A Mathematical Model for Ranking High-Level User Interface Regression Tests

Ilya Zarubin [1] and Aleksandr Filinskikh [1]

[1] *Nizhny Novgorod State Technical University n. a. R.E. Alexeev, Minina 24, Nizhny Novgorod, 603950, Russia*

Abstract

A mathematical model has been developed for ranking high-level verification scenarios (manual tests) when forming a pool of regression tests. The model takes into account the significance of selection methods based on the analysis of previous runs of regression tests, the opinion of the expert group, the specifics of the changes made in the current release, and also has the ability to use an arbitrary number of additional test selection methods. The model can be used to analyze historical data of regression test runs using neural networks in order to identify the most effective approaches to selecting tests for regression testing. The model can be implemented in a software package that interacts with various testing management systems in order to significantly accelerate the formation of a pool of regression tests with a different approach to selection, which can be used by a test engineer of average qualification and without a deep understanding of the features and architecture of the information system being developed.

Keywords

regression testing, methods for selecting regression tests, testing multicomponent information systems, forming a pool of regression tests.

1. Introduction

The process of developing information systems (IS) [1] is a complex multi-stage procedure. The resulting IP may have specific features - for example, due to the scope of application of this IP. At the same time, most IS should be reliable, fault-tolerant and easy to use. To ensure these IS parameters, there are specific approaches to the implementation of IS development projects, such as the Waterfall model [2] and the V-model [3]. In conditions of high competition in the market of developers IS, as well as to obtain a working prototype of the system in the shortest possible time, an Iterative model [4], a Spiral model [5], a RAD model [6], Agile [7] and so on can be used. At the same time, in each of these models, considerable attention is paid to the testing procedure [8].

The main task of the testing procedure is to detect and log errors. In the future, analyzing the errors found, it can be concluded that the IS is ready to be transferred to the customer or end user for operation. During the period of active development and operation of the IS, a large number of methods for finding errors in the system were developed, depending on the specifics of the IS, its scope of application, display, complexity, architecture, and so on. Not only working IS prototypes can be tested, but also isolated modules [9], documentation [10], development processes [11]. Correct and timely application of testing procedures reduces the time and cost of development and increases the attractiveness of the IS from the user's point of view and, consequently, increases the commercial success of the developed IS.

GraphiCon 2021: 31st International Conference on Computer Graphics and Vision, September 27-30, 2021, Nizhny Novgorod, Russia
EMAIL: zarubin@nntu.ru (I. Zarubin); fad@nntu.ru (A. Filinskikh)
ORCID: 0000-0002-4870-6171 (I. Zarubin); 0000-0003-3826-6771 (A. Filinskikh)

Currently, the most popular model of IS development is Agile, which allows you to flexibly approach the formation of the order of implementation of IS capabilities both at the start of the project and in the process of its execution. Iterations ("sprints") are used for gradual functional growth of the IS [12], which usually last about two weeks. At the same time, within one iteration, all the necessary testing procedures must be performed, such as smoke [13], integration [14], functional [15], regression [16] types of testing. In addition, depending on the specifics of the system, you may need to perform other types of testing, such as performance testing [17] or security testing [18]. It should be noted separately that in the conditions of extremely tight deadlines for the development of modern IS and, often, a lack of qualified personnel to perform full-fledged testing, a situation may arise when all the necessary types of testing cannot be performed in a timely manner. For example, smoke testing, if there are test scenarios, can be performed by an engineer of low qualification. At the same time, the list of tests for regression testing (RT) must first be correctly formed, which cannot be performed by a test engineer with insufficient qualifications and who does not have a deep understanding of the specifics and architecture of the developed IS.

In the process of creating a pool of regression tests, three main tasks are solved [19]. Within the framework of solving the problem of selecting scenarios for RT, the following approaches are known:

- integer programming approach [20];
- Data-flow analysis approach [21];
- symbolic execution approach [22];
- dynamic slicing based approach [23];
- graph-walk approach [24];
- textual difference approach [25];
- the approach of dividing the graph of system dependencies (SDG Slicing approach) [26];
- Path analysis [27];
- Firewall approach [28];
- Cluster Identification [29].

Unfortunately, almost all of these approaches are focused on analyzing the IC code and are of little use for test engineers who search for errors from the point of view of the graphical user interface and operate high-level tests. The selection of tests can be performed separately using test filtering in test management systems [30], based on the analysis of the results of previous launches [31] or based on the opinion of an expert group. To increase the speed of forming a pool of regression tests and reduce the number of missed scenarios for RT, with which it would be possible to detect an error, it is necessary to develop a methodology that would take into account an arbitrary number of test selection parameters for RT in terms of previous launches, newly made changes to the IS, the opinion of the expert group, and could also be freely scaled using additional test selection approaches.

