UDC 004.415.53 **DOI: 10.20998/2079-0023.2021.02.02**

O. V. SHEPELIEV, M. O. BILOVA

SOFTWARE TESTING RESULTS ANALYSIS FOR THE REQUIREMENTS CONFORMITY USING NEURAL NETWORKS

The relevance of scientific work lies in the need to improve existing software designed to analyze the compliance of the results of software testing of the stated requirements. For the implementation of this goal, neural networks can be used by quality control specialists to make decisions about software quality, or project managers as an expert system, for one of the quality indicators for the customer. The article deals with software testing which is a process of validation and verification of compliance of the software application or business program with the technical requirements that guided its design and development, and work as expected, and identifies important errors or deficiencies classified by the severity of the program to be fixed. Existing systems do not provide for or have only partial integration of systems of work with the analysis of requirements, which should ensure the formation of expert assessment and provide an opportunity to justify the quality of the software product. Thus, a data processing model based on a fuzzy neural network was proposed. An approach to allow determining the compliance of the developed software with functional and non-functional requirements was proposed, taking into account how successfully or unsuccessfully implemented this or that requirement. The ultimate goal of scientific work is the development of algorithmic software analysis of compliance of software testing results to stated requirements for support in the decisions taken. The following tasks are solved in scientific work: analysis of advantages and disadvantages of using existing systems when working with requirements; definition of general structure and classification of testing and requirements; characteristic main features of the use of neural networks; designing architecture, the module of research of conformity of results of testing software to the stated requirements.

Keywords: quality; requirement; testing; pipe-line; machine learning; CI/CD; Google; ANFIS.

О. В. ШЕПЕЛЄВ, М. О. БІЛОВА

ДОСЛІДЖЕННЯ ВІДПОВІДНОСТІ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗАЯВЛЕНИМ ВИМОГАМ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Актуальність наукової роботи полягає в необхідності вдосконалення існуючого програмного забезпечення, призначеного для аналізу відповідності результатів тестування програмного забезпечення заявленим вимогам. Для досягнення цієї мети можуть бути використані нейронні мережі, що буде корисно спеціалістам з контролю для прийняття рішень щодо якості програмного забезпечення або керівникам проектів як еквівалент експертної системи, що слугуватиме одним з індикаторів якості для замовника. У статті розглядається тестування програмного забезпечення, яке представляє собою процес перевірки відповідності програмного додатка або бізнес-програми технічним вимогам, які визначали особливості його проектуванням і розробки, функціонування його належним чином, а також виявлення важливих помилок або недоліків, класифікованих за їх серйозністю. Існуючі програмні продукти не передбачають або мають лише часткову інтеграцію систем для роботи з аналізом вимог, що має забезпечити формування експертної оцінки та дати можливість обґрунтувати якість програмного продукту. Таким чином, запропонована модель обробки даних на основі нечіткої нейронної мережі. Запропоновано підхід, що дозволяє визначити відповідність розробленого програмного забезпечення функціональним і нефункціональним вимогам з урахуванням того, наскільки успішно реалізована та чи інша вимога. Кінцевою метою наукової роботи є розробка алгоритмічного та програмного забезпечення відповідності результатів тестування заявленим вимогам для підтримки прийняття рішень. У науковій роботі вирішуються такі завдання: аналіз переваг та недоліків використання існуючих систем при роботі з вимогами; визначення загальної структури та класифікації тестування вимог; основні особливості використання нейронних мереж; архітектура програмного забезпечення та розробка модулю дослідження відповідності результатів тестування програмного забезпечення заявленим вимогам.

Ключові слова: якість, вимога, тестування, пайп-лайн, машинне навчання, CI/CD, Google, ANFIS.

А. В. ШЕПЕЛЕВ, М. А. БЕЛОВА

ИССЛЕДОВАНИЕ СООТВЕТСТВИЯ РЕЗУЛЬТАТОВ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ЗАЯВЛЕННЫМ ТРЕБОВАНИЯМ С ИСПОЛЬЗОВАНИЕМ НЕЙРОННЫХ СЕТЕЙ

