UDC 001.8

TECHNIQUE OF TESTING CYBER VULNERABILITIES AND QUALITY OF CYBERPHYSICAL SOFTWARE SYSTEMS

Yuriy Danyk[1], Victoriya Vysochanska[1]

[1]*National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Educational and Research Institute of Physics and Technology*

Abstract

Cyber vulnerability testing and software quality cyberphysical systems (complexes) is an important task in ensuring its reliability and security. When working with several variations of products or their versions, testing all software for every variation is resource intensive and irrational. To implement effective technological and economical quality of testing and cyber vulnerabilities of cyberphysical systems software (complexes) in terms of its increasing complexity, both in time (when considering the version) and in space (when considering variation) and lack of access to program code should be developed as follows new methods. Those methods will allow to use the results of previous tests and focus on the most important, for their testing, not yet tested parts. This is possible using regression testing methods and the appropriate choice of test cases and their prioritization to identify and address software issues and cyber vulnerabilities. Of course, testing variations and versions without access to source code, is an extremely problematic and costly task. The article analyzes the stages of regression testing and proposes an improved method for selecting test cases for testing of cyber vulnerabilities of software of cyberphysical systems (complexes) without access to program code. During the study, an analysis of the achievements in this area was conducted, investigating leading experts works. This article also identifies and compares the effectiveness of prioritized and non-prioritized test cases using the average percent detection rate (APFD). As a result of the study, new metrics for measuring test coverage are presented.

Keywords: vulnerability, testing, software, cps, cybersecurity

Introduction

Nowadays, there is a predicted future of destructive activity in cyberspace, which is characterized by diversity, complexity and asymmetry of actions, lead to chain and synergistic destructive effects in different, interconnected in cyberspace, areas. Those effects affect all aspects of daily life and national security. To prevent detection of destructive cyber effects, risk assessment and effective counteraction to cyber threats, cybersecurity systems are created. They have a lot of different levels in which, comprehensive organizational and technical measures are implemented. Development of information and cyber technologies, artificial intelligence, the Internet of Things, global informatization and widespread use of cloud services, etc. have led to the emergence of large the number of various cyberphysical systems (complexes).They also include those that are used in the national security and defence sector and at critical facilities' infrastructure. According to the current standards, cyber-physical systems (CPS) are the systems in which physical and software components are deeply intertwined by themselves. They are able to work in different spatial and temporal scales and are controlled either by computer algorithms or program codes. Their components have numerous and separate behavioural modalities and can interact with each other in a way which varies according to the context [1]. Creating and using cyberphysical systems usually have transdisciplinary approaches, merging information theory and practice technologies, cybernetics, mechatronics, design and technological processes [3, 4]. Cyberphysical systems (complexes) have a high level of integration, combination and coordination between physical and computational elements [5]. Examples of cyber-physical systems are: Internet of Things, ground, underwater, surface, air and space, robotic systems (complexes), medical monitoring complexes, industrial control and operating systems, avionics, etc. [6].

The most vulnerable elements are the components of software and hardware systems management. The effectiveness and safety of their usage depends primarily on quality and cybersecurity of their software. Therefore, the creation and development of methods and systems for testing and detecting cyber vulnerabilities and ensure its high quality and cybersecurity is an urgent task.

The aim of the study was to increase the efficiency of software testing providing cyberphysical systems in the absence of access to their software code by developing a method of regression testing of the quality and cyber vulnerabilities of their software codes based on a new approach to the selection and prioritization of test cases.

Results and discussion In modern software systems there are always many improvements, bugs corrected, new functionality added, etc. This leads to new versions software [9]. By the same principle, new properties can be added to various software variations. Thus, there is a need for periodic software testing cyberphysical systems (complexes) at all stages of their life cycle due to many reasons, the main of which are:

- integration of new functionality. Modern software systems constantly improving, providing new functionality.
- correction of various, previously undetected errors. Even already released final product may contain errors. In this case, edits are made by developers in the form of an update that allows you to fix the shortcomings.
- software adaptation. Quite often, software adapt to new components. Testing is required to verify correctness of such adaptation.

Therefore, any software changes require testing to make sure that the system has not lost its cybersecurity and still meets its requirements & specifications. Accordingly, regression testing focuses on already tested parts of software and aims to ensure the proper operation of these parts in already new adapted versions of the software.

