Automation of Selection of a Pool of Graphical Interface Regression Tests for Multi Module Information Systems [*]

Ilya Zarubin [0000-0002-4870-6171] and Aleksander Filinskikh [0000-0003-3826-6771]

Nizhny Novgorod State Technical University n. a. R.E. Alexeyev, Nizhny Novgorod, Russia
simarglz@yandex.ru, alexfil@yandex.ru

Abstract: Features of using the regression test selection method for automated testing of the graphical user interface in the development of information systems that consist of a set of modules are considered. The source of the need to create additional test environments required in the development of multi-module information systems that are using databases is specified. The three most popular approaches to organizing test environments – Copying, Scaling, and Scaling with synthetic data generation – are considered. The positive and negative sides are considered in terms of implementation, using, and resources spent on creating and maintaining resources, as well as in terms of the reliability of the results obtained in the process of testing models created using these approaches. The positive aspects of checking the quality of complex multi-module information systems from the point of view of the graphical user interface by various testing methods and, in particular, in the process of performing regression testing are presented. The positive aspects of using regression testing automation in conditions of lack of resources using various software platforms are indicated. The advantages of using the dynamic selection method for regression tests for automated testing are also given, as well as recommendations for implementing the selection method in existing and beginning projects.

Keywords: Regression Testing, Automated Testing, Graphical User Interface Testing, Selection of Test Scenarios.

1 Introduction

The process of developing information systems (IS) [1] is a complex procedure consisting of a significant number of stages, which, in turn, can be divided into some phases. In a simplified way, the IS development lifecycle can be divided into

[*] Publication is supported by RFBR grant №19-07-00926.

planning, approving the IS architecture, writing code, testing, implementing, and supporting [2].

Of course, each of these stages is important and cannot be excluded from the IS development process. Depending on the chosen model of IS development, these stages can follow strictly one after another or be performed cyclically [3]. Thus, from the point of view of commercial success is testing phase the most important because it allows us to consider IS from the perspective of the end user, i.e. the consumer of the software product, even if the architecture of is illogical, the code is overly complicated and unreadable, but during testing was discovered almost all the important errors in functionality and usability for the user is this is going to be more attractive than a system with well written code, but full of errors.

The testing stage, in turn, is also divided into several stages. Usually, a test strategy that plans the types, categories, and order of testing is developed for each project separately, but there is a list of tests that are mandatory for any of this – unit testing, integration testing, smoke testing, release testing, system testing, regression testing, interaction testing [4], and so on. Each of these test types has its own goals and objectives and they are performed at different stages of information system development [5], but regression testing (RT) [6] is a very important stage of detecting errors in is that are under active development and functional growth, while forming a list of tests for RT is a very non-trivial task.

Regression testing is a special type of IS quality control aimed to detecting errors in areas and modules that have not been modified specifically but may have lost their correct functioning due to changes made to adjacent areas or modules. The complexity of conducting a correct RT procedure is caused by the need to understand the IS architecture when selecting tests for RT, which is especially difficult in the case of developing a multi-module is with possible interaction of modules not directly, but indirectly. In addition, it is necessary to have a good knowledge of various methods for selecting scenarios for RT, which may not be available in the context of saving resources for IS development and insufficient qualification of testing engineers.

The frequency of RT execution depends on the specifics of the developed IP, the selected IS development process [7], as well as the frequency of issuing new versions of the software product to the customer, but it can not be less than once per development cycle. At the same time, it is necessary to understand that the scenarios performed within the RT are, as a rule, a standard test scenario [8]. In the case of frequent release of IS versions, repetition of the same scenario by the testing engineer often leads to reduced testing efficiency and omission of errors. To avoid such situations, make sense to automate regression testing to shift the performance of routine regular operations to automated test management systems.

2 Regression Testing of the Graphical User Interface

One of the simplest and most effective ways to check the quality of IS which are developed using the most popular process continuous integration/continuous delivery (CI / CD) [9], is to check the correctness of the graphical user interface (GUI) [10] – the end user of non-specialized IS evaluates the quality of the interface and the correctness of reactions to the user's impact.

