed by most software development companies on the market, and not only, also researchers in the software development area were interested by this topic and delved more into it.

In this section we are going to present the current state of the art in the software testing area and interesting findings in this domain.

A group of German researchers approached the topic of test automation in the context of designing and building various test suites for different cases, environments and system parametrizations. Their approach intended to reduce execution times by skipping the tests that will never get executed in a specific environment due to some parametrizations. The implementation was done for a software product in the area of public transportation, where system parameters and setups can change the flows of the tests [3].

Furthermore, in the researchers spotlight software tools such as Selenium, QTP and Cucumber were present. The case studies have shown that Selenium is the most performant and commonly used tool for automating the tests for different Web applications. It has been proved that Selenium is suitable for the testing automation field due to its time management performances and rich information logs [4].

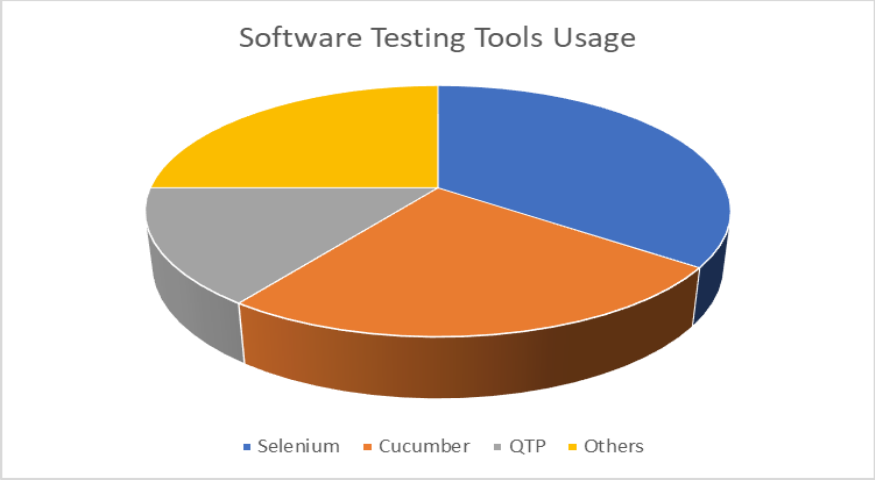

Fig. 1. Software Testing Tools Usage within companies

Another research done by a Bulgarian researcher was a case study for test automation in microservice architectures. Their main focus was on the switch of a monolith system to a microservice architecture, where each functionality is divided separately in services. The main focus when developing the automated tests was the testing pyramid (Fig.2). What they wanted to research was the way in which separate microservices interact to each other, so if one gets changed, the impact on the others can be predicted accordingly. Their test automation system was based on a solution based on PACT framework (People, Activities, Context and Technology). They gathered information about each service, how it works and how it interacts with its "neighbors", and then they created the test files accordingly, isolating the testing of a microservice, regardless of the external services with which they were integrated. This helped them to migrate the monolith system to a microservice architecture, moving the business logic into separate services, and developing more and more automated tests for the new services created [5].

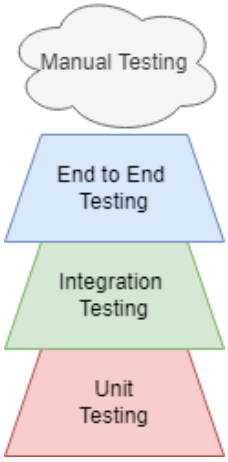

Fig. 2. Testing Pyramid

In another research, done by a group of Nigerian students, the test automation subject was approached for educational portals, more exactly focusing on the enhancement of quality and behavior of the applications from that category, having as case study the Obafemi Awolowo University. They designed a testing framework using Java language and used MySQL to store the test data. The developed framework was able to support test automation for any portal powered by web technology, creating HTML reports with various stages of the test execution process. They evaluated the performance of the framework using different indicators, such as Test Time Performance, Performance Test Efficiency and Automation Scripting Productivity. Their results shown that the framework

produced about 360 operations per hour, the test efficiency was 80% and the test time performance was 4% (higher the percentage means more testing time) [6].