1. Modern problems in software testing of cyberphysical systems.

There are several testing methods that help reduce the cost of testing and indicate how to choose the right test cases. To test software versions providing the most popular means to reduce efforts is the selection and prioritization of test cases [10]. However, these techniques usually require knowledge of the code [11, 14] or are based on application of models [14]. Unfortunately, complex software systems are often component, so they consist of different software components, each developed individually. Despite the fact that such components enhance the reusage of provided software and support parallel development, they cause new challenges when testing, because the source code of such components is usually not available. Only several methods deal with black box metadata, for example, the history of test cases [8, 14].

There is no method that would simultaneously include many components of testing in conditions due to lack of access to program codes. In addition, the complexity of software systems does not allow to thoroughly test each variation individually. Modern methods for testing variations mostly focus on selection [10, 11] and prioritization [11, 14] of variations or on the generation of test cases. Identifying important test cases for variations and versions of the software is difficult, as it is impossible to change code levels directly observing and analyzing. Thus, new methods are needed for solving the complexity of testing variations and versions in conditions without access to code. In the process of software development, models are often used that help to control various aspects of its creation. Examples of such models are Waterfall model, V-model or Scrum [8].

Due to the complexity of modern cyberphysical systems, testing their software requires significant resources and time. In the stages of the software product life cycle, there is a stage at which testing of part or all of the product is performed. Regardless, accordingly to which software life cycle methodology is developed, it is possible to allocate the general stages (states) and stages which are inherent in each of the methodologies. They are the basis of any process of software development and testing, [15]. Each stage of conceptual development has an analogue on the testing side, for example design system provides a specification for the appropriate stage of testing at different levels of development. In each phase of testing, the specification is used to define the set of test cases $TC = \{test_1, ..., test_n\}$ of the system being tested. Each test case covers a specific one aspect of the system under test and ensures that it meets the specifications. In this article, the two phases of testing are considered in detail - integration and system. System testing completes the verification of the system implementation. The system is defined by the set of requirements $REQ = \{req_1, ..., req_n\}$ and includes all key moments. In the course of system testing, functional testing is performed as well the characteristics of the developed software are tested, including stability, productivity, reliability and security [21].

The black box approach: system is used for system testing is considered as a single whole, real data is submitted to the input, the work of the system is analysed by the results obtained. The system testing phase reveals related errors with incorrect implementation of software functions, incorrect interaction with other systems, hardware, incorrect memory allocation, lack of correct release of resources and so on [17]. The data source is a technical task for the development of the system, specifications for its components, its environment and standards used. To assess whether the system is working according to its specifications, a set of system test cases is selected. That's right, tracking the relationship between requirements and test cases affects the quality of the system. In other words, it is important to know what requirement a test case covers in order to be able to measure certain indicators related to quality. One of the main assumptions for system testing is that each requirement is covered by at least one test case, so coverage of requirements [18], necessary to ensure the required quality.

2. Selection and prioritization of test cases.

In 2002, Rothermel [1] compared existing prioritization techniques, such as random prioritization, optimal prioritization, and lack of priorities, using the Siemens suite of applications. The optimal prioritization was possible, because the research was conducted in a controlled environment, so all errors were known. The results showed that the above method showed a higher percentage fault detection than accidental prioritization or absence. Search-based testing is now a popular area of research. Most of them focus on single-purpose regression testing, while only several authors have focused on multi-purpose regression testing.

Harman [11] notes that this type of optimization is a very important area for research. He notes that the goals for optimization are the values that should be maximized or the means that should be minimized.

In 2008 Srivastava P. [12] presented a new algorithm for prioritization of test cases to calculate the average number of errors found per minute. Using results of the APFD metric, he proved the efficiency of the presented algorithm. Calculating the effectiveness of prioritized and non-prioritized test cases was the main goal of his work.

Also, in 2011 Kavita R. [13] proposed an algorithm that calculates the speed of troubleshooting and prioritizes test cases based on this data. Experimental data showed that APFD is higher and more effective detection of serious ones occurs faults at an early stage of the testing process can be obtained using of the proposed algorithm for prioritized test cases is compared with non-prioritized.

