https://doi.org/10.30857/2786-5371.2025.5.1

Received: 11.09.2025
Revised: 14.10.2025
Accepted: 22.10.2025

УДК 004.415.2:004.8

Artem ANTONENKO[1], Oleksandr GOLUBENKO[2], Andrii ONYSKO[3], Serhii CHECHYK[4], Sergii VOSTRIKOV[4]

[1] National University of Life and Environmental Sciences of Ukraine, Kyiv, Ukraine
[2] Higher Education Institution "Academician Yuri Bugay International Science and Technical University", Kyiv, Ukraine
[3] National Technical University of Ukraine "Igor Sikorskykyiv Polytechnic Institute", Ukraine
[4] State University of Information and Communication Technologies, Kyiv, Ukraine

PROSPECTS OF USING AI FOR AUTOMATED SOFTWARE TESTING

Purpose. *The research method is to analyze the prospects for using artificial intelligence to overcome key limitations of traditional automated software testing, in particular, test fragility, low adaptability, large volumes of test scenarios and the inconsistency of testing speed with the pace of modern development, as well as to justify the transition to intelligent, self-learning quality assurance systems.*

Methodology. *The work uses the method of systematic review and synthesis of modern scientific literature and publications of leading technology companies, a comparative analysis of traditional and AI-oriented approaches to testing, general practical use of examples using large language models, reinforcement learning, graph neural networks and self-healing technologies, as well as illustrative modeling of the benefits of AI through schematic diagrams and examples.*

Findings. *The study identified six main problems of classical testing and demonstrated that the integration of AI allows you to automatically generate test scripts (including from natural language), predict defects, implement self-healing mechanisms, optimize test priority in CI/CD, detect flaky tests, and significantly increase code coverage and process stability while simultaneously reducing support costs. autotests*

Originality. *The novelty lies in the systematic synthesis and updating of trends in the use of large language models for generating tests based on natural language, graph neural networks for dependency analysis, as well as reinforcement learning methods for adaptive navigation in interfaces, which simultaneously contribute to the formation of the concept of proactive, intelligent, and context-oriented software testing. provision*

Practical value. *The results of the work have direct practical value for QA teams and DevOps processes: the proposed AI approaches allow you to reduce time and resources for supporting automated tests, accelerate CI/CD cycles, increase the reliability of releases, optimize the focus of testing on risky areas of the code, and ensure the availability of processes even for a team with limited technical resources*

Keywords: *software testing; test automation; artificial intelligence; machine learning; test generation; self-healing; defect prediction; quality assurance; neural networks; CI/CD.*

Introduction. The rapid development of digital technologies, the growth of complexity of software systems and the increase in software quality requirements have significantly transformed testing approaches in modern industry. If previously quality assurance was based mainly on manual tests and classic automation tools, now this is not enough for the successful functioning of high-loaded, dynamic and multi-component systems. Modern products process huge amounts of data, work in cloud environments, adapt to different platforms and devices, and are also in a state of constant updates. In such conditions, there is a need for more powerful, flexible and self-learning testing mechanisms. Traditional automated tests, despite their popularity, have significant limitations. They are often fragile – any change in the structure of the UI, API or program modules can cause massive test failures. Supporting such tests requires significant resources, and the speed of response to changes in the code base increasingly does not meet the requirements of CI/CD cycles. In addition, the growth of system complexity leads to an increase in the number of test scenarios, which are difficult to fully cover even with a large QA team. Thus, there is a need for new tools that can understand the logic of the software, adapt to its changes and make decisions on the analysis, generation and optimization of tests. Artificial intelligence (AI) opens up fundamentally new opportunities for automated software testing. Unlike classical methods, AI can analyze large arrays of telemetry, code, logs and test results, detect hidden patterns, form predictive models for potential defects and automatically build optimal test strategies. Machine learning and deep learning models demonstrate the ability to independently generate tests, identify critical scenarios, adapt UI selectors, analyze the causes of test failures and perform intelligent prioritization. Self-healing testing technologies are becoming especially widespread, when automated tests are able to independently update their elements in case of changes to the interface or application logic. Such tools significantly reduce the time spent on supporting automated tests, increase the stability of CI/CD processes, and ensure high reliability of test suites even in dynamic systems. In addition, AI is able to automatically identify atypical anomalies in logs, predict risky areas of code, and identify areas that require enhanced testing. With the advent of large language models (LLMs), such as GPT-4/5, PaLM, LLaMA, etc., opportunities have opened up for generating tests based on natural language. Engineers can form requirements or a description of system behavior in the form of text, and AI can create full-fledged test cases or scripts. This significantly speeds up the testing process and makes it more accessible even to new specialists.

In general, integrating AI into testing processes not only increases the accuracy and quality of checks, but also promotes the transition from reactive methods to proactive ones. The models are able to not only detect defects after they occur, but also predict their occurrence, helping the development team prevent errors before the code is implemented. This creates a foundation for building intelligent, self-learning quality assurance systems, which leaves significant potential for further research and practical implementations.