3. Methodology

3.1. Test automation and best practices of test automation

Test automation represents a very important stage in the life-cycle of an application, if it's planned during the early stages of the project. Furthermore, it is a good practice to involve the testing engineers from the business requirements analysis stages, when the requirements are designed, in order to review these activities and get a good understanding of the customer's needs. Therefore, the test engineers can produce more detailed test designs and create more appropriate test environments. In addition to this, a well-defined set of functional requirements will surely help in the automated testing and will reduce the time and cost required for the testing [7]. Based on that, we can define the automated tests lifecycle in Fig.3 below:

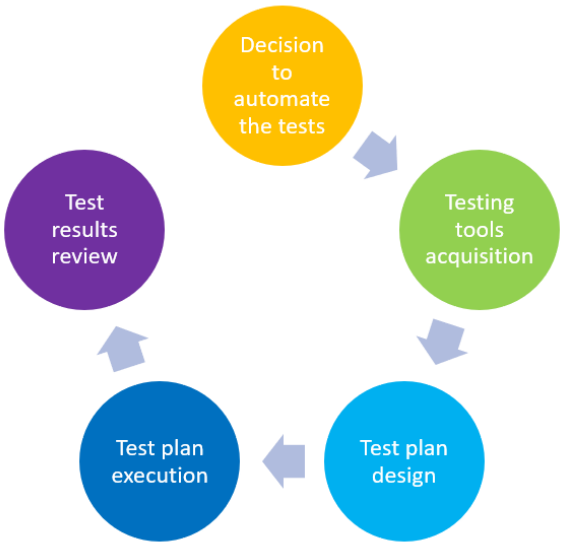

Fig. 3. Automated testing lifecycle

First, when the decision to automate the tests for a project is taken, an analysis should be done in order to decide what is the appropriate testing tool to be used. After its acquirement, the test plan should be designed, containing all the business test cases described as detailed as possible, so the testing team could understand. This step is followed by the test plan execution, and last but not least

after the test results are obtained, they should be further analyzed in order to improve the process. When following the automated testing lifecycle, a good set of practices must be followed by the engineers, otherwise the results of the testing won't be as effective as they should be.

Based on 26 academic literature and 55 grey literature sources, here are some of the best practices for test automation [8]:

- ✓ Define an effective test automation strategy and adjust it to the eventual changes on the project
- ✓ Involve the key stakeholders when defining the strategy
- ✓ Provide enough testing resources
- ✓ Have competent test professionals and keep them motivated
- ✓ Promote collaboration within the team
- ✓ Share the available test automation knowledge
- ✓ Allow enough time for the engineers to train and learn the test automation approach
- ✓ Use the appropriate testing tools
- ✓ Set up good test environments, suitable for the cases tests
- ✓ Create high-quality test data
- ✓ Define test automation requirements based on efficient and effective analysis (detailed test scenarios, use cases)
- ✓ Prioritize the automated tests in the execution flows
- ✓ Report the test automation results to the key stakeholders involved
- ✓ Use the right test automation metrics to measure the performance
- ✓ Always analyze and adapt to the new technologies for the automation of the tests

3.2. Test automation software

Nowadays, there are a lot of solutions available on the market for the test automation field. From the long list of test automation frameworks, we can mention the most popular ones, which are: Selenium, Cucumber and QTP.

Selenium is an open-source framework used for test automation. It is widely used for testing web applications, and it offers support for different programming languages, operating systems, browsers and others. From the long list of programming languages that Selenium is offering support to, we can mention the following ones: Python, Java, C, Ruby and PHP. Selenium is composed of several components, IDE, RC, Web Driver and Grid. Selenium IDE is a browser plugin that enables the reproduction of the steps that the user is performing. Selenium RC is a user interface, which allows us to perform commands from the editor and also tests on the application. The Web Driver is used for sending the information to a web server. After the request is performed, a

response will be received in exchange. Last, but not least, the grid is an extra feature that Selenium is offering, allowing the user to execute tests in parallel on several computers and browsers at the same time, having the advantage of short execution time [9].

Cucumber is another test automation software solution, being mainly used by the software development companies and other enterprises. The framework is used for functional and/or acceptance testing, in designing the testing flows and running them based on different schedules. The test suites written in Cucumber are easy to read and understand, due to the Gherkin syntax, which tries to imitate the human language in the testing steps code. Cucumber is offering support as well for different programming languages (Java, Scala, Python etc.). The main advantage of this framework is the increased level of collaboration and communication between different roles inside and outside the company (programmer, analyst, tester, business owner), because the syntax of the tests is easy to understand by anybody who knows the expected flows to be tested. The test flow is written as a feature, and it can contain one or more scenarios (test cases). The test scenarios can have one or several steps (testing actions) [10].

3.3. Continuous integration tools

The continuous integration tools are used to develop automated builds for the software applications, also known as pipelines, using a set of stages like build, test, publish, deploy and so on. In the test stage of a continuous integration build, the automated tests can be integrated and scheduled to run on each new version of the application. The solutions for CI present nowadays are very well known and strong, and we can mention two of the most popular ones: Azure DevOps and AWS (Amazon Web Services).