Актуальность научной работы заключается в необходимости усовершенствования существующего программного обеспечения, предназначенного для анализа соответствия результатов тестирования программного обеспечения заявленным требованиям. Для достижения этой цели могут быть использованы нейронные сети, что будет полезно специалистам по контролю для принятия решений о качестве программного обеспечения или руководителям проектов как эквивалент экспертной системы, которая служит одним из индикаторов качества для заказчика. В статье рассматривается тестирование программного обеспечения, которое представляет собой процесс проверки соответствия программного приложения или бизнес-программы техническим требованиям, определяющим особенности его проектирования и разработки, функционирования его должным образом, а также выявление важных ошибок или недостатков, классифицированных согласно их серьезности. Существующие программные продукты не предусматривают или имеют лишь частичную интеграцию систем для работы с анализом требований, обеспечивающих формирование экспертной оценки и дать возможность обосновать качество программного продукта. Таким образом, предложена модель обработки данных на основе нечеткой нейронной сети. Предложен подход, позволяющий определить соответствие разработанного программного обеспечения функциональным и нефункциональным требованиям с учетом того, насколько успешно реализовано то или иное требование. Конечная цель научной работы - разработка алгоритмического и программного обеспечения соответствия результатов тестирования заявленным требованиям для поддержки принятия решений. В научной работе решаются следующие задачи: анализ преимуществ и недостатков использования существующих систем при работе с требованиями; определение общей структуры и классификация требований; архитектура программного обеспечения и разработка модуля исследования соответствия результатов тестирования программного обеспечения заявленным требованиям.

Ключевые слова: качество, требование, тестирование, пайп-лайн, машинное обучение, CI/CD, Google, ANFIS.

Introduction. The urgency of the work lies in the need to improve existing software designed to analyze the compliance of software testing results with the stated requirements. To achieve this goal, neural networks can be used by quality control specialists to make decisions about software quality, or by project managers as an

expert system, as one of the quality indicators for the customer. Existing systems do not provide or have only partial integration of requirements analysis systems, which should ensure the formation of expert assessment and provide an opportunity to justify the quality of the software product.

The object of research is the process of analyzing the compliance of software testing results with the stated requirements.

The subject of research: conformity analysis of the software testing results to the declared requirements with use of neural networks

The science work solves the following tasks: analysis of the advantages and disadvantages of using existing systems for the requirements analysis; determination of the general structure and classification of testing and requirements; main features characteristics of the neural networks use; design of module architecture for the conformity of software testing results to the declared requirements.

The ultimate goal of the science work is to develop algorithmic software for the software testing results analysis to the stated requirements to support decision-making.

General theory of testing. Software testing is the process of validating and verifying the compliance of a software application or program with the business and technical requirements that guided its design and development, as well as working as expected, and identifies important errors or deficiencies classified according to the severity of the program [1]. Software testing is also used to identify other software quality factors, such as reliability, usability, integrity, security, capability, efficiency, portability, maintainability, compatibility, and so on. The approach to testing differs for different programs, levels of testing, and purpose of testing.

Software testing should be conducted within budget and planning limits. Due to a large number of limitations on testing, such as comprehensive (general) testing, it is impossible to find a compromise between diligence, time, and budget – it is impossible to be sure that every mistake in the program has been eliminated [2]. Adherence to established principles can make testing simpler and more effective, as well as ensure maximum achievement of testing objectives, despite certain limitations. They also ensure the repeatability of the process. Software testing is a very important quality filter and should be planned with its goals, principles, and limitations in mind.

The purpose of testing is the predicted state of affairs that a person or system plans or intends to achieve. The goal must be achievable and measurable. It is good if all the goals are interconnected. During testing, goals can be described as the planned results of the software testing process.

Thus, software testing is a vital element of systems development life cycle (SDLC) and can provide excellent results if done correctly and effectively. Unfortunately, software testing is often less formal and rigorous than it should be, and the main reason for this is that the team has tried to identify best practices, methodologies, principles, and standards for optimal software testing. For effective and effective testing, everyone involved in testing should be familiar with the basic goals, principles, limitations, and concepts of software testing. Much work has already been done in this area and even continues today. Implementing real-world testing principles of software development to achieve testing goals to the fullest extent, given the limitations of testing, will validate research as well as pave the way for future research [3].

Types of requirements and their formation. Definition, analysis, verification, and management of requirements are constantly recognized as key areas of business analysis and are one of the target areas of software testing. Requirements are needs-oriented; solutions-oriented designs. The difference between requirements and design is not always clear. The same methods are used to identify, model and analyze both. The requirement leads to a design that, in turn, can help identify and analyze more requirements [4].

In software engineering, a functional requirement defines a system or its component. It describes the functions that the software should perform. A function is nothing but inputs, its behavior, and results. This can be computing, data processing, business process, user interaction, or any other specific functionality that determines what function the system can perform.

Functional software requirements help determine the predicted behavior of the system. Such behavior can be expressed as functions, services, or tasks or what system needs to be performed.