Already in 2019, Sudhir Kumar Mukhapatra [15] proposed a new genetic algorithm for solving the problem of prioritization of test cases. The results of the experiment showed that the percentage of fault detection when using the algorithm was higher than when using the above-mentioned technique of random prioritization. It was also held experiment for fixing execution time with different number of generated test cases, and it was found that the result does not depend directly on the number of generated test cases. Testing in the absence of access to software code is very important for integration, system and acceptance testing of software cyberphysical systems, because it involves knowledge of external data and internal interface of the tested program. Thus, the control point and the observation point are outside the internal structure of the systems, f.e the tester does not have access to the code and only the observed results determine the test result [16].

Existing black box testing methods often focus on the creation process test cases, such as equivalence classes, boundary value analysis, or tables decision-making [8]. However, the performance of test cases is not trivial, especially when testing software versions, where only a certain subset of test cases can be performed due to limited resources. Software version testing is available as a complex task due to the time-limited retesting of new versions. In this context, regression testing should be considered. Regression testing focuses on retesting previously tested software parts that are subject to changes that have been made in the new version of the software provision [17]. Quality of test documentation created during system development and modified during testing will affect the cost of regression testing. If given test cases were correctly recorded and saved, duplication of effort is minimized [17]. The main problem of regression testing is the choice between full and partial retesting and replenishment of test sets. At partial repeated testing only those parts of the project that are related to the changed components are controlled [15]. Without access to the source code, to identify potentially broken system parts non-trivial task, which depends on the knowledge of experts and repetitive manual attempts. This makes the reliable identification of important regression test cases difficult and expensive, especially for their large number.

In addition, retesting is generally not possible because of the amount of effort required to the execution of all test cases may be greater than the available resources for testing, not every test case can be run again for each version. This is a problem, especially in flexible development methodology [18], where new versions are released after a short interval time and testing should be done quickly and often. Another problem arises if testing should be performed manually, as in the case of system testing, that dramatically reduces test bandwidth and forces the tester to focus on all possible subsets of test cases. Thus, regression testing is both very effective and at the same time very effective costly testing method. Highest costs and often lower quality testing associated with the execution of each test case for each function of the program after the smallest change of functionality. Problems with the correct selection of test cases can be solved with the help of their correct prioritization. Methods for prioritizing test cases include planning execution of test cases in the order that improves regression testing. Methods of prioritization of test cases allow to significantly increase efficiency performing tests in practice, namely to increase the speed of finding errors and cyber vulnerabilities. According to the empirical assessment, the order of regression test cases for the method prioritization is measured using an evaluation metric such as the average percentage measuring the quality of prioritization of test cases.

The three main approaches to regression testing are the prioritization of test cases, correct selection of test cases and minimization of test set. Therefore, it is necessary to improve correct selection and prioritization of test cases in the methodology of cyberphysical testing systems in the absence of access to program codes. Selection of test cases. When working with a large set of test cases, $TC = \{test_1, ..., test_n\}$ the correct selection and execution of a subset of test cases is a well-known approach for reduction of resources for testing [8]. We will define the problem of choosing test cases, according to Rothermel [1].

Definition. Let pr be a software system, pr' be a new version of a software system, and TC be a set of test cases. *Problem statement:* Find the set of test cases $TS' \subseteq TC$ to test pr'.

Categorization of test cases

By definition [19], there are four categories of test cases:

1) New. When adding new functionality to the system, new requirements are added. They lead to the need to create a set of new test cases: $TC_{new} \subseteq TC$ Usually new test cases have a high priority, because they have not been reproduced before, respectively, their execution must be mandatory for the current program under test.
2) Obsolete. Previously implemented functionality can be removed from the new version. For In addition, there are many obsolete test cases: $TC_{ob} \subseteq TC$, which are not part of the regression testing for the program under test, as test cases are no longer executable.

3) Reusable. A number of test cases that were performed for at least one previous version of the software and can also be run for the current one version of the program under test. This set of test cases: $TC_r \subseteq TC$ is the basis for regression testing. Although each test case from such a set is executable, their number can be very large.
4) They can be re-checked. To increase the effectiveness of regression testing, there is a subset of test cases selected from a set of multiple test cases: $TC_{re} \subseteq TC_r$. This set of test cases applies to the current program being tested and should be based on the best techniques for selecting test cases to obtain the highest probability of error detection. Set like this should include only effective ones test cases that detect new errors.