In simple monolith ("solid") IS [11], which consist of a single module which performs all the necessary calculations, the interface is one of the parts of the code that closely interacts with other parts of the code. Checking the correctness of the display of the elements of the UI, transmitting graphical information between various software environments, information systems and formats [12], their presence and successful interaction with each other allows us to identify with high reliability the occurrence of possible errors when making changes or additions to the code.

For small sites on the Internet [13], as well as mobile applications, the assessment of the correctness of the UI [14] also helps to conclude with a high degree of confidence that errors have occurred made by any change by simply comparing the results of RT before and after the changes made.

For both types of described IS, the task of organizing an isolated environment [15] for regression testing is not difficult – the version of the IS that is being tested can be deployed on household computers or on mobile phones.

Multicomponent IS are becoming increasingly popular for various reasons [16]. they interact with a large amount of information [17], several databases, hundreds of processors, and thousands of gigabytes of RAM. Such IS requiring a separate approach to testing, which is due to the large amount of resources required for the correct operation. The most common approaches to organizing test environments for checking the quality of multicomponent IP systems are the following:

1. Duplicating the infrastructure and data of an instance which is currently in commercial use (Fig. 1).

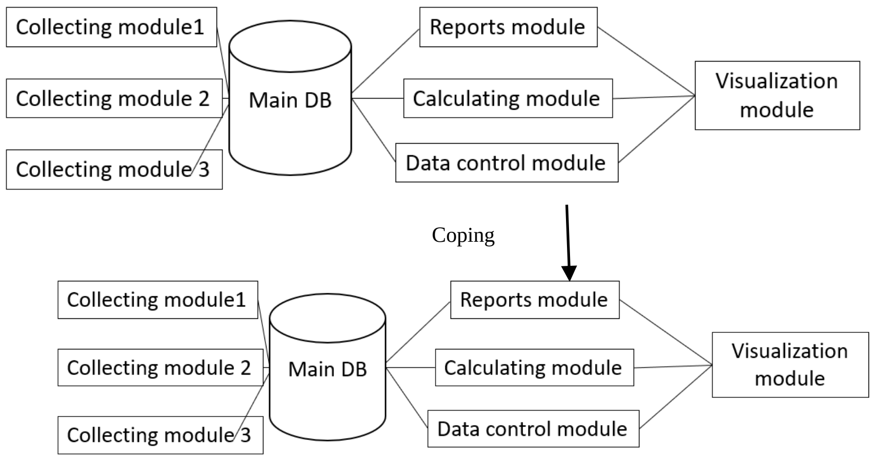


Fig. 1. Duplicate infrastructure for testing

This approach is only applicable for small multi-module IS with a small incoming data flow.

2. Using a scaled-down industrial instance model-using fewer resources, copying a certain percentage of data from the industrial instance database (Fig. 2).

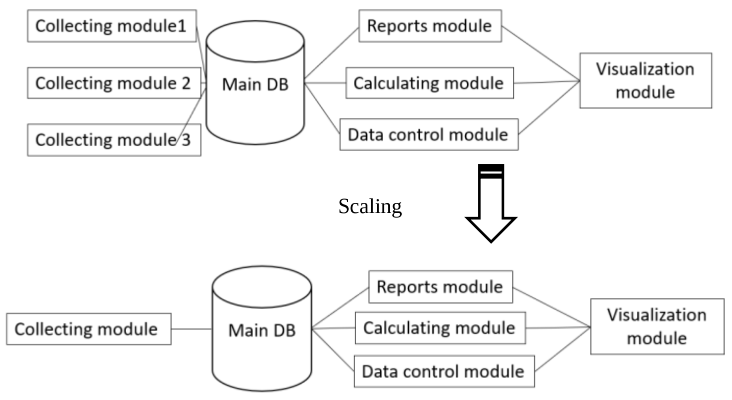


Fig. 2. Scaling the infrastructure for testing

This approach allows you to save the resources which are necessary for the operation of a test instance of the IS, while maintaining the ability to interact with modules of different levels. At the same time partial copying of data from an industrial database allows you to perform testing on data with a high degree of similarity to real data.

3. Using a scaled-down industrial instance model with generating synthetic data in the database and using the necessary modules (Fig. 3).