2. The methodology of forming the RT pool based on the analysis of changes in the UI

In the conditions of using the "continuous integration/continuous delivery" approach (CI/CD) [32] in the process of developing IS and the absence of" code-freeze " [33], there is a need for rapid (literally – within a few minutes) formation of a pool of RT that is relevant at the moment – with a certain list and configuration of components that would take into account the changes made, take into account the potential mutual impact of the changes made, and so on. The formed pool of tests should be ranked in such a way that at the beginning of the list there are tests, the implementation of which will allow us to detect errors in the most critical areas of the IS as efficiently as possible and, therefore, to make a conclusion about the quality of the IS as quickly as possible. Thus, the tests performed at the beginning of regression testing are more valuable for making a decision about the readiness of the IS for transfer to the customer, while continuing to perform RT with each test will be less important in terms of the importance of the detected errors (Fig. 1).

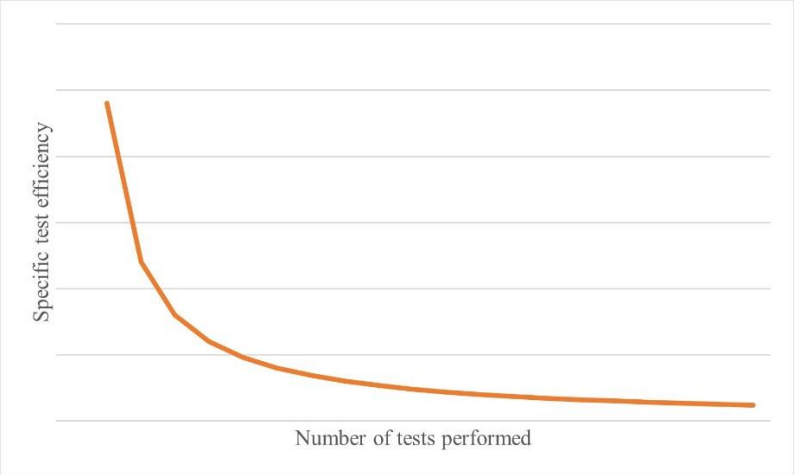


Figure 1: Performance of ranked RT

For the selection of high-level regression tests, the following approaches are most often used:

- according to the importance of the test (high/medium/low);
- by the number of errors detected by this test;
- by the date of the last launch;
- by duration of execution;
- by testing area;
- by the degree of updating of the component/code area.

Depending on the specifics and architecture of the developed IS, the list of methods that can be used for selecting tests for RT may vary, for example, for microservice IS, it is very useful to use a selection method that takes into account the degree of potential impact of the changes made to the components adjacent to the microservice, while this method cannot be applied in an IS with a monolithic architecture.

For the correct formation of a ranked pool of regression tests, it is necessary to use not one of the proposed approaches, but a set of them. In this case, the significance of the test (ST) can be calculated as the sum of the test values for each of the applied selection approaches:

$$I_a = \sum(I_1 + I_2 + \cdots + I_n) \quad (1)$$

where: I_a – total ST; I_1 – ST selected by the method 1; I_2 – ST selected by method 2; I_n – ST selected by the method n.

It is obvious that the approaches to selecting tests for the RT pool are unequal and differ in efficiency from the point of view of finding errors in the developed IS. To take into account the different effectiveness of selection approaches, as well as in the case of selecting tests using approaches that rank tests by certain numerical values (for example, the number of errors detected by the test), the coefficient k is used:

$$I_a=\sum(k_1 I_1 + k_2 I_2 + \cdots + k_n I_n) \quad (2)$$

The coefficient k is selected by the expert group on the basis of a comprehensive assessment of this approach to the selection of RT from the point of view of error detection.

Another effective way to select scenarios for RT is to analyze previously performed sets of regression tests. The number of RT launches that need to be analyzed is specific to each of the projects and depends on many factors, for example, the frequency of RT launches, the amount of functional gain between RT launches, the number of other IS testing methods that are used in the process of IS development, and so on.

It should be taken into account that the analysis of previous RT launches does not take into account the specifics of the current version of the IC on which RT is planned to be launched (which modules were updated, which new functionality was introduced, which errors were fixed, etc.), therefore, it is necessary

to take into account the opinion of the expert group, which makes a consolidated decision on the value of the ST for each of the RT selection methods for the current release.

In addition, the availability of historical data on previous RT launches will make it possible to implement effective training of neural networks [34] to generate coefficients for various RT selection approaches.