A non-functional requirement defines an attribute of software system quality. They are a set of standards used to assess the specific performance of the system. For example, how fast does a website load?

A non-functional requirement is important to ensure the convenience and efficiency of the entire software system. Failure to meet non-functional requirements may result in systems not being able to meet user needs.

Non-functional requirements allow you to impose restrictions or restrictions on the design of the system in different flexible lags. For example, a site should load in 3 seconds when the number of concurrent users is > 10,000. The description of non-functional requirements is as important as the functional requirement [5].

Approaches to the analysis of software compliance with the stated requirements. The purpose of verification of requirements is to ensure that the requirements and specifications of the design and models meet quality and suitability standards for the purposes they serve.

Verification of requirements ensures that requirements and designs have been correctly defined. Requirement's verification is the verification by a business analyst and key stakeholders of whether the requirements and designs are ready for verification, and provides the information needed for further work to be performed.

The most important characteristic of quality requirements and designs is suitability for use. They must meet the needs of stakeholders who will use them for a specific purpose. Ultimately, quality is determined by stakeholders [5].

Requirements (verified): a set of requirements or designs that are of sufficient quality to be used as a basis for further work.

The purpose of validation requirements is to ensure that all requirements and designs meet business requirements and support the required value provision.

Requirement validation is an ongoing process that ensures compliance with the requirements of stakeholders, decisions and the transition to the requirements of the business, and the compliance of structures with the requirements [5].

If the design cannot be verified to support the requirement, there may be a lack or misunderstanding of the requirement, or the design must change.

Approved requirements and designs are those that can be demonstrated to benefit stakeholders and agree on business goals and objectives for change. If a requirement or project cannot be verified, it either does not benefit the organization, does not fall within the scope of the decision, or both.

Algorithmic implementation of the outcome conformance analysis algorithm. From the characteristics of the subject area, we can conclude that the results obtained in the testing process are reduced to the conclusion: passed tests (pass) or not passed (fail) [6].

Such an assessment is relatively superficial and does not accurately define how well the developed software meets its requirements. Therefore, recognized as high-quality software can be a source of failure when changing the configuration or operating conditions.

The development of a method for determining the level of compliance software with functional and non-functional requirements would further provide a greater depth of measurements, and, accordingly, increase the efficiency of testing.

Since the input data for the software's assessment problem meets with the requirements of the test findings, mathematical approaches for solving it are not applicable.

Formalization of the input data of the above problem would simplify the solution and, as a result, improve the quality of the software. For formalization, it is expedient to use methods of fuzzy logical inference, which allow operating of numerical values of their belonging to the corresponding sets instead of non-numerical values.

We will assume that each requirement is met by a separate test. For the result of each test, we define its belonging to two fuzzy sets "execution" and "non-execution".

Depending on the objectives of the test, a test is considered to have been completed if all or more of half of its runs have been successful. The test is considered as failed if one, all, or more of half of its runs have failed.

Therefore, the fuzzy set of "execution" will consist of three subsets: "fully executed", "partially executed" and "more executed than not executed". The fuzzy set "non-execution" will consist of the following subsets "not executed completely", "partially executed" and "no more executed than executed".

Let's determine μ_j^i and $\mu`_j^i$. μ_j^i is the degree of affiliation of the test to each of the subsets of the set "execution", and – the ordinal number of the test, j – the number of fuzzy subset (1 – "completed", 2 – "partially failed", 3 – "more than not fulfilled"), $\mu_j^i \in [0,1]$, these degrees are determined by experts, based on the results of evaluation of test results by software developers. $\mu`_j^i$ is the degree of belonging of the test to each of the subsets of the set "failure", and – the ordinal number of the test, j – the number of fuzzy subset (1 – "not fully performed", 2 – "partially performed", 3 – more not performed than performed"), $\mu_j^i \in [0,1]$, these degrees are also determined by experts, based on the results of evaluation of test results by software developers.

To determine the general correspondence of each test to the corresponding requirement μ^i, we use the rule of difference of fuzzy sets [6]

$$\mu^i = \mu_j^i \wedge \left(1 - \left(\mu_2^i \wedge \left(1 - \mu_3^i\right)\right)\right), \mu^i \in [0,1]. \quad (1)$$

Similarly, we find the value of the general non-compliance of each test with a specific requirement

$$\mu`^i = \mu`_j^i \wedge \left(1 - \left(\mu`_2^i \wedge \left(1 - \mu`_3^i\right)\right)\right), \mu`^i \in [0,1]. \quad (2)$$

Test results are not interdependent and cannot compensate for worse values of some tests with better values of others.