3. Solving existing problems in the effective selection of test cases.

In the past, many techniques have been proposed for the selection of test cases, most of which have a common disadvantage - their use requires access to the code and accurate information about modifications of the system [8]. Because even with the right choice of test cases and creation necessary set, their number may still be too large, there is a need to pay attention to the prioritization of selected test cases.

3.1. Prioritization of test cases.

Since the selection of test cases is intended to find a subset of tests that will be re-verified - TC_{re}. Prioritization of test cases [7] aims to prioritize a set of TC for a specific program under test according to their priorities. Formally, this is described by Elbaum [20] as a definition.

Definition. Let $TS \subseteq TC$ be a set of tests, $\mathcal{P}(TS)$ be the set of their permutations, and $f : \mathcal{P}(TC)\ to\mathbb{N}$ be a priority function. *Problem statement:* Find $TS' \in \mathcal{P}(TS)$ such that $\{\forall TS'' \in \mathcal{P}(TS) |\ TS'' \neq TS' \wedge f(TS') \geq f(TS'')\}$

One of the main advantages of prioritizing test cases over their selection is that testing can stop at any time, and by setting priorities, we ensure that the most important test cases have been completed by that point in time. The main purpose of the priority of test cases is to detect faults in the system as early as possible, by performing effective test cases first [1]. It is not possible to know the optimal number of permutations for test cases to perform, because fault information is available for the first time during the execution of test cases. However, certain approaches have been developed to achieve the goal in the fastest way. Most of them are based on the code [20], test models [8], test history and much less often, the method of functional testing.

3.2. Assessment of the quality of priorities.

Regardless of which technique is used, the desired result is early detection of faults. To evaluate, there was this goal achieved or not, we introduce the concept of the average percentage of fault detection (APFD)[7]. This metric calculates values between 0 and 1, where 1 is the best possible result. It calculates the error rate in existing test cases from a numbered list. Accordingly, the earlier the test case at position TF_i finds the i-error, the higher its priority. In this work, we measure the finding of faults, not bugs, because we do not have access to information at the code level, therefore, we can only observe manifestations of bugs in the form of malfunctions[24].

Example.

Measuring the quality of the priority of test cases Assume the following two permutations from the five test cases $TC = \{test_1, ..., test_5\}$: $TS_1 = (test_1, test_2, test_3, test_4, test_5)$

$TS_2 = (test_2, test_4, test_1, test_3, test_5)$ Now suppose that $test_2$ and $test_4$ are false test cases. Looking at the first two permutations, I see that TS_2 is the best permutation of test cases, but usually both failures can be detected by executing the first two test cases. Instead, TS_1 finds errors in another and penultimate test case. Therefore, TS_1 reaches $APFD$ 0.5, while TS_2 wins to 0.8.

$$APFD = 1 - \frac{2+4}{5 \cdot 2} + \frac{1}{2 \cdot 5} = 1 - 0.6 + 0.1 = 0.5 \quad (1)$$

$$APFD = 1 - \frac{1+2}{5 \cdot 2} + \frac{1}{2 \cdot 5} = 1 - 0.3 + 0.1 = 0.8 \quad (2)$$

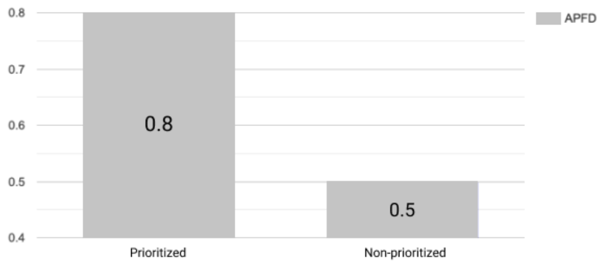

Fig. 1. Visualization of difference between a prioritized and a non-prioritized set of test cases

Therefore, TS_1 reaches the value of $APFD$ 0.5, while TS_2 leads to the value 0.8, which is shown in the figure 1.

3.3. Improving the scheme of the regression testing process

Because software updates occur frequently, regression testing of different versions of software is a complex and necessary task. In order to cope with the large volume of regression testing, there are various methods of selecting test cases. Based on the selection criteria, a subset of test cases is determined. The quality of the selection of test cases for regression depends on the test objectives, which vary depending on the software version.