Azure DevOps is a Cloud computing platform offering services such as repositories, where the code of the application can be managed and stored, and pipelines, where the engineers can create and manage different pipeline builds for the software application. Azure Repos is used to manage the code in a project, tracking changes and creating updated versions of the code. It keeps a history of the code changes, and has an option to rollback, to get back to any past version of the implementation. This is mainly used to manage the code changes and then build the application based on the stored code, using the pipeline builds. Azure Pipelines represents a feature that can be used by engineers to design automated builds of the applications, with different stages, prerequisites and configurations, based on the application design. Among this stages, a testing one can be created, and here the engineers can include different sets of test to run after each code change on the application, such as unit testing, regression testing, functional testing and so on, depending on the test plan and application design [11].

AWS (Amazon Web Services) is also a Cloud computing solution, that allows the users to store their application code and other data in repositories (AWS CodeCommit) like in Azure DevOps, as well as allowing the engineers to create automated builds, using AWS CodePipeline. It's a quite similar solution when compared to Azure one, being a Cloud solution that offers a big chunk of the features also offered by the Microsoft solution, distinguished by some technical aspects of each functionality. Usually, the choice between them is made based on an analysis for the needs of the company, based on details like costs, ease of implementation, knowledge base of the engineers, etc. [12].

In this paperwork, we will focus on Azure DevOps as the main CI tool, and we will try to showcase the integration of automated tests in an application build using it.

4. Implementation and results

For this paperwork, we have used a sample Java project that implements a virtual board game, and we wrote Unit Tests for it using JUnit framework. The test suite shown in this example was a regression suite, which covered all possible test scenarios that could be impacted by a pending change done as a proof of concept. To integrate the regression test suite in an automated countinuous integration flow, we have developed an Azure Pipeline that builds the application and then runs the tests. If the build fails, we have put the option to not allow the merging of the new code (source branch) into the target branch (master).

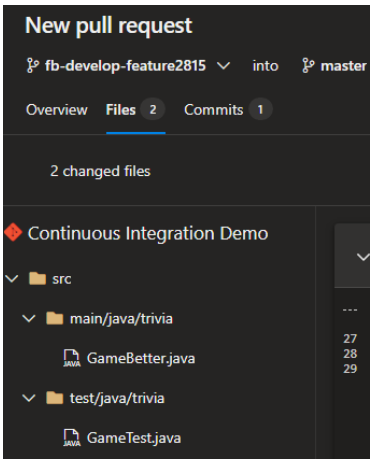

Fig 4. Adding the new code to master

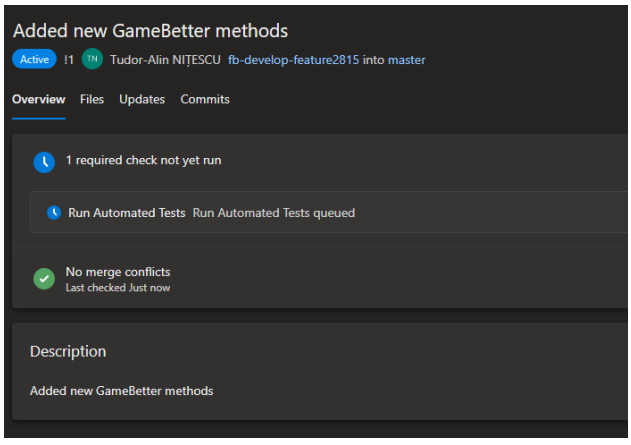

Fig 5. Triggering a new build for the added code

On the images from above, we can see that the new code is being added to the master branch, on the left side (which should contain the application that gets released to the customer), and the new change targets a java class and a test suite. On the right side, we can see that a new build, named by us "Run Automated

Tests" has been triggered to run automatically, and it is a required check that needs to pass in order for the new code to be added. We will see that the new code added made three tests to fail, and therefore the whole build failed and the code in its current state could not reach master (Fig. 6)