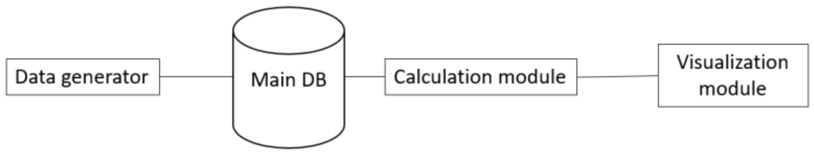


Fig. 3. Test infrastructure with data generation and use of necessary modules

This approach allows you to drastically reduce the resources necessary for deploying the test infrastructure, isolate one or more connected modules, and generate a certain range of data, which allows you to check the correctness of the system in simulated situations (in situations that occur extremely rarely in industrial operation, as well as in emergency situations). The application of this approach is significantly simplified when using automatic control systems for containerized applications, such as Kubernetes [18].

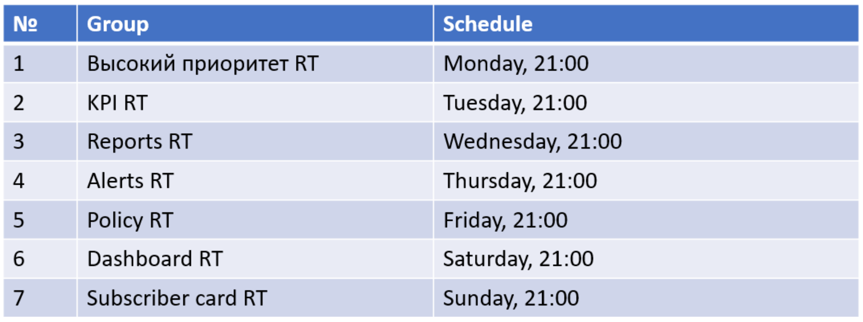
3 Automation of Regression Testing

Using the most common and simple methods for selecting regression tests (selection by priority, functionality, result of previous execution, date of execution) leads to the formation of a fairly stable list of tests. In the process of developing an IS with a regular CI/CD functional increment, regression testing is performed regularly, meaning that testing engineers have to regularly perform the same scenarios, which can lead to a decrease in concentration when performing such tests and as a result a decrease in the effectiveness of searching for errors in the IC. To avoid such situations, as well as to make more efficient use of testing engineer's resources, modern regression testing projects use automation for executing test scenarios [19].

Once an automated test can be run an unlimited number of times in the future, almost without the participation of an engineer. Tests can be developed and run using, for example, TestComplete, Selenium, or Robot framework environments. The automated launch of RT can be combined with the release of a new version of the IS using, for example, the software system CI / CD Jenkins [20].

In the case of a long process of IS development, the number of automated tests and regression tests can reach large amounts and, consequently, the execution of all automated tests can take a very long time. If the IS is a small application, site, or mobile device app, you can solve the problem of running automated tests for a long time by running these tests on multiple devices in parallel.

Running a significant number of automated processes for complex multi-module is with a database can take a very long time, while running several processes in parallel may not be possible due to the presence of only one test infrastructure, which can also be used for manual testing during the working day. It is possible to partially solve this problem by dividing all automated tests into groups and running tests in groups according to a certain schedule (Fig. 4).



№	Group	Schedule
1	Высокий приоритет RT	Monday, 21:00
2	KPI RT	Tuesday, 21:00
3	Reports RT	Wednesday, 21:00
4	Alerts RT	Thursday, 21:00
5	Policy RT	Friday, 21:00
6	Dashboard RT	Saturday, 21:00
7	Subscriber card RT	Sunday, 21:00

Fig. 4. Grouping the execution of automated tasks

Such approach allows you to implement the execution of all automated RT but requires significant costs for the correct formation of the list of RT and the schedule for running these tests.

4 Automatic Selection of the Regression Test Pool

In projects of the development of multicomponent IS with the regular release of new versions with high functional growth, the issue of prioritizing, selecting and minimizing the list of RT is very acute not only because of insufficient time to conduct all the necessary checks, but also because of the high employment of the test infrastructure. It is necessary to consider that the practice of CI/CD can lead to results new version of is almost every day, which entails the need to conduct all possible checks in the shortest terms and, therefore, to spend time on manual selection of automated RT is very desirable for the reason that there are a significant number of methods of selection of RT, which are based on different principles of determining whether to run the test and correct their application often requires a large amount of resources. In addition, when testing multicomponent IS, it is highly desirable to consider the possible mutual impact of changes in modules and, more importantly, the impact of changes on the UI. The method of selecting and ranking test scenarios considering an arbitrary number of selection methods allows you to automate the selection of RT.