The resulting ST of each of the tests can be calculated as the sum of ST for each of the ranking methods (expert group, analysis of historical data or generated using neural network technology):

$$I_t=\sum(I_h + I_e + I_{ml}) \quad (3)$$

where I_t – total ST; I_h – ST calculated using historical data;I_e – ST calculated using expert group; I_{ml} – ST calculated using neural network technology.

When using an arbitrary number of ways to rank ST, the formula (3) is transformed into the following form:

$$I_t=\sum(I_1 + I_2 + \cdots + I_n) \quad (4)$$

where I_t – total ST; I_1 – ST calculated using method 1; I_2 – ST calculated using method 2; I_n – ST calculated using method n.

3. Application of a mathematical model for calculating the significance of tests in the formation of a pool of RT

The presented mathematical model can be implemented in a software application that allows you to flexibly adjust the coefficients for each of the selection methods by an expert group (Fig. 2) and also determine the coefficients when analyzing historical data (Fig. 3).

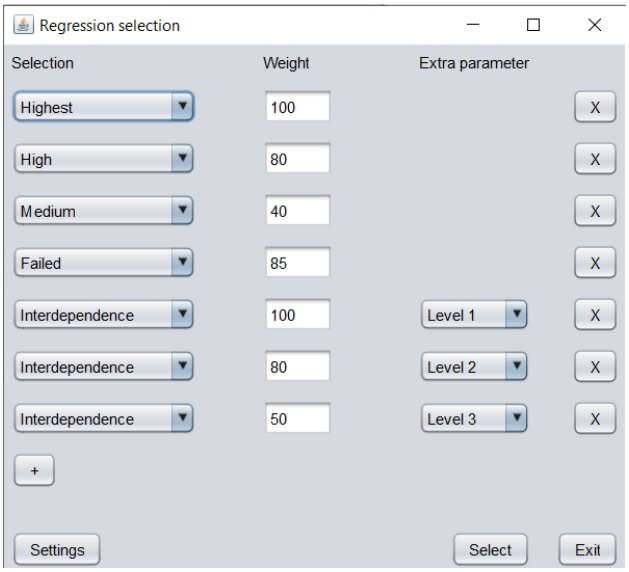


Figure 2: Form for setting parameters and weights for selecting regression tests

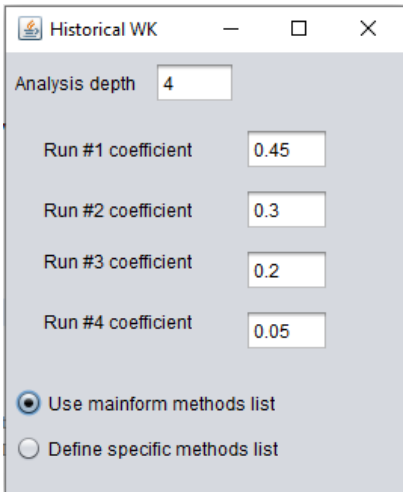
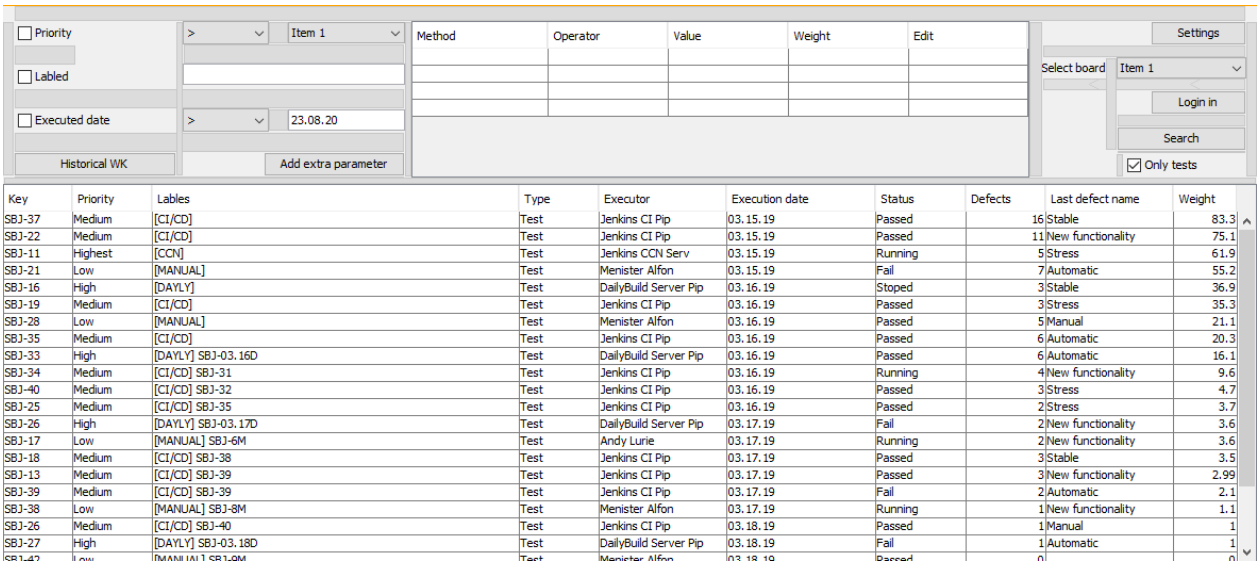