In general, there are enough methods for testing software, but most of them require access to software code, which is not always possible, especially in system testing [8]. The basic scheme of the advanced regression testing process is shown in Figure 2.

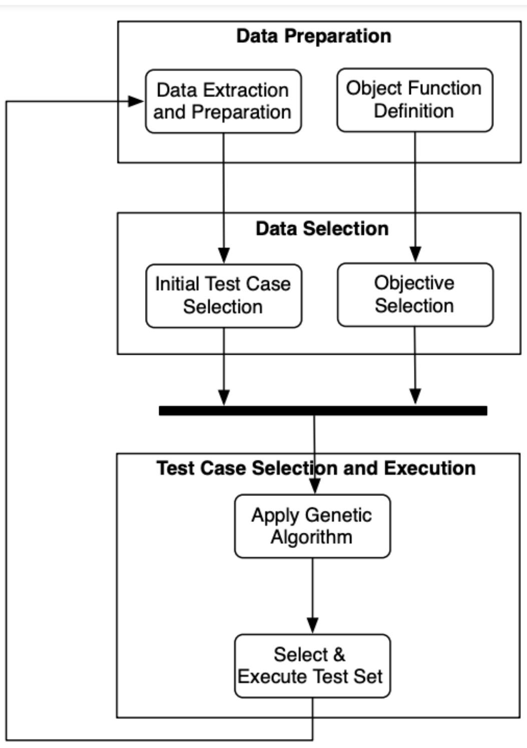
Data Preparation
Data Extraction and Preparation
Object Function Definition
Data Selection
Initial Test Case Selection
Objective Selection
Test Case Selection and Execution
Apply Genetic Algorithm
Select & Execute Test Set

Fig. 2. Scheme of multi-purpose selection of test cases

There are three main phases:

1) Preparation and analysis of data for testing
2) Data selection
3) Selection and execution of test cases

Before applying the test case selection method, it is necessary to prepare and analyse the data for testing. The available data related to system testing include:

- Covered requirements and specifications
- Faults detected
- Fault priority
- The last execution of the test case
- The average cost of running a test case

Example. *Identify test cases for regression testing.[24]*

Assume that the software architecture has changed from $\mathcal{Z}$ to $\mathcal{Z}'$ as shown in Figure 3, where the software version $\mathcal{Z}$ is shown on the left, and right - $\mathcal{Z}'$ Each version contains three components, A, B, and C. In the already tested version of Z, three connectors connect system components. Their sent signals are irrelevant for this example. So, the software has been upgraded to Z', which we can see on the right side of the picture. We see that in the new version we added a connection between A and C. The tester needs to determine which parts of the system he will test in regression testing. The re-testing approach involves a complete overview of the entire system. Other methods would most likely focus only on the difference between the proposed versions of the software, that is, in this case, on the relationship between A and C. However, the changes could affect the components A and C unpredictably, which would lead to difficulties in communicating with B. Thus, thanks to this figure, we can see all the difficulties in choosing a set of test cases $TC = \{test_1, ..., test_n\}$.

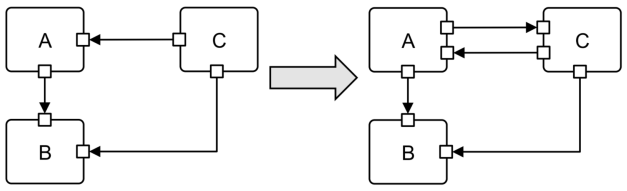

Fig. 3. Example of changes between software versions

3.4. Introduction of new metrics for measuring testing efficiency of cyberphysical software systems

One or more coverage criteria are typically used to measure how much code has been used by a set of test cases. Such criteria are determined by a set of requirements or specifications that test cases, in turn, must meet[22]

Based on the above-proven efficiency of the average percentage of fault detection, new metrics were derived, depending on such criteria of testing coverage as:

1) Operators (statement)
2) Branches
3) Functions

Accordingly, new coverage metrics were calculated:

1) Average Operator Coverage Percentage (APSC)
2) Average percentage of branch coverage (APBC)
3) Average Function Coverage Percentage (APFC)

For example, in order to calculate the APFC metric (3 point), define [23]:

Let $TC = \{test_1, ..., test_n\}$ be a numbered set of tests, with n tests, $Fn = \{fun_1, ..., fun_k\}$ be a set with k functions. APFC for such a set is calculated by the following formula:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TFn_i}{n \cdot k} + \frac{1}{2n}, n, k > 0 \qquad (3)$$

The result of this metric is between 0 and 1, where 1 is the best possible result. APSC, APBC are defined similarly to APFC, except that they measure the coverage criteria of operators and branches.