Let us assume that the selection of tests for RT is made using a certain number of selection methods. For each of these methods, you can determine the significance of this selection method in terms of the importance of running the test and whether the test can detect an error. For convenience of calculations, you can use the value of the test significance in the range from 0 to 100.

Thus, the methodology for calculating the significance of the test for an arbitrary number of RT selection methods can be defined as follows:

$$I_a=\sum 〚(I_1+I_2+\cdots+I_n)〛 , \qquad (1)$$

where
I_a – the total value of the test;
I_1 – significance of the test selected by method 1;
I_2 – significance of the test selected by method 2;
I_n – significance of the test selected by method n.

Using (1), we get the significance of the test in terms of the possibility of detecting errors in the IS – the higher the received significance of the test, the higher the priority of the test execution.

Separately, it should be noted that (1) is correct for the significance of the test (ST) for each selection method in a certain General range, for example, from 0 to 1 or from 0 to 100. If it is impossible to fulfill this condition for various reasons, it is suggested to use additional normalization by weight coefficient (WK).

$$I_a=\frac{w_1\times I_1+w_2\times I_2+\cdots+ w_n\times I_n}{w_1+w_2+\cdots+w_n}, \qquad (2)$$

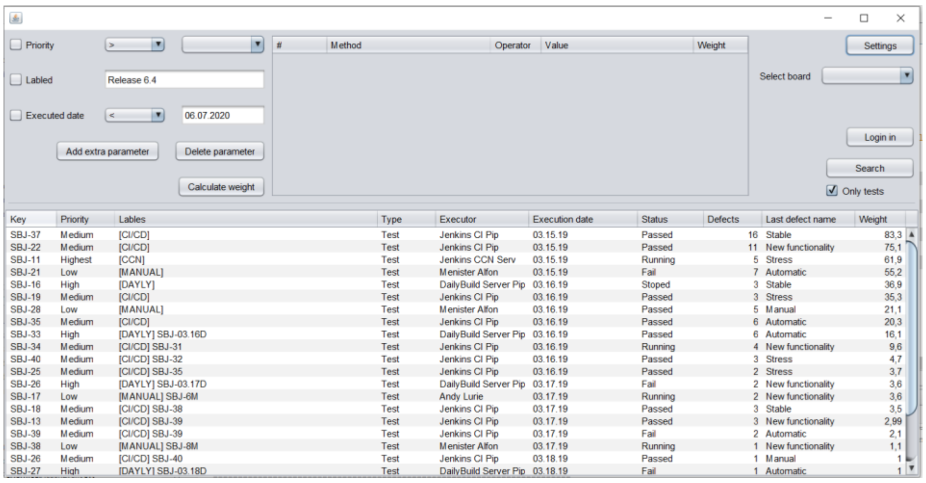
where
I_a – total ST;
w_1 – WK for test group 1;
I_1 – ST, selected by method 1;
w_2 – WK for test group 2;
I_2 – ST, selected by method 2;
w_n – WK for test group n;
I_n – ST, selected by method n;

Thus, using (2) it is possible to get a queue of texts selected using an arbitrary number of selection methods, which allows us to take into account the potential mutual influence of IS modules, ranked by the priority of test execution, taking into account the use of the UI. Having data in the test about the IS modules that are necessary for the correct execution of this test will allow you to limit the use of test infrastructure resources only to those modules that are necessary.

In the future, the resulting list of automated tasks can be entered into the configuration file of the software system for ensuring the continuous integration process and will be performed in a period of time when the test infrastructure is not involved in higher-priority tasks.

5 Using the Methodology for Selecting and Ranking Test Scenarios

To implement the described method of RT selection, a software package was developed that allows working with various testing management systems via the API. Development was carried out in the object-oriented programming language JAVA. The UI of the software package provides the possibility to use an arbitrary number of test selection methods for RT, specify the WK for each of the selection methods, and an area for displaying the test pool, ranked in descending order of ST (Fig. 5).