Setting of the task. The purpose of the study is to analyze the prospects for using artificial intelligence to overcome key limitations of traditional automated software testing, in particular, test fragility, low adaptability, large volumes of test scripts and inconsistency of testing speed with the pace of modern development, as well as to justify the transition to intelligent, self-learning quality assurance systems. The main task is to identify effective approaches to:

1. Automatic generation of test scripts - creation of full-fledged test cases, unit tests, API tests and end-to-end scripts based on natural language, code and business requirements using large language models (LLM), which provides wider coverage and reduces manual labor.

2. Implementation of self-healing mechanisms - development and application of intelligent methods for automatic updating of locators, UI selectors and test logic in case of changes in the interface, API or program structure, which significantly reduces the fragility of autotests and the cost of their support.

3. Defect prediction and test prioritization – using machine learning, graph neural networks (GNN) and change history analysis, logs and telemetry to identify risky modules, predict potential errors and intelligently distribute test efforts in CI/CD processes.

4. Flaky test detection and elimination – using algorithms for analyzing logs, run history and behavioral patterns to automatically identify the causes of test instability (synchronization, network delays, time dependencies) and stabilize them.

5. Adaptive and intelligent testing – integrating reinforcement learning methods for autonomous navigation in complex interfaces, evolutionary testing for finding hidden defects, as well as AI-based combinatorial and boundary analysis, which allows you to cover atypical and rare scenarios.

This article aims to eliminate existing "blank spots" in the practical application of AI in automated testing, in particular the issues of scalability of intelligent solutions, stability of self-healing mechanisms, interpretability of forecasts and integration with existing CI/CD pipelines. In addition, the work is aimed at a systematic comparison of traditional and AI-oriented methods, which allows to determine the optimal combinations of technologies for different types of systems (web, mobile, microservices, cloud) and to provide scientifically based recommendations for QA teams, DevOps engineers and researchers in the field of software quality assurance.

Research results. The rapid development of information technologies and the scaling of software systems create a number of new challenges in the field of software quality assurance. Testing, which is one of the key stages of the software life cycle, is significantly complicated by factors related to the increase in the number of functional components, the intensity of releases, the emergence of microservice architectures and the need to support systems in continuous delivery (CI/CD) mode. In these conditions, traditional testing methods – both manual and classic automated – lose their effectiveness, as they are no longer able to provide the proper level of accuracy, speed and adaptability. One of the most important problems is the rapid increase in the volume of test scenarios. Each new function, change in logic or interface leads to the need to expand tests, as well as to periodic rewriting or updating of existing sets. In large-scale projects, the number of tests can reach tens of thousands, which creates a significant burden on QA teams and makes it difficult to maintain the stability of the test infrastructure. Test suites become cumbersome, difficult to manage, and vulnerable to any changes in the product. The second significant problem is the fragility of automated tests, primarily UI tests. Even minor changes in the DOM structure, element attributes, API responses, or system configuration cause massive errors in tests. As a result, automation, which was supposed to speed up the testing process, turns into an additional burden, and a significant part of the QA team's time is spent on eliminating errors that are not related to product defects, but only to technical changes in the interface or logic. The third problem is that classic testing tools are not able to analyze the context. They work within predefined scenarios and cannot adapt to non-standard situations not foreseen by the tester. For example, a traditional framework cannot automatically detect a change in the user's behavioral pattern, adjust the test execution logic, or predict a potential failure point. This makes the testing system inflexible and not sufficiently adaptable to the needs of the real environment. The fourth important challenge is the limited ability of QA teams to work with large arrays of logs and telemetry generated by modern software. In complex distributed systems, the volume of technical data can reach gigabytes and terabytes every day. Analyzing such data manually or even using traditional tools is often not effective enough. Some anomalies or system deviations remain unnoticed, which leads to missing critical defects or delays in their detection. The fifth problem is the discrepancy between the speed of development and the speed of testing. In modern CI/CD processes, releases can be released daily or even several times a day. However, a full run of all tests can take hours, which contradicts the concept of continuous integration. Classic automation is not able to quickly determine which tests are the most critical for this particular commit, and often executes an excessive number of scripts, wasting valuable resources. The sixth problem is the need for highly qualified specialists who create and support automated tests. Testing requires strong technical skills: knowledge of programming languages, frameworks, architectural models, principles of operation of server and client components. In the absence of such skills, high-quality automation becomes practically impossible, and the implementation of new tools becomes difficult.

Modern research in the field of artificial intelligence in automated testing demonstrates a rapid growth of interest and active development of tools that combine machine learning, deep learning, and natural language processing methods. Publications from leading technology companies – Google, Meta, Microsoft, and Amazon – confirm the effectiveness of AI in generating test scripts, predicting defects, optimizing test suites, and intelligently analyzing system logs. For example, deep learning models are used to build user behavioral models, and large language models (LLMs) demonstrate successful results in creating unit tests and tests for APIs based on text requirements. Scientific works also emphasize the importance of reinforcement learning in prioritizing tests in dynamic CI/CD processes, which allows reducing the time for regression testing without reducing quality. At the same time, a significant part of the research points to the challenges and limitations of implementing AI in testing. A number of works draw attention to the problem of insufficient interpretability of model solutions, dependence on a large amount of high-quality training data, the complexity of integrating AI tools into existing development processes, and the risks of incorrect recommendations in complex scenarios. Studies emphasize that although AI can significantly reduce the cost of supporting tests and increase the stability of automation, most technologies are at the stage of active development and require further standardization and empirical verification. In general, modern literature indicates the high prospects for the use of AI in software testing, while noting that mass industrial implementation requires a comprehensive approach, technical maturity, and the development of an appropriate infrastructure.