Figure 3: Setting up automated generation of weight coefficients

This application quickly forms a pool of regression tests, ranked by the significance of the test in terms of the effectiveness of finding errors in the work of the information system being developed (Fig. 4)



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

Figure 4: Ranked list of RT

The use of a mathematical model for selecting regression tests using the developed software package led to an increase in the detected errors during one regression testing session by 11% for an IC with a high functional increase during the development iteration and by 6% for an IC with an average functional increase (Fig. 5).

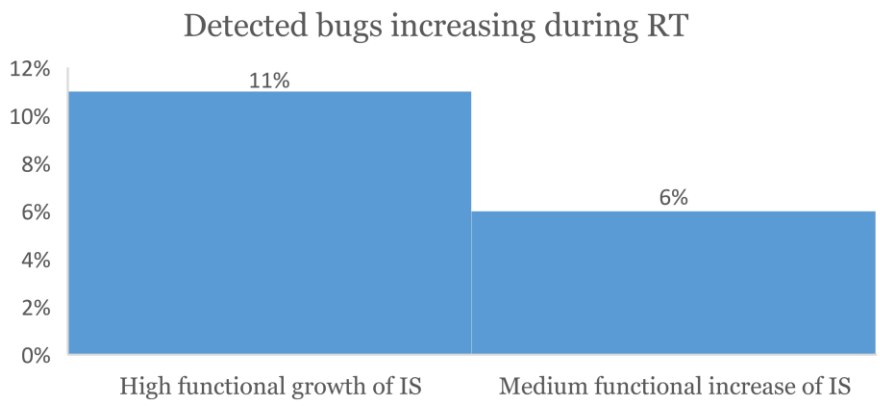


Figure 5: Increase in detected bugs

4. Conlusion

Using the proposed mathematical model of the RT pool formation allows you to quickly create a list of tests ranked in terms of the effectiveness of detecting errors in the developed IS, which can be calculated for a certain execution time (4 hours, a day, a week, and so on) and can be quickly re-formed if any significant changes are made to the code of the tested IS.

Using the application, with an implemented mathematical model of RT selection, allows you to significantly reduce the time costs for forming a RT pool, avoid mistakes when choosing tests, and also perform test selection for employees without high qualifications and a deep understanding of the architecture of the developed IS.

5. References

[1] S. D. Kuznetsov, Great Russian encyclopedia. 2004. URL: https://bigenc.ru /technology_and_technique/text/3444940. (in Russian).
[2] A. Shokoya, Waterfall to Agile – A Practical Guide to Agile Transition, TamaRe House Publishers Ltd, 2012, pp. 53-58.
[3] S. M. Staroletov, Fundamentals of software testing and verification. Textbook, Publishing house «Lan», 2018, pp. 16-18. (in Russian).
[4] C. Larman, V. R. Basili, Iterative and Incremental Development: A Brief History, Computer 36(6) (2003) 47-56.
[5] R. Höller, Rapid Application Development. diplom.de, 2001, pp. 96-102.
[6] R. Höller, Rapid Application Development. diplom.de, 2001, pp. 74-81.
[7] M. H. Martin, M. C. Fergal, C. Garret, An Agile Implementation within a Medical Device Software Organization, in: Proceedings of the Software Process Improvement and Capability Determination, Communications in Computer and Information Science, 2014, pp. 190–201. doi:10.1007/978-3-319-13036-1_17.
[8] G. Myers, T. Badgett, C. Sandler, The art of testing programs. Third edition, 2019, pp. 20-21.
[9] V. Khorikov, Principles of unit testing. A separate publication, 2021, pp. 212-228. (in Russian).
[10] G. Lynford, Internal Control Audit and Compliance. Documentation and Testing Under the New COSO Framework, John Wiley & Sons Limited, 2017, pp. 98-103.
[11] E. Scanlon Thomas, Breaking the Addiction to Process. An Introduction to Agile Development, IT Governance Publishing Ltd, 2011, pp. 87-95.
[12] G. Caldwell, Agile Project Management. The Complete Guide for Beginners to Scrum, Agile Project Management, and Software Development, SD Publishing LLC, 2020, pp. 111-121.