Labled: Release 6.4

Executed date: 06.07.2020

☑ Only tests

#	Method	Operator	Value	Weight

Key	Priority	Lables	Type	Executor	Execution date	Status	Defects	Last defect name	Weight
SBJ-37	Medium	[CI/CD]	Test	Jenkins CI Pip	03.15.19	Passed	16	Stable	83,3
SBJ-22	Medium	[CI/CD]	Test	Jenkins CI Pip	03.15.19	Passed	11	New functionality	75,1
SBJ-11	Highest	[CCN]	Test	Jenkins CCN Serv	03.15.19	Running	5	Stress	61,9
SBJ-21	Low	[MANUAL]	Test	Menister Alfon	03.15.19	Fail	7	Automatic	55,2
SBJ-16	High	[DAYLY]	Test	DailyBuild Server Pip	03.16.19	Stoped	3	Stable	36,9
SBJ-19	Medium	[CI/CD]	Test	Jenkins CI Pip	03.16.19	Passed	3	Stress	35,3
SBJ-28	Low	[MANUAL]	Test	Menister Alfon	03.16.19	Passed	5	Manual	21,1
SBJ-35	Medium	[CI/CD]	Test	Jenkins CI Pip	03.16.19	Passed	6	Automatic	20,3
SBJ-33	High	[DAYLY] SBJ-03.16D	Test	DailyBuild Server Pip	03.16.19	Passed	6	Automatic	16,1
SBJ-34	Medium	[CI/CD] SBJ-31	Test	Jenkins CI Pip	03.16.19	Running	4	New functionality	9,6
SBJ-40	Medium	[CI/CD] SBJ-32	Test	Jenkins CI Pip	03.16.19	Passed	3	Stress	4,7
SBJ-25	Medium	[CI/CD] SBJ-35	Test	Jenkins CI Pip	03.16.19	Passed	2	Stress	3,7
SBJ-26	High	[DAYLY] SBJ-03.17D	Test	DailyBuild Server Pip	03.17.19	Fail	2	New functionality	3,6
SBJ-17	Low	[MANUAL] SBJ-6M	Test	Andy Lurie	03.17.19	Running	2	New functionality	3,6
SBJ-18	Medium	[CI/CD] SBJ-38	Test	Jenkins CI Pip	03.17.19	Passed	3	Stable	3,5
SBJ-13	Medium	[CI/CD] SBJ-39	Test	Jenkins CI Pip	03.17.19	Passed	3	New functionality	2,99
SBJ-39	Medium	[CI/CD] SBJ-39	Test	Jenkins CI Pip	03.17.19	Fail	2	Automatic	2,1
SBJ-38	Low	[MANUAL] SBJ-8M	Test	Menister Alfon	03.17.19	Running	1	New functionality	1,1
SBJ-26	Medium	[CI/CD] SBJ-40	Test	Jenkins CI Pip	03.18.19	Passed	1	Manual	1
SBJ-27	High	[DAYLY] SBJ-03.18D	Test	DailyBuild Server Pip	03.18.19	Fail	1	Automatic	1

Fig. 5. UI of the RT selection software package

This software package was used to create a pool of manual regression tests for two different IS development projects on a daily basis, which made it possible to quickly assess the potential IS regression in the conditions of daily updating of the system modules.

The effectiveness of this method was evaluated by comparing the RT results (the number of errors found), which were formed by manual selection of test scenarios and using a software package for IS versions similar in number and volume of functional growth, for the same RT period (10 working days), on an identical number of tests (103 tests).

For a multi-module IS that evaluates the quality of the provided broadband data and TV content service (the number of modules is more than 70) with a high functional increase (over the period of 10 days, changes were made to the code of 22 modules), the number of detected errors detected by RT using an automated selection method increased by 11%.

For a modular IS that controls the car's multimedia information system (the number of modules is 8) with an average functional increase (over a period of 10 days, changes were made to the code of 3 modules), the number of detected errors detected by RT using an automated selection method increased by 6% (Fig. 6).