Modern approaches to automated software testing are increasingly integrated with artificial intelligence technologies, which allows not only to simplify the quality control process, but also to significantly increase its accuracy and efficiency. The basic idea is that AI is able to analyze huge amounts of data, draw conclusions and generate test scripts based on behavioral patterns of the system, code changes or defect history. This provides deeper and more adaptive testing that classic scripting approaches cannot provide. In this context, machine learning models and large language models are of particular value, which are able to automate the creation, optimization and prioritization of tests. Significant attention deserves the use of large language models (LLM), such as ChatGPT, Codex, Gemini or Claude, which can automatically analyze program code and generate test scripts in various formats - from unit tests to end-to-end verification scripts. Unlike classic generators, LLMs are able to take into account context, semantics and business logic, which makes them much more effective when testing complex systems. Due to the generalization effect, such models can even suggest test options that were not included in the initial templates, providing wider coverage and reducing test development time.

An important area of application of AI is comparing the effectiveness of traditional and intelligent testing methods. One of the key advantages of using AI is the ability to significantly increase test coverage in a shorter period of time. The figure below shows a comparison of the level of code coverage by tests when using a manual approach and an approach based on AI-based test generation. As can be seen from the diagram, intelligent models are able to achieve significantly higher coverage, because they analyze the code structure, identify edge cases and automatically generate positive and negative scenarios.

In addition, AI models are able to apply techniques such as boundary value analysis, combinational testing, equivalence classification and comparison of behavioral variants of the system, which traditionally required a lot of experience from the tester. For example, using models to generate test variations avoids the problem of "blind spots", where certain scenarios remain untested due to human error (Figure 1). Thus, artificial intelligence provides a more complete and systematic assessment of software functionality.

Another important direction of development is the use of evolutionary testing algorithms and symbolic execution methods in combination with machine learning. Such algorithms allow systems to automatically find complex paths of program execution that contain hidden defects, or to generate tests that can cover rare and hard-to-reach scenarios. Neural networks can learn from previous test run results to determine the most valuable areas of code for testing, increasing testing efficiency without additional resource expenditure. Modern research also demonstrates the effectiveness of reinforcement learning methods for automated navigation in complex interfaces. An agent, learning from interaction with the UI, can independently build test scenarios, discover hidden transitions, find atypical interface states, and explore functionality much deeper than is possible in scripting approaches. Such systems are able not only to repeat actions, but also to adaptively change them depending on the program's response, which makes testing more intelligent and dynamic.

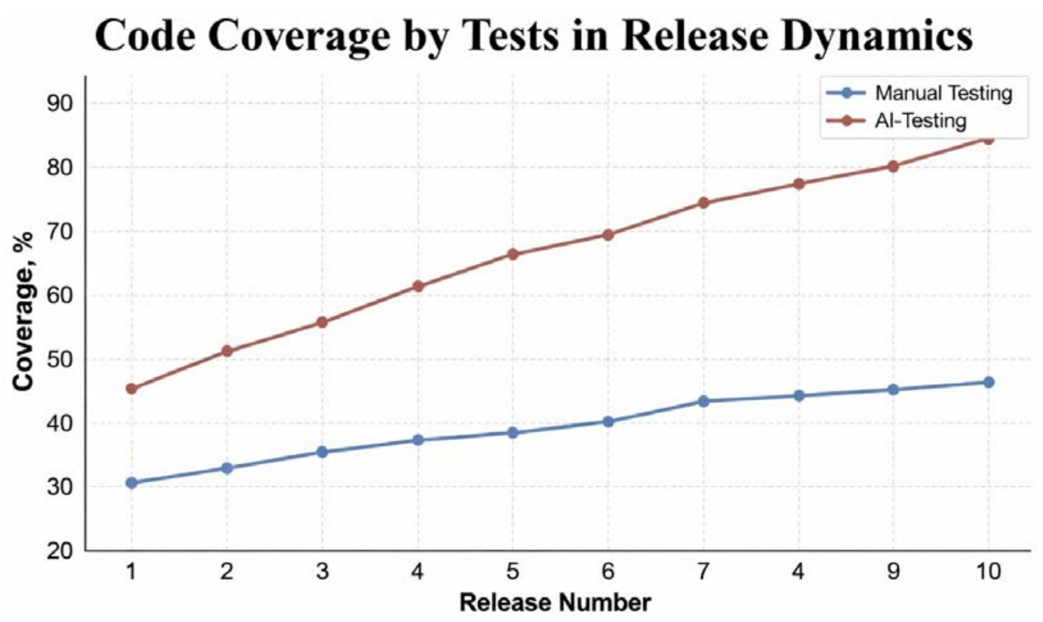

Fig. 1. **Coverage comparison, manual tests vs AI tests**

One of the most important areas is the prediction of defects in software. Machine learning allows you to analyze the history of changes in the repository, the frequency of module updates, the complexity of the code, the number of past errors, and the structure of the relationships between components. Based on this data, the models form a forecast of which modules and with what probability errors may occur in the future. This allows you to optimize the resources of the testing team, directing them to the most risky parts of the system. The text below provides an example of such a predictive model in the form of a diagram that shows the riskiness of individual modules (Fig. 2). In large technology companies, predictive models are used to automatically prioritize tests: for example, Google uses machine learning to determine which tests are most likely to find an error in a particular commit, and Microsoft - to optimize the order of test execution in large-scale systems. This allows you to significantly reduce the feedback time when testing new code changes, while maintaining high product quality. Defect prediction also helps to identify unstable modules that require refactoring or additional attention from architects.