[13] K. Beck, Extreme programming. Development through testing, Peter, 2020, pp. 125-135.
[14] R. Savin, Dot Com testing or a Manual on the abuse of bugs in Internet startups, 2017, pp. 72-83. (in Russian).
[15] D. Efanov, V. Sapozhnikov, V. Sapozhnikov, Fundamentals of the theory of functional control of logic automation devices, OmniScriptum Publishing KS, 2018, pp. 124-129. (in Russian).
[16] V. Lipaev, Quality of software, Moscow, Finance and statistics, 1983, pp. 175-195. (in Russian).
[17] B. Erinle, Performance Testing with JMeter 3, Packt Publishing, 2017, pp. 112-115.
[18] R. Svensson, From Hacking to Report Writing. An Introduction to Security and Penetration Testing, Springer Nature Customer Service Center LLC, 2016, pp. 189-193.
[19] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: A survey, King's College London, 2007, pp. 8-12.
[20] K. Fischer, A test case selection method for the validation of software maintenance modification, in: Proceeding of International Computer Software and applications Conference. IEEE Computer Society Press, 1977, pp. 421-426.
[21] R. Gupta, M. J. Harrold, M. L Soffa, An approach to regression testing using slicing, in: Proceedings of the International Conference on Software Maintenance (ICSM 1992), IEEE Computer Society Press. 1992, pp. 299-308.
[22] S. S. Yau, Z. A. Kishimoto, Method for revalidating modified programs in the maintenance phase, in: Proceedings of the International Computer Software and applications Conference (COMPSAC 1987). IEEE Computer Society Press, 1987, pp. 272-277.
[23] H. Agrawal, J. R. Horgan, E. W. Krauser, S. A. London, Incremental regression testing, in: Proceedings of the International Conference on Software Maintenance (ICSM 1993), IEEE Computer Society Press, 1993, pp. 348-357.
[24] G. Rorhermel, M. J. Harrold, A safe, efficient regression test selection technique, ACM Transaction on Software Engineering and Methodology (1997) 173-210. doi:10.1145/248233.248262.
[25] F. Vokolos, P. Frankl, A. Pythia, A Regression test selection tool based on text differencing, in: Proceedings of the International Conference on Reliability Quality and Safety of Software Intensive Systems, Chapman and Hall, 2000. doi: 10.1007/978-0-387-35097-4_1.
[26] S. Bates, S. Horwitz, Incremental program testing using program dependence graphs, ACM Transactions on Programming Languages and Systems, 9(3) (1987) 319-349. doi:10.1145/24039.24041.
[27] P. Benedusi, A. Cmitile, U. De Carlini, Post-maintenance testing based on path change analysis, in: Proceeding of International Conference on Software Maintenance (ICSM 1988). IEEE Computer Society Press, 1988, pp. 352-361. doi 10.1109/ICSM.1988.10187.
[28] HKN. Leung, L. White, Insights into testing and regression testing global veriables, Computer Science, 1990, pp. 209-222. doi: 10.1002/smr.4360020403.
[29] J. Laski, W. Szermer, Identification of program modification and its applications in software maintenance, in: Proceeding of International Conference on Software Maintenance (ICSM 1992). IEEE Computer Society Press, 1992, pp. 282-290. doi: 10.1109/ICSM.1992.242533.
[30] R. Black, Managing the Testing Process. Practical Tools and Techniques for Managing Hardware and Software Testing, John Wiley & Sons Limited, 2019.
[31] I. B. Zarubin, A. D. Filinskikh, Automated generation of weight coefficients of selection methods when forming a pool of regression tests, in: Proceedings of the IST-2021 conference, 2021, pp 393-398. (in Russian).
[32] M. Pol, Duval, M. Stiven, III. Matias, A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series), Williams, 2008, pp. 158-164.
[33] P. Goodliffe, chapter 22: The curious case of the frozen code. Becoming a Better Programmer: A Handbook for People Who Care About Code, O'Reilly Media, Inc, 2014, pp. 195-203.

[34] D. Brown, Artificial Intelligence for Business. Understand Neural Networks and Machine Learning for Robotics. A Step-By-Step Method to Develop AI and Ml Projects for Business, 17 Books Publishing, 2020, pp. 54-59.