The general compliance of the developed software with the requirements M can be found by the formula of the additive criterion and the difference between the sets of compliance and non-compliance, where N is the total amount of subsets

$$M = \frac{\sum_{i=1}^{N} \mu^i}{N} \wedge \left(1 - \frac{\sum_{i=1}^{N} \mu`^i}{N}\right). \quad (3)$$

The proposed method allows to determine the compliance of the developed software with functional and non-functional requirements, taking into account how successfully or unsuccessfully implemented a particular requirement, but it has a number of significant shortcomings.

First, its application can ignore important but isolated facts that do not fit into the proposed formulas.

Second, the mathematical apparatus provides only approximate calculations of compliance, as it does not take into account the nonlinear relationships between the input data of the problem of assessing the conformity of the test results of critical application software to the requirements and its initial result.

As a result, to increase the quality of analyzing the compliance of software test results with requirements, it will be more practical to design a fuzzy neural network based on the method described above [7].

The structure of the fuzzy neural network to solve the problem of assessing the compliance of the application test results to the requirements. An ANFIS (adaptive neuro-fuzzy inference system) [8] can assist us in determining the optimal distribution of membership functions by determining the mapping relation between input and output data via hybrid learning. This inference system is made up of five levels. The node function describes numerous nodes in each tier. Fixed nodes,

shown by circles, represent parameter sets that are fixed in the system, whereas adaptive nodes, represent parameter sets that are adjustable in these nodes. The current layer's input will be the output data from the preceding levels nodes. Created ANFIS structure is shown in fig. 1.

ANFIS consists of inputs, that equal to the number of tests, four layers, an output that carries out the defuzzification process.

Layers of ANFIS consist of

Layer for calculating the membership of input variables. On this layer, membership in one of the sets is calculated – the test is passed or not**.**

The second layer aggregates the prerequisite values for each rule in accordance with the selected T-norm [9].

Thus, there is compliance of the developed software with the "execution" and "non-execution" and determine how the developed software does not meet the requirements.

The diagram of activity of the developed software for research of testing conformity results of the software to the declared requirements with use of neural networks is given in fig. 2.

The starting point of the model of research of conformity of testing conformity results of the software to the declared requirements with use of neural networks is an initialization of a pipeline in GIT Actions.

Next, if the tests that were marked as required were positive, i.e. the expected result corresponds to the actual result, a webhook is passed on the software API to study

In the third layer, the values received as input are normalized; normalization is carried out using the z-normalization method [10].

The fourth layer forms the output value.

The last layer performs defuzzification. The purpose of defuzzification is to obtain the usual (not fuzzy) value of each of the output variables using the results of the accumulation of all output linguistic variables. Defuzzification is also called clarity reduction [11].

The proposed method based on a fuzzy neural network allows to eliminate this shortcoming by replacing the non-numerical results of tests with numerical values of their belonging to the corresponding fuzzy sets of requirements. If the response from the API is negative, then the next step – the deployment of the system is not possible.

If the answer from the API came in the affirmative, then the deployment of the system on the stands is possible and is performed using the last step in the system.

Overview of system functionality. The software works at the level of an automated system integrated into the life cycle of the task, or branch.

For a full understanding, let's look at the stages of the life cycle of the problem.

Backlog – the task is created and moved to the backlog, from where it gets to the stage of work.

To do – the task is moved from the backlog and taken to work.

In progress – the task is under development.

Run Autotests – the task is in the state of passing automatic tests.

For review – the task is reviewed – evaluated by another developer, for better code quality.