Of particular interest is the use of Graph Neural Networks (GNN) for analyzing the architecture of software systems. Since complex applications have extensive dependencies between components, graph models allow you to identify nodes with the greatest risk of failure, find hidden cyclic dependencies, and assess the potential impact of a defect on other modules. This provides more accurate prioritization of tests and allows for more rational use of QA team resources.

A separate role in intelligent testing systems is played by self-healing mechanisms, which ensure the ability of tests to automatically adapt to changes in the user interface or API. For projects where the UI is regularly updated, such a mechanism is critically important, as it significantly reduces the number of random test failures and reduces the load on the QA team. In the figure below, you can insert a schematic representation of the self-healing process, which usually includes the stages of DOM structure analysis, search for similar elements and automatic update of locators.

Artificial intelligence is also widely used in the context of CI/CD processes. Intelligent models can determine test execution priorities, predict build duration, identify flaky tests and even automatically form optimal pipeline sequences. This makes the testing process much faster, and releases more stable. In the second figure, it is recommended to insert an illustration of a typical CI/CD architecture with the AI test analysis module enabled (Fig. 3).

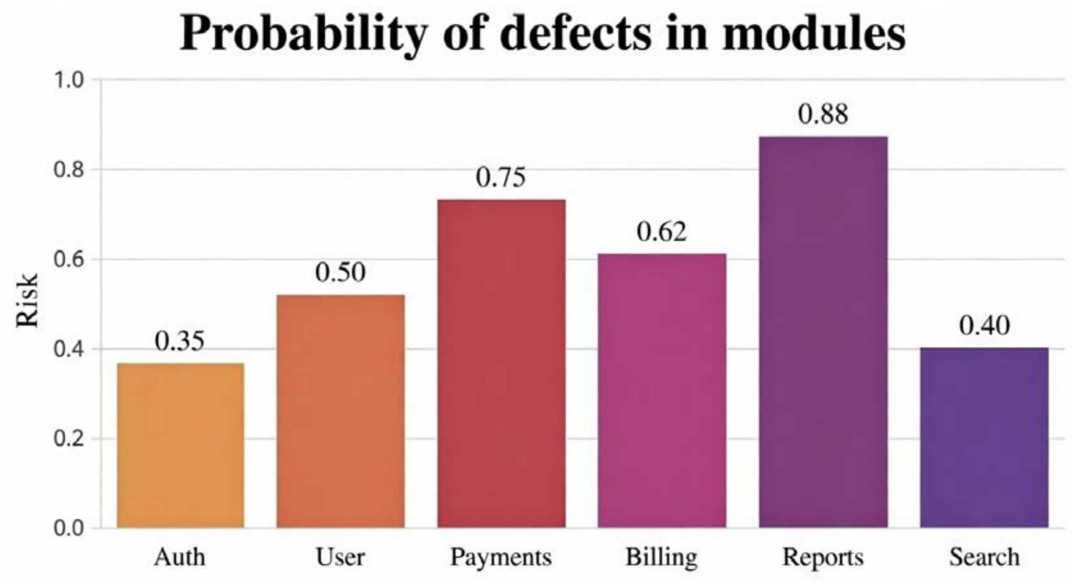

Fig. 2. **Predicting defects in modules**

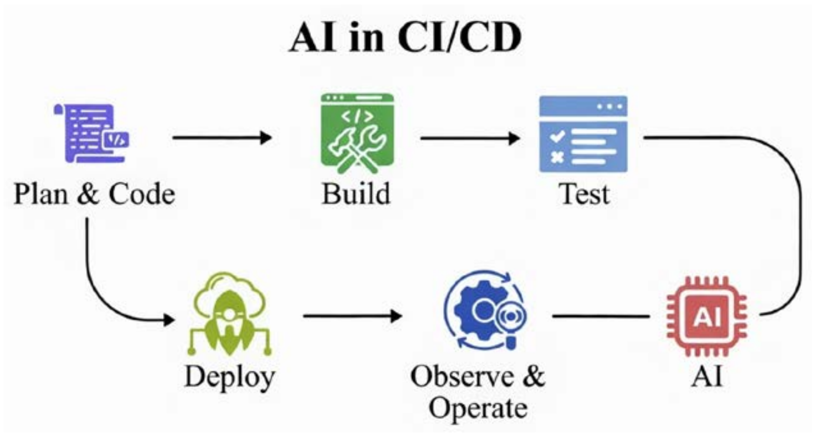

Fig. 3. **Example of a CI/CD process with an integrated AI module**

The use of artificial intelligence in CI/CD also includes optimizing the sequence of test execution. Based on historical data, models can predict which tests are most likely to detect defects in the early stages of pipeline execution, forming the most efficient order of regression runs. This allows you to significantly reduce the total testing time and minimize delays in the release process.