Conclusion

Conclusion

Analysis of the software testing process and testing in conditions without access to program code has shown the effectiveness of the application in conditions that methods of regression testing are considered and allowed to reveal existing problems in the selection and prioritization of test cases. Thus, the improved methodology was developed and practically applied to testing of cyber vulnerabilities and quality of cyberphysical systems (complexes) and an improved approach to the selection of test cases and analyzed the obtained results. Future research involves the development of selection methods and prioritization of test cases for risk-based testing. This article also analyzes the results of a study of predecessors on the importance of prioritizing test cases in their implementation. The difference in efficiency between the priority and non-priority sets of test cases was also determined using APFD metric comparisons. As a result of the study, new metrics for measuring test coverage were presented.

References

References

[1] Rottermel G. Elbaum S. Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering. — 2002.

[2] US National Science Foundation, Cyber-Physical Systems (CPS) — 2010.

[3] Hancu, O., Maties, V. Mechatronic approach for design and control of a hydraulic 3-dof parallel robot. // The 18th International DAAAM Symposium — 2007.

[4] Suh, S.C., Carbone, J.N., Eroglu, A.E.: Applied Cyber-Physical Systems. // Springer — 2014.

[5] Rad, Ciprian-Radu; Hancu, Olimpiu; Takacs, Ioana-Alexandra; Olteanu, Gheorghe : Smart Monitoring of Potato Crop: A Cyber-Physical System Architecture Model in the Field of Precision Agriculture // Conference Agriculture for Life, Life for Agriculture. — 2015.

[6] Khaitan et al. Design Techniques and Applications of Cyber Physical Systems: A Survey // IEEE Systems Journal. — 2014.

[7] Rottermel G., Harrold M. A Safe, Efficient Regression Test Selection Technique. - p.173-210.- 1997.

[8] Runeson P., Skoglund M. A systematic review on regression test selection techniques // Information and Software Technology. - p.14-30. - 2010.

[9] M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. - p.1-17. - 1980.

[10] M. Harman. Regression Testing Minimisation, Selection and Prioritisation : A Survey. // Wiley InterScience - 2007.

[11] M. Harman. Pareto Efficient Multi-objective Test Case Selection. // ISSTA '07: Proceedings of the 2007 international symposium on Software testing and analysis. - p.140-150. - 2007.

[12] P. Srivastava. Test Case Prioritization. // Journal of Theoretical and Applied Information Technology IEEE. - 2008.

[13] R. Kavitha. Factors Oriented Test Case Prioritization Technique in Regression Testing. // European Journal of Scientific Research. - 2011.

[14] Mondal D., Hemmati H. Exploring Test Suite Diversification and Code Coverage in Multi-Objective Test Case Selection. - p.1-10. - 2015.

[15] K. Mohapatra. A Genetic Algorithm-Based Approach for Test Case Prioritization // European Journal of Scientific Research - 2019.

[16] B. Dobran. Black Box Testing vs White Box Testing: Know the Differences. - p.1-2. - 2018.

[17] W. E. Perry. Effective Methods Of Software Testing. - 2006.

[18] R. Cole. Brilliant Agile Project Management: A Practical Guide to Using Agile, Scrum and Kanban. - 2015.

[19] Kent A., Williams J. Encyclopedia of Computer Science and Technology: Volume 32. - p.300-311. - 1995.

[20] S. Elbaum. Test Case Prioritization: A Family of Empirical Studies. // IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 28, NO. 2 - p.1-25. - 2002.

[21] I. Sommerville. Software Engineering. - 2016.

[22] Ammann P. Offutt J. Introduction to Software Testing // Cambridge University Press. — 2013.

[23] Vysochanska V. Proving the efficiency of prioritization of test cases based on APFD metrics and derivation of new metrics - 2021.

[24] Vysochanska V. Regression testing improvement in terms of black-box technique. - p.30-34. - 2021.