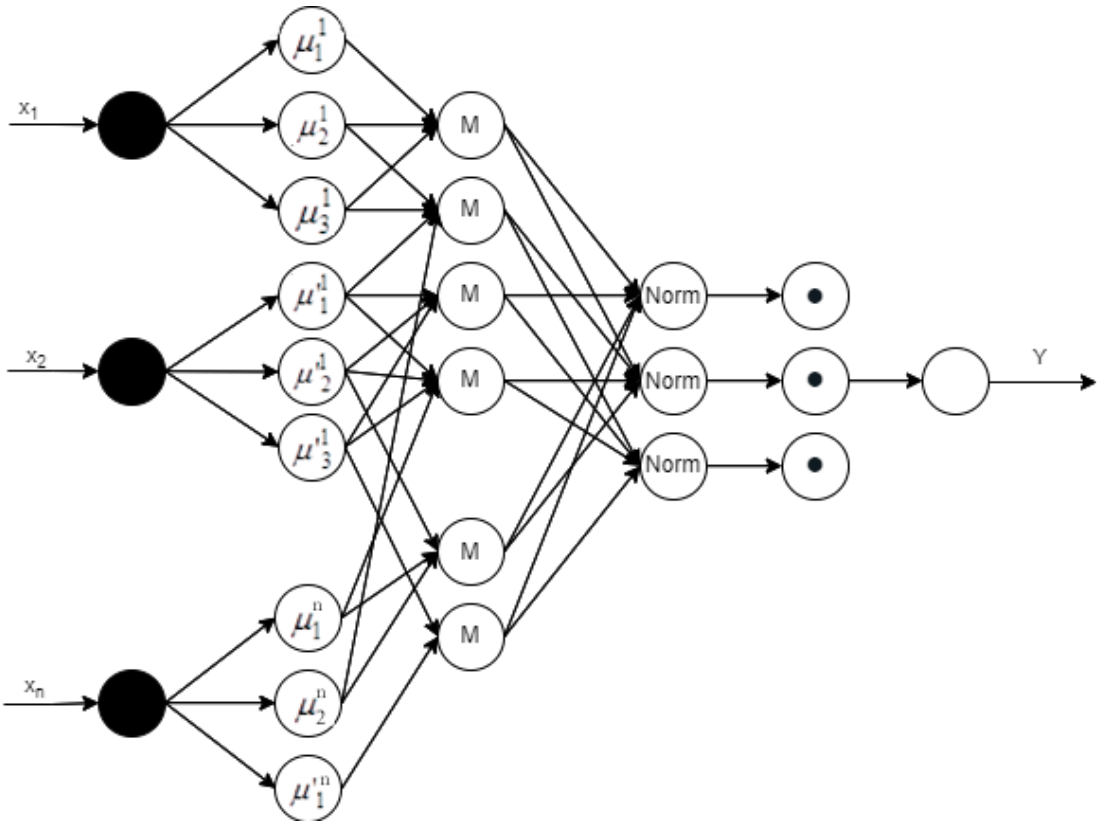

Fig. 1. The structure of a fuzzy neural network

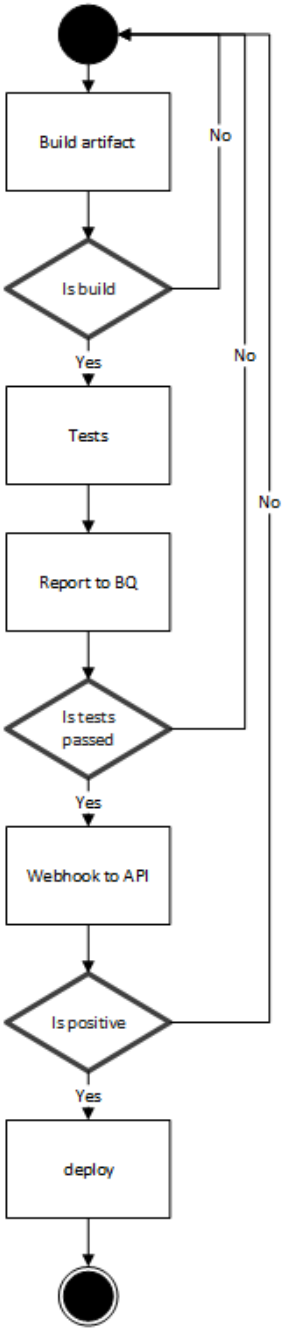

Fig. 2. Diagram of activity

For testing – the task is in a state of testing, or writing auto tests Ready for merge – a task tested, and ready for release into the product environment.

On Prod – the task is tested and is on the product environment.

Closed – the task is closed, i.e. it is executed or the decision to stop development is made.

From the reference stage it is seen that to improve the process it is necessary to implement the created software at the stages of transition from one state to another, and more precisely at the stages that are more vulnerable to system error. That is why it is advisable to implement the software in stages Run Autotests, For testing, Ready for merge, On Prod.

To better understand what is happening at these stages, we need to consider the reference process of working with a branch in the pipeline.

The CI / CD pipeline is the most important component of automated software development. Although the term has been used to describe many different aspects of computer science, git Actions and in most areas of DevOps use pipeline to illustrate the widespread use of behaviors and processes involved in continuous integration (CI).

CI is a software development strategy that increases the speed of development while ensuring the quality of the deployed code. Pipeline CI / CD is a complete set of processes that run projects start. Conveyors cover workflows, which coordinate tasks, and this is all defined in the project configuration file [12].

Examining the reference process, we can understand that automatic processes, such as unit testing, API testing, UI testing, static code analysis – requires process improvement, and therefore it is advisable to use the developed software product.