AI algorithms analyze the duration, stability and coverage of each test, which allows you to optimize the test suite without losing the quality of the checks. Of particular importance is the detection of flaky tests – unstable tests that pass and fail without changing the code. By analyzing logs, run history and behavioral patterns of such tests, AI models can automatically determine the causes of instability: synchronization problems, wait times, network dependencies or incorrect locators (Fig. 4). This allows you to significantly reduce noise in CI/CD processes and increase the stability of the entire pipeline, which is critically important for large development teams.

Thus, the use of artificial intelligence in the field of software testing allows not only to automate routine processes, but also to significantly increase the stability, accuracy and speed of quality control. Technologies for defect prediction, intelligent log analysis, adaptive self-healing and test prioritization demonstrate enormous potential and are already becoming key elements of modern QA platforms.

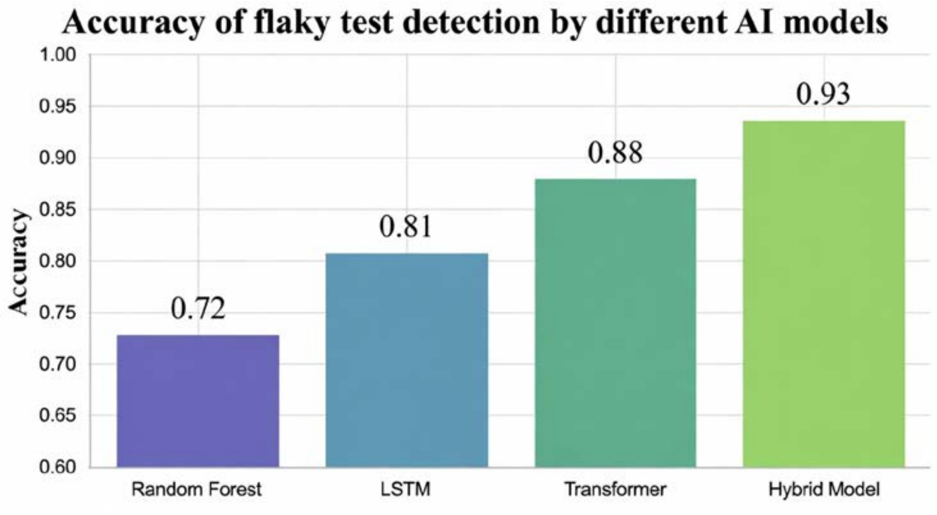

Fig. 4. **Accuracy of flaky test detection by different AI models**

Conclusions. The application of artificial intelligence technologies in the field of automated software testing opens up a new level of efficiency and intellectualization of quality control processes. Machine learning models allow you to automatically generate test scenarios, predict the appearance of defects, detect anomalies in logs and adaptively update tests using self-healing mechanisms. This significantly reduces time and resource costs, increases the stability of CI/CD processes and ensures higher reliability of software systems. However, the implementation of AI requires high-quality training data, integration with complex architectures and highly qualified specialists, which remains a challenge for many teams. Despite this, modern research and practical examples show that intelligent testing systems are becoming a key element of quality assurance and have all the prerequisites to become an industry standard in the coming years.

References

1. Ajorloo, S., Jamarani, A., Kashfi, M., Haghi Kashani, M., & Najafizadeh, A. (2024). A systematic review of machine learning methods in software testing. *Applied Soft Computing*, 162, 111805. DOI: https://doi.org/10.1016/j.asoc.2024.111805.

2. Tverdokhlib, A. O., Korotin, D. S., & Antonenko, A. V. (2023). Efektyvnist funktsionuvannia komp’iuternykh system pry vykorystanni tekhnolohii blokchein i baz dannykh [Efficiency of computer systems operation when using blockchain technology and databases]. *Tavriiskyi Naukovyi Visnyk. Seriia: Tekhnichni Nauky – Tavria Scientific Bulletin. Series: Technical Sciences*, (6), 25–36. DOI: https://doi.org/10.32851/tnv-tech.2022.6.4 [in Ukrainian].

3. Deming, C., Khair, M. A., Mallipeddi, S. R., & Varghese, A. (2021). Software testing in the era of AI: Leveraging machine learning and automation for efficient quality assurance. *Asian Journal of Applied Science and Engineering*, 10(1), 66–76. DOI: https://doi.org/10.18034/ajase.v10i1.88.

Література

1. Ajorloo S., Jamarani A., Kashfi M., Haghi Kashani M., Najafizadeh A. A systematic review of machine learning methods in software testing. *Applied Soft Computing*. 2024. Vol. 162, Art. 111805. DOI: https://doi.org/10.1016/j.asoc.2024.111805.

2. Твердохліб А. О., Коротін Д. С., Антоненко А. В. Ефективність функціонування комп’ютерних систем при використанні технології блокчейн і баз данних. *Таврійський науковий вісник. Серія: Технічні науки*. 2023. No. 6. С. 25–36. DOI: https://doi.org/10.32851/tnv-tech.2022.6.4.

3. Deming C., Khair M. A., Mallipeddi S. R., Varghese A. Software Testing in the Era of AI: Leveraging Machine Learning and Automation for Efficient Quality Assurance. *Asian Journal of Applied Science and Engineering*. No. 10(1).