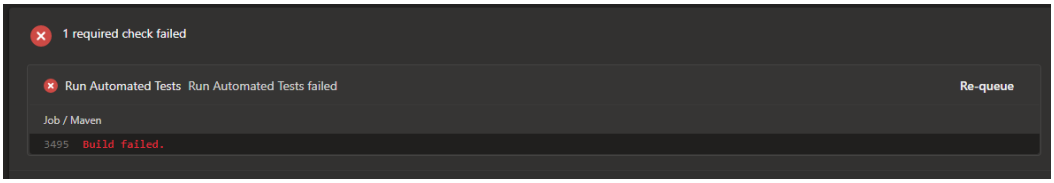

Fig 6. Failed build for the new code added

If we go to see the details about the failed build, more exactly, in the detailed logs of the pipeline (Fig. 7), we can see the exact tests that failed, which can hint us where to look exactly, and there could be two possibilities: either the code changes were faulty, or the new tests added for the regression were badly written. This has to be an analysis done to find the best solution and push a new change to the pull request. Besides that, Azure offers another detailed (and more graphical option) about the automated test run results (Fig. 8). We can see that out of 14 tests that run as part of this automated build, 3 failed, resulting in a 78.57% pass percentage. We also see the run duration, which was about 1.6s.

```
Results :

Failed tests:   testCreateQuestion1(trivia.GameTest): Change detected for question Test question.
  testModifyQuestion2(trivia.GameTest): Change detected for modified question Test question. To br
  testModifyQuestion5(trivia.GameTest): Change detected for modified question Test question. To br

Tests run: 14, Failures: 3, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  10.060 s
```

Fig 7. Failed pipeline detailed logs

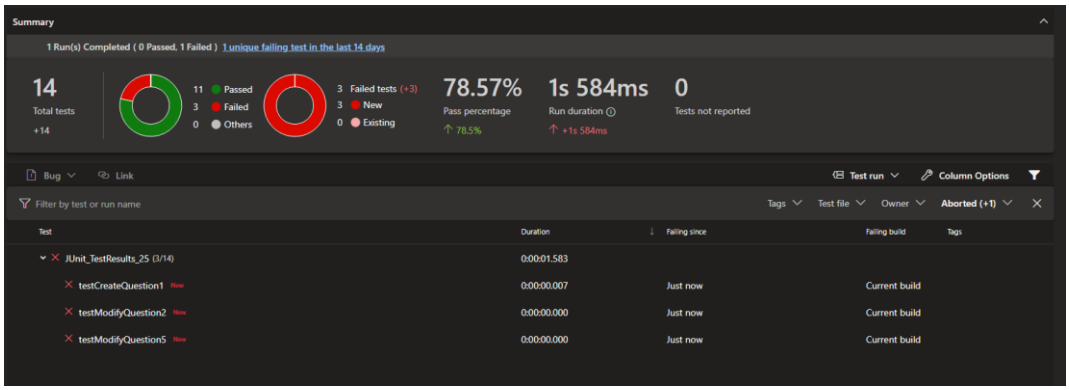

Fig 8. Automated test run report for a failed build

After the change is done in either the code or in the tests (in our case, we intentionally wrote faulty tests to prove that not only the code can impact the tests, and it is required to follow the best practices for test automation), and a new commit is made to the pull request, another build will be triggered, to test the updated changes. In our case, we fixed the tests to have the correct behaviour and the new build passed (Fig. 9)

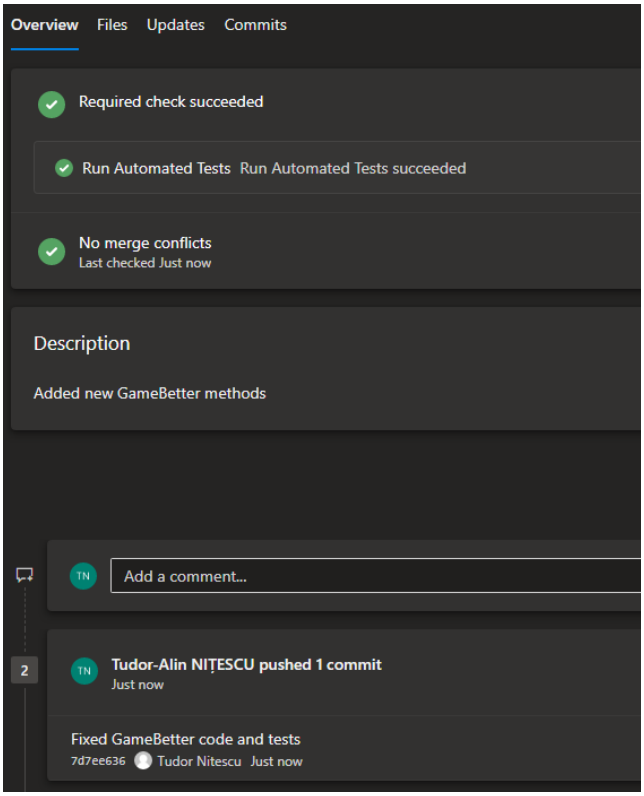

Fig 9. Passed build for the updated tests

After the build passed, the new code can be merged into master, and a new version of the application can be released. We can also see the updated test report in Fig. 10, with no test failures after the latest update, and the execution time remaining approximately the same.