Example of the software work is given for a branch in the project. Pipeline was started by an automatic action from the bot – service on the side of the project management system.

The pipeline for a branch is based on the reference process of working with a branch in the pipeline. Pipeline is shown in fig.3.

As can be seen from the figure and based on the reference process, we see the following stages:

1. Stage of compiling and assembling docker images. There are two steps to build tests at this stage, namely the construction of the image with the solved project task.

2. Stage of passing self-tests. At this stage following is happening: static code analysis, automatic unit tests, automatic API tests, automatic UI tests, as well as the stage of analysis of passing tests and their compliance with the set of requirements.

3. Stage of data collection and import to Google BigQuery, Google Data Studio. At this stage, data is aggregated and imported into data storage and analysis systems.

Pipeline results can be found in Allure report format or in aggregate data format using Google Data Studio. Aggregate results for the period are shown in fig. 4.

Thus, an example of software operation for compliance of test results with the stated requirements was given.

Conclusions. To achieve the goal of the work, the following tasks were performed. Based on the analysis of the subject area, the advantages and disadvantages of using modern systems are determined and the conclusion is made about the need to develop algorithmic software to meet the results of software testing to the stated requirements. A data processing model based on a fuzzy neural network is proposed. A method has been developed that allows determining the compliance of the developed software with functional and non-functional requirements, taking into account how successfully or unsuccessfully implemented this or that requirement, but it has a number of significant shortcomings. Therefore, the information and algorithmic software of the solution was designed to