4. Lemeshko, A. V., Antonenko, A. V., & Petryk, A. V. (2023). Neiromorfni systemy yak instrument realizatsii shtuchnoho intelektu [Neuromorphic systems as a tool for implementing artificial intelligence]. *Vcheni Zapysky TNU imeni V.I. Vernadskoho. Seriia: Tekhnichni Nauky – Scientific notes of the V.I. Vernadsky TNU. Series: Technical Sciences*, 34(73)(3), 175–183. DOI: https://doi.org/10.32782/2663-5941/2023.3.1/28 [in Ukrainian].

5. Fontes, A., & Gay, G. (2023). The integration of machine learning into automated test generation: A systematic mapping study. arXiv:2206.10210v5 [cs.SE]. https://arxiv.org/abs/2206.10210.

6. Gakman, D. V., & Antonenko, A. V. (2023). Osoblyvosti rozrobky ta vykorystannia freimvorkiv dlia avtomatyzovanoho testuvannia [Features of development and use of frameworks for automated testing]. *Tavriiskyi Naukovyi Visnyk. Seriia: Tekhnichni Nauky – Tavria Scientific Bulletin. Series: Technical Sciences*, (2), 21–32. DOI: https://doi.org/10.32782/tnv-tech.2023.2.3 [in Ukrainian].

7. Xie, Q., & Chen, T. (2020). Log parsing and anomaly detection with deep learning: A survey. *ACM Computing Surveys*.

8. Antonenko, A., Pakhomov, M., Kalyta, T., & Haleta, V. (2023). Using artificial intelligence in automated systems [Use of artificial intelligence in automated systems]. *Visnyk Khmelnytskoho Natsionalnoho Universytetu – Herald of Khmelnytskyi National University. Technical Sciences*, 323(4), 11–20. DOI: https://doi.org/10.31891/2307-5732-2023-323-4-11-20 [in Ukrainian].

9. Fraser, G., McMinn, P., Arcuri, A., & Padberg, F. (2015). Does Automated Unit Test Generation Really Help Software Testers? A Controlled Empirical Study. *ACM Transactions on Software Engineering and Methodology* (TOSEM), 24(4), Art. 23. DOI: https://doi.org/10.1145/2699688.

10. Zhang, J., Pan, X., & Chen, Y. (2023). Large language models for test generation: A study of ChatGPT and Codex. arXiv:2302.07090 [cs.SE]. DOI: https://doi.org/10.48550/arXiv.2302.07090.

11. Balvak, A. A., Lemeshko, A. V., Ziniar, D. A., Burachynskyi, A. Yu., Antonenko, A. V., & Prykhodko, A. P. (2024). Obrobka ta analiz danykh na prykladi naboru spambase z vykorystanniam bibliotek dlia mashynnoho navchannia [Data processing and analysis on the example of the spambase dataset using machine learning libraries]. *Tavriiskyi Naukovyi Visnyk. Seriia: Tekhnichni Nauky – Tavria Scientific Bulletin. Series: Technical Sciences*, (2),

P. 66–76. DOI: https://doi.org/10.18034/ajase.v10i1.88.

4. Лемешко А. В., Антоненко А. В., Петрик А. В. Нейроморфні системи як інструмент реалізації штучного інтелекту. *Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки*. 2023. Том 34 (73), № 3. С. 175–183. DOI: https://doi.org/10.32782/2663-5941/2023.3.1/28.

5. Fontes A., Gay G. The Integration of Machine Learning into Automated Test Generation: A Systematic Mapping Study. *arXiv*. 2023. arXiv:2206.10210v5. URL: https://arxiv.org/pdf/2206.10210.

6. Гакман Д. В., Антоненко А. В. Особливості розробки та використання фреймворків для автоматизованого тестування. *Таврійський науковий вісник. Серія: Технічні науки*. 2023. No. 2. С. 21–32. DOI: https://doi.org/10.32782/tnv-tech.2023.2.3.

7. Xie Q., Chen T. Log parsing and anomaly detection with deep learning: a survey. *ACM Computing Surveys*. 2020.

8. Антоненко А., Пахомов М., Калита Т., Галета В. Використання штучного інтелекту в автоматизованих системах. *Вісник Хмельницького національного університету*. 2023. Том 323, № 4. С. 11–20. DOI: https://doi.org/10.31891/2307-5732-2023-323-4-11-20.

9. Fraser G., McMinn P., Arcuri A., Padberg F. Does Automated Unit Test Generation Really Help Software Testers? A Controlled Empirical Study. *ACM Transactions on Software Engineering and Methodology* (TOSEM). 2015. Vol. 24, No. 4. Art. 23. DOI: https://doi.org/10.1145/2699688.

10. Zhang J., Pan X., Chen Y. Large Language Models for Test Generation: A Study of ChatGPT and Codex. *arXiv*. 2023. arXiv:2302.07090v2. DOI: https://doi.org/10.48550/arXiv.2302. 07090.

11. Балвак А. А., Лемешко А. В., Зіняр Д. А., Бурачинський А. Ю., Антоненко А. В., Приходько А. П. Обробка та аналіз даних на прикладі набору spambase з використанням бібліотек для машинного навчання. *Таврійський науковий вісник. Серія: Технічні науки*. 2024. No. 2. С. 3–20. DOI: https://doi.org/10.32782/tnv-tech.2024.2.1.