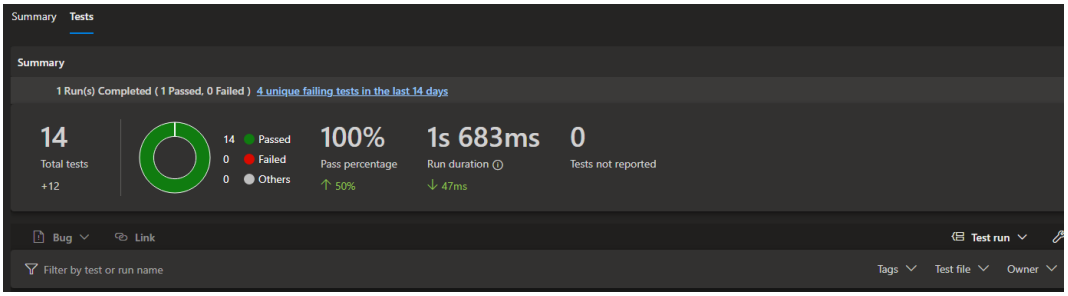

Fig 10. Automated test run report for a passed build

5. Conclusion

In this paperwork, we highlighted the best practices to follow when writing tests to be automated, and we've also presented a modern solution that can be used for Continuous Integration (Azure DevOps) to include automated test suites. Build automation can spare manual work time for the engineers and can be used to detect eventual code flaws in the early stages of a project. However, we should not neglect the best practices when writing test suites to be automated (or even ran manually). By creating strong collections of tests, we can always count on the automated builds that integrate them to find any eventual issues, indicating precisely which code has problems that needs to be addressed in order to do a new release for the software application. In the end, the results of our build were successful, the entire test suite passed, with ~1.6s execution time, significantly less time than it would have taken for a manual testing of the code. Besides that, it is important to be mentioned that the set of tests to be automated should be implemented very carefully, not only to cover the functionalities/code correctly (using the best practices for that), but also to avoid leaking sensitive information in the test log results, if these would be accessed by a potential attacker. By using the solution provided from Microsoft, the access to the pipelines/logs can be restricted only to certain members of a team.

REFERENCES

[1]. *S. Rossel*, "Continuous Integration, Delivery and Deployment. Reliable and faster software releases with automating builds, tests and deployment", Packt Publishing, 2017.

[2]. *W. Felidré, W.F.L. Furtado, D.A da Costa, B. Cartaxo, G. Pinto*, Continuous Integration Theater, IEEE, 2019.

[3]. *N.T.T Soe, N. Wild, S. Tanachutiwat, H. Lichter,* "Design and Implementation of a Test Automation Framework for Configurable Devices.", 2022 4th Asia Pacific Information Technology Conference (pp. 200-207), January 2022.
[4]. *S. D. Ezhuthachan, J.K.Tailor,* "Test Case to Test Automation with Selenium IDE: a Case Study.", Compendium of Management Case Studies (2022): 263, 2022.
[5]. *N. Raychev*, "Test automation in microservice architecture", IEEE Spectrum, 2020.
[6]. *A. Aniwange, P.T. Nyishar, B.S. Afolabi, A. O. Ejidokun*, "A hybrid software test automation for educational portals", Novateur Publications, International Journal of Innovations in Engineering Research and Technology, 2021.
[7]. *A. Asif, K. Maheen, K. Hameed,* "Software Test Automation", Instant Approach to Software Testing: Principles, Applications, Techniques, and Practices, Bpb Publications, 2019.
[8]. *Y. Wang, M. V. Mäntylä, Z. Liu, J. Markkula, R. P. Jurvanen*, "Improving test automation maturity: A multivocal literature review", Software Testing, Verification and Reliability, e1804, 2022.
[9]. *V. Ray,* "Inside Clinical Arena Automated Evaluation Using Selenium testing framework", YMER Digital, Volume 21, 2022.
[10]. *R. Anand, M. Arulprakash,* "Business driven automation testing framework", International Journal of Engineering & Technology, 7 (2.8) (2018) 345-349, 2018.
[11]. *D. Obayomi, D. Athyala, J. K. Duphey, G. Kapadiya*, "Azure DevOps", Frankfurt University of Applied Sciences, 2021.
[12]. *A. Bandaru*, "Amazon Web Services", AWS Conference, 2020.