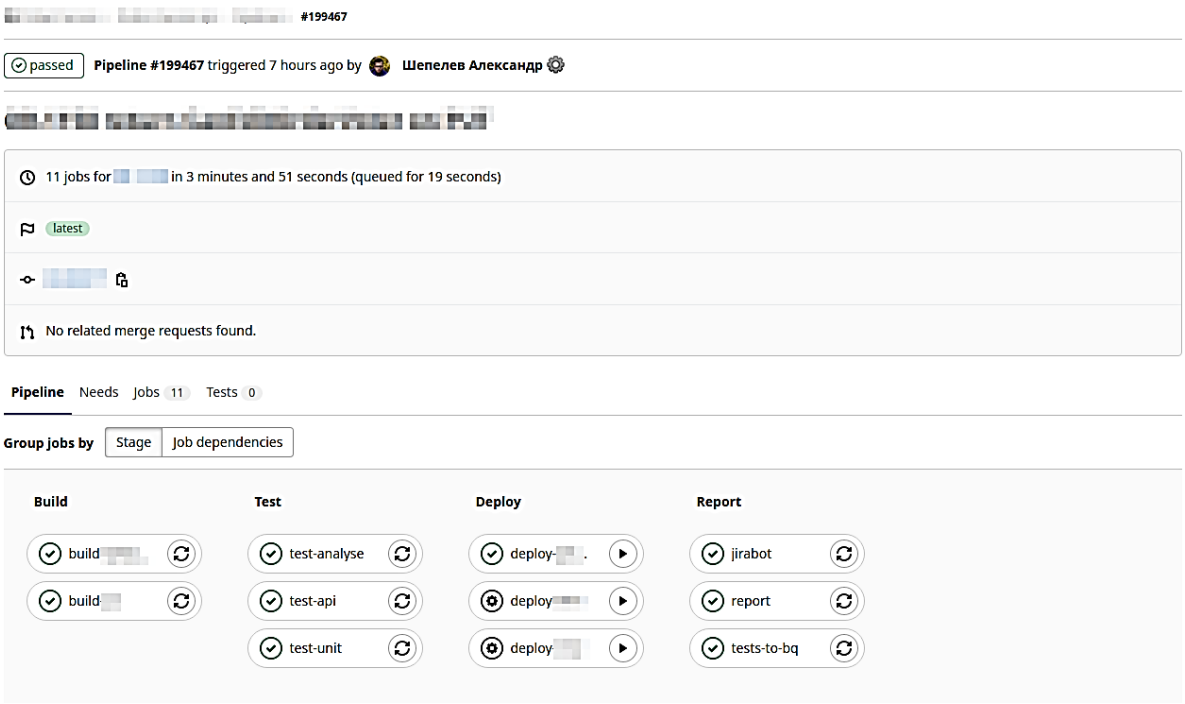

Fig. 3. Pipeline

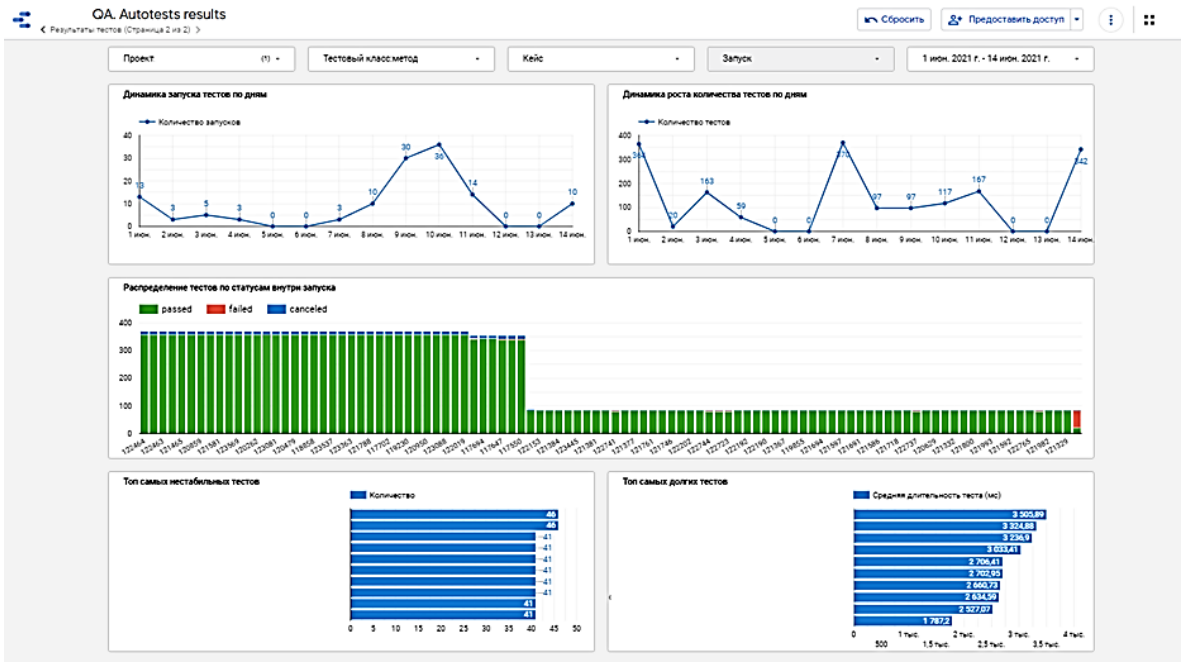

Fig. 4. Aggregate results

implement the analysis of the compliance of the software testing results with the stated requirements, which will allow forming an expert assessment of the quality of the software.

References

1. Bentley J., Bank W., Charlotte N. C. Software Testing Fundamentals – Concepts, Roles, and Terminology. *Planning, Development and Support. Available at https://support.sas.com/resources/papers/proceedings/proceedings/sugi30/141-30.pdf* (access date: 18.10.2021).
2. *Farooq Sh. U., Quadri S. M. K. Effectiveness of Software Testing Techniques on a Measurement Scale. Oriental Journal of Computer Science & Technology. 2010. № 3(1). P. 109-113.*
3. Jenkins N. *A Software Testing Primer v.2*. OPENLIBRA, 2017. 55 p.
4. Myers G. J. *The art of software testing 3rd edition*. New York: Wiley, 2011. 256 p.
5. *BABOK V3 a guide to the business analysis body of knowledge Available at*

https://book.akij.net/eBooks/2018/September/5b8a80dd494ce/BABOK_Guide_v3_Member.pdf (access date: 29.09.2021).
6. Круглов В. В., Дли М. И., Голунов Р. Ю. Нечеткая логика и искусственные нейронные сети. Москва: Физматлит, 2001. 224 с.
7. Kruse R. *Fuzzy neural network. Available at http://www.scholarpedia.org/article/Fuzzy_neural_network* (access date: 11.09.2021).
8. Jang JSR. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Trans Syst, Man, Cybernet.* 1993. № 23(3). P. 665–685.
9. Beg, Ismat & Ashraf, Saminax. *Similarity measures for fuzzy sets*. Applied and Computational Mathematics. 8. 2009 192-202.
10. DevGoogle. *Data Preparation and Feature Engineering for Machine Learning. Available at* https://developers.google.com/machine-learning/data-prep/transform/normalization#z-score (access date: 11.09.2021).
11. Chakraverty S., Sahoo D.M., Mahato N.R. Defuzzification. In: Concepts of Soft Computing. *Springer*, 2019 Singapore. *Available at* https://doi.org/10.1007/978-981-13-7430-2_7 (access date: 10.10.2021).
12. Fowler, M, Foemmel M. *Continuous integration*. (2006). *Available at* https://moodle2019-20.ua.es/moodle/pluginfile.php/2228/mod_resource/content/2/martin-fowler-continuous-integration.pdf (access date: 10.10.2021).