3–20. DOI: https://doi.org/10.32782/tnv-tech.2024.2.1 [in Ukrainian].

12. Tekade, P., Patil, A. A., Shriyal, S. S., Umbare, R. T., Smit, T., & Damre, S. S. (2025). Framework for automated software testing using machine learning and artificial intelligence. *Lex Localis – Journal of Local Self-Government*, 23(S4), 2654–2663. URL: https://lex-localis.org/index.php/LexLocalis/article/view/800936.

13. Chen, Y., Li, S., Dai, J. et al. (2020). NeuSight: Neural performance diagnosis in large-scale systems. In: *Proceedings of the ACM Symposium on Cloud Computing*.

14. Abadi, M., Barham, P., Chen, J. et al. (2016). TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation* (OSDI 2016). URL: https://www.tensorflow.org.

15. Antonenko, A. V., Vostrikov, S. O., Burachynskyi, A. Yu., Tverdokhlib, A. O., Balvak, A. A., & Slobodian, O. A. (2024). Osoblyvosti avtomatyzovanoho testuvannia z vykorystanniam freimvorkiv [Features of automated testing using frameworks]. *Tavriiskyi Naukovyi Visnyk. Seriia: Tekhnichni Nauky – Tavria Scientific Bulletin. Series: Technical Sciences*, (4), 3–14. DOI: https://doi.org/10.32782/tnv-tech.2024.4.1 [in Ukrainian].

16. Landauer, M., Onder, S., Skopik, F., & Wurzenberger, M. (2023). Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications*, (12), 100470. DOI: https://doi.org/10.1016/j.mlwa.2023.100470.

17. Antonenko, A., Solskyi, D., Solobaiev, S., Chechyk, S., & Cherevyk, O. (2025). Using the Scikit-learn library in machine learning classification methods. Measuring and computing devices in technological processes, (1), 468–472. DOI: https://doi.org/10.31891/2219-9365-2025-81-58.

18. Isam Al Mrayat, O., Jawarneh, M., Ibrahim, D., & Altrad, A. (2024). AI-powered software testing: A novel framework for enhancing bug detection and code reliability. *International Journal of Intelligent Systems and Applications in Engineering*, 12(23s), 1871.

19. Vorochek, O., & Solovei, I. (2024). Doslidzhennia zasobiv shtuchnoho intelektu dlia avtomatyzatsii protsesu testuvannia prohramnoho zabezpechennia [Research on artificial intelligence tools for automating the software testing process]. *Bulletin of National Technical University “KhPI”. Series: System Analysis, Control and Information*

12. Tekade P., Patil A. A., Shriyal S. S., Umbare R. T., Smit T., Damre S. S. Framework for automated software testing using machine learning and artificial intelligence. *Lex Localis – Journal of Local Self-Government*. 2025. No. 23(S4). P. 2654–2663. URL: https://lex-localis.org/index.php/LexLocalis/article/view/800936.

13. Chen Y., Li S., Dai J. et al. NeuSight: Neural Performance Diagnosis in Large-Scale Systems. *Proceedings of the ACM Symposium on Cloud Computing*. 2020.

14. Abadi M., Barham P., Chen J. et al. TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation* (OSDI 2016). 2016. URL: https://www.tensorflow.org.

15. Антоненко А. В., Востріков С. О., Бурачинський А. Ю., Твердохліб А. О., Балвак А. А., Слободян О. А. Особливості автоматизованого тестування з використанням фреймворків. *Таврійський науковий вісник. Серія: Технічні науки*. 2024. No. 4. С. 3–14. DOI: https://doi.org/10.32782/tnv-tech.2024.4.1.

16. Landauer M., Onder S., Skopik F., Wurzenberger M. Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications*. 2023. Vol. 12. Art. 100470. DOI: https://doi.org/10.1016/j.mlwa.2023.100470.

17. Antonenko A., Solskyi D., Solobaiev S., Chechyk S., Cherevyk O. Using the Scikit-learn library in machine learning classification methods. *Measuring and computing devices in technological processes*. 2025. No. 1. С. 468–472. DOI: https://doi.org/10.31891/2219-9365-2025-81-58.

18. Isam Al Mrayat O., Jawarneh M., Ibrahim D., Altrad A. AI-Powered software testing: A novel framework for enhancing bug detection and code reliability. *International Journal of Intelligent Systems and Applications in Engineering*. 2024. No. 12(23s). Art. 1871.

19. Ворочек О., Соловей І. Дослідження засобів штучного інтелекту для автоматизації процесу тестування програмного забезпечення. *Вісник Національного технічного університету "ХПІ". Серія: Системний аналіз, управління*

Technologies, 1(11), 58–64. DOI: https://doi.org/10.20998/2079-0023.2024.01.09 [in Ukrainian].

20. Karhu, K., Kasurinen, J., & Smolander, K. (2025). Expectations vs reality – A secondary study on AI adoption in software testing. arXiv:2504.04921 [cs.SE]. URL: https://arxiv.org/abs/2504.04921.

21. Wang, F., Arora, C., Tantithamthavorn, C., Huang, K., & Aleti, A. (2025). Requirements-driven automated software testing: A systematic review. arXiv:2502.18694 [cs.SE]. URL: https://arxiv.org/abs/2502.18694.