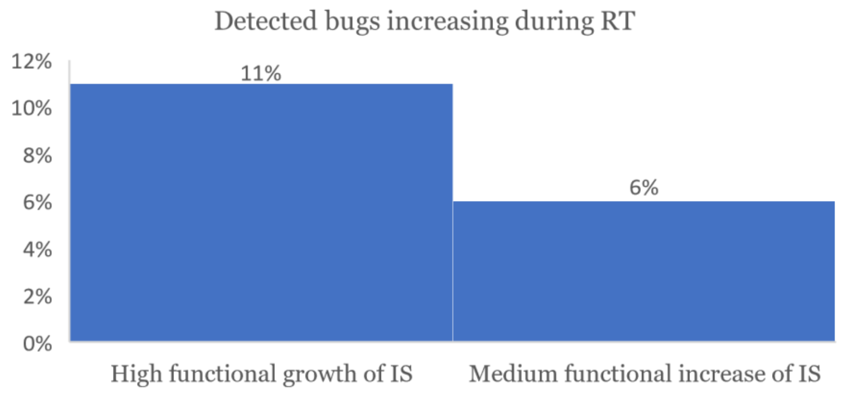


Fig. 6. Increase in detected errors when using the selection method

At the same time, the time spent on forming the RT pool was reduced by 2 times when using the automated selection method – from three hours to one and a half hours (in total – when forming the RT pool daily).

6 Conclusions

This study has shown the possibility of applying the dynamic selection method of regression tests for automated testing of graphical interfaces of multi-module information systems. Integration of this methodology into existing IS development projects provides the following advantages:

1. Allows to significantly reduce the time spent on the selection of automated RT;
2. Allows to perform automated RT on a priority basis, the results of which can be concluded with a high degree of probability that there are errors in the most priority IS modules from the point of view of the UI;
3. Allows to take into account the potential mutual influence of different modules in a multi-module IS when performing automated RT;
4. Allows to save the use of test infrastructure resources by prioritizing the implementation of automated test scenarios.

References

1. Kuznetsov, S.D. Great Russian encyclopedia. https://bigenc.ru/technology_and_technique/text/3444940, accessed 20-September-2020.
2. Terry, R., O'Connor, R.V., Dorling, A.: An Agile Implementation within a Medical Device Software Organization. Software Process Improvement and Capability Determination. Communications in Computer and Information Science. Springer (2015).
3. Larman, C., Basili, V.R.: Iterative and Incremental Development: A Brief History. Computer (2003).
4. Savin, R.: Testing Dot Com or a Manual on how to abuse bugs in Internet startups, “Delo”, Moscow (2017).
5. Ambler, S., Lines, M.: Choose Your WoW!: A Disciplined Agile Delivery Handbook for Optimizing Your Way of Working (WoW). Disciplined Agile, Inc. (2019).
6. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: A survey, King`s College London (2007).
7. Hohhof, B.: Developing Information System for Competitive Intelligence Support. Intelligent Information, LIBRARY TRENDS, Vol. 43, No. 2 (1994).
8. Golze, A., Sarbiewski, M., Zahm A.: Quality optimization to achieve high business results. Third edition. Wiley Publishing, Inc., Indianapolis, IN. (2008).
9. Duvall, P., Matyas, S., Glover, A.: Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series). Wiliams (2008).
10. Filinskikh, A. D., Merzlyakov, I. N. Evaluating geometric models based on the structure of their parameters. In: Information-measuring and control systems T. 13 (3), 69-74, (2015).
11. Richards, M.: Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media (2020).
12. Filinskikh, A. D.: The information metric is the transmission and recovery of geometric models in professional software environments, dissertation of the candidate of technical Sciences, Nizhny Novgorod (2013).
13. Colborne, G.: Simple and Usable web, mobile, and interaction design 2nd edition. New Riders, (2018).
14. Mobile Ui/Ux Design Notebook: Mobile UI/UX Design Notebook: (Yellow) User Interface & User Experience Design Sketchbook for App Designers and Developers - 8.5 x 11. Independently Published (2019).
15. Weidman, G.: Penetration testing. No starch press. San Francisco (2014).
16. Levin, M. S.: Modular System Design and Evaluation (Decision Engineering). Springer (2014).
17. Kleppmann, M.: Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media (2017).
18. Lucsa, M.: Kubernetes in Action. DMK Press, Moscow (2018).

19. Axelrod, A.: Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. Apress (2018).
20. Laster, B.: Jenkins 2: Up and Running: Evolve Your Deployment Pipeline for Next Generation Automation. O'Reilly Media (2018).