References (transliterated)

1. Bentley J.., Bank W., Charlotte N. C. Software Testing Fundamentals – Concepts, Roles, and Terminology. *Planning, Development and Support. Available at https://support.sas.com/resources/papers/proceedings/proceedings/sugi30/141-30.pdf* (access date: 29.09.2021).
2. *Farooq Sh. U., Quadri S. M. K. Effectiveness of Software Testing Techniques on a Measurement Scale. Oriental Journal of Computer Science & Technology. 2010. № 3(1). pp. 109-113*.
3. Jenkins N. *A Software Testing Primer v.2*. OPENLIBRA, 2017. 55 p.
4. Myers G. J. *The art of software testing 3rd edition*. New York: Wiley, 2011. 256 p.
5. *BABOK V3 a guide to the business analysis body of knowledge Available at* https://book.akij.net/eBooks/2018/September/5b8a80dd494ce/BABOK_Guide_v3_Member.pdf (access date: 29.10.2021).
6. Kruglov V.V., Dli M.I., Golunov R.Yu. Fuzzy logic and artificial neural networks. Moscow: Fizmatlit, 2001. 224 p.
7. Kruse R. *Fuzzy neural network. Available at http://www.scholarpedia.org/article/Fuzzy_neural_network* (access date: 11.09.2021).
8. Jang JSR. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Trans Syst, Man, Cybernet.* 1993. no 23(3). pp. 665–685.
9. Beg, Ismat & Ashraf, Saminax. *Similarity measures for fuzzy sets*. Applied and Computational Mathematics. 8. 2009 192-202.
10. DevGoogle. *Data Preparation and Feature Engineering for Machine Learning. Available at* https://developers.google.com/machine-learning/data-prep/transform/normalization#z-score (access date: 11.09.2021).
11. Chakraverty S., Sahoo D.M., Mahato N.R. Defuzzification. In: Concepts of Soft Computing. *Springer*, 2019 Singapore. *Available at* https://doi.org/10.1007/978-981-13-7430-2_7 (access date: 10.10.2021).
12. Fowler, M, Foemmel M. *Continuous integration*. (2006). *Available at* https://moodle2019-20.ua.es/moodle/pluginfile.php/2228/mod_resource/content/2/martin-fowler-continuous-integration.pdf (access date: 10.10.2021).

Received 03.11.2021

Відомості про авторів / Сведения об авторах / About the Authors

Шепелєв Олександр Вадимович – бакалавр технічних наук, студент, Національний технічний університет «Харківський політехнічний інститут», студент кафедри Програмної Інженерії та Інформаційних Технологій Управління; м. Харків, Україна; ORCID: https://orcid.org/0000-0002-6258-3446; e-mail: zirgus1@gmail.com

Білова Марія Олексіївна – кандидат технічних наук, Національний технічний університет «Харківський політехнічний інститут», доцент кафедри Програмної Інженерії та Інформаційних Технологій Управління; м. Харків, Україна; ORCID: https://orcid.org/0000-0001-7002-4698; e-mail: missalchem@gmail.com

Шепелев Александр Вадимович – бакалавр технических наук, студент, Национальный технический университет «Харьковский политехнический институт», студент кафедры Программной Инженерии и Информационных Технологий Управления; г. Харьков, Украина; ORCID: https://orcid.org0000-0002-6258-3446; e-mail: zirgus1@gmail.com

Белова Мария Алексеевна – кандидат технических наук, Национальный технический университет «Харьковский политехнический институт», доцент кафедры Программной Инженерии и Информационных Технологий Управления; г. Харьков, Украина; ORCID: https://orcid.org/0000-0001-7002-4698; e-mail: missalchem@gmail.com

Shepeliev Oleksandr Vadymovich – Bachelor of Technical Sciences, Student, National Technical University “Kharkiv Polytechnic Institute”, student at the Department of Software Engineering And Management Information Technologies; Kharkiv, Ukraine; ORCID: https://orcid.org/0000-0002-6258-3446; e-mail: zirgus1@gmail.com

Bilova Mariia Oleksiivna – PhD, National Technical University «Kharkov Polytechnic Institute», Associate Professor of the Department of Software Engineering And Management Information Technologies; Kharkiv, Ukraine; ORCID: https://orcid.org/0000-0001-7002-4698; e-mail: missalchem@gmail.com