22. Antonenko, A., Lemeshko, A., & Tsyvik, O. (2023). Analiz i osoblyvosti prohramnoho zabezpechennia dlia kontroliu trafiku [Analysis and features of software for traffic control]. *Visnyk Khmelnytskoho Natsionalnoho Universytetu – Bulletin of Khmelnytskyi National University*, 1(3(321), 64–68. DOI: https://doi.org/10.31891/2307-5732-2023-321-3-64-68 [in Ukrainian].

23. Ricca, F., Marchetto, A., & Stocco, A. (2024). A multi-year grey literature review on AI-assisted test automation. *arXiv*, arXiv:2408.06224 [cs.SE]. DOI: https://doi.org/10.48550/arXiv.2408.06224.

24. Lemeshko, A. V., Antonenko, A. V., Balvak, A. A., & Novichenko, Ye. O. (2023). Aktualni zasady stvorennia alhorytmiv obrobkyinformatsii dlia lohistychnykh tsentriv [Current principles of creating information processing algorithms for logistics centers]. *Tavriiskyi Naukovyi Visnyk. Seriia: Tekhnichni Nauky – Tavria Scientific Bulletin. Series: Technical Sciences*, (1), 25–32 [in Ukrainian].

25. Fontes, A., & Gay, G. (2022). The integration of machine learning into automated test generation: A systematic mapping study. *arXiv*, arXiv:2206.10210 [cs.SE]. DOI: https://doi.org/10.48550/arXiv.2206.10210.

26. Escalante-Viteri, A. et al. (2024). Natural language processing-based software testing: A systematic review. *IEEE Access*.

27. Zaitsev, Ye. O., & Antonenko, A. V. (2022). Smart zasoby vyznachennia avariinykh staniv u rozpodilnykh elektrychnykh merezhakh mist [Smart means for determining emergency states in distribution electric networks of cities]. *Tavriiskyi Naukovyi Visnyk. Seriia: Tekhnichni Nauky – Tavria Scientific Bulletin. Series: Technical Sciences*, (5), 3–12. DOI: https://doi.org/10.32851/tnv-tech.2022.5.1 [in Ukrainian].

28. Kaminsky, O., & Demenko, I. (2023). Analiz vplyvu avtomatyzovanoho testuvannia na yakist ta bezpeku PZ [Analysis of the impact of automated testing on software quality and security. *Modeling and Information System in*

та інформаційні технології. 2024. No. 1 (11), С. 58–64. DOI: https://doi.org/10.20998/2079-0023.2024.01.09.

20. Karhu K., Kasurinen J., Smolander K. Expectations vs reality – A secondary study on AI adoption in software testing. *arXiv*. 2025. arXiv:2504.04921. URL: https://arxiv.org/abs/2504.04921.

21. Wang F., Arora C., Tantithamthavorn C., Huang K., Aleti A. Requirements-driven automated software testing: A systematic review. *arXiv*. 2025. arXiv:2502.18694. URL: https://arxiv.org/abs/2502.18694.

22. Антоненко А., Лемешко А., Цвик О. Аналіз і особливості програмного забезпечення для контролю трафіку. *Вісник Хмельницького національного університету*. 2023. Том 1, № 3 (321). С. 64–68. DOI: https://doi.org/10.31891/2307-5732-2023-321-3-64-68.

23. Ricca F., Marchetto A., Stocco A. A multi-year grey literature review on AI-assisted test automation. *arXiv*. 2024. arXiv:2408.06224. DOI: https://doi.org/10.48550/arXiv.2408.06224.

24. Лемешко А. В., Антоненко А. В., Балвак А. А., Новіченко Є. О. Актуальні засади створення алгоритмів обробкиінформації для логістичних центрів. *Таврійський науковий вісник. Серія:Технічні науки*. 2023. No. 1. С. 25–32.

25. Fontes A., Gay G. The integration of machine learning into automated test generation: A systematic mapping study. *arXiv*. 2022. arXiv:2206.10210. DOI: https://doi.org/10.48550/arXiv.2206.10210.

26. Escalante-Viteri A. et al. Natural language processing-based software testing: A systematic review. *IEEE Access*. 2024.

27. Зайцев Є. О., Антоненко А. В. Smart засоби визначення аварійних станів у розподільних електричних мережах міст. *Таврійський науковий вісник. Серія: Технічні науки*. 2022. No. 5. С. 3–12. DOI: https://doi.org/10.32851/tnv-tech.2022.5.1.

28. Камінський О. Є., Деменко І. Аналіз впливу автоматизованого тестування на якість та безпеку ПЗ. *Моделювання та інформаційні системи в економіці*. 2023.

Economics, (103), 91–103. DOI: http://doi.org/10.33111/mise.103.8 [in Ukrainian].

29. Zaitsev, I., Golubenko, O., Tkachenko, O., Pidmohylnyi, O., & Antonenko, A. (2023). Exploring advanced hypothesis generation in astronomy through the implementation of a mathematical model of linguistic neural networks. *CEUR Workshop Proceedings*, 3687, 121–128.

№ 103. С. 91–103. DOI: http://doi.org/10.33111/mise.103.8.

29. Zaitsev I., Golubenko O., Tkachenko O., Pidmohylnyi O., Antonenko A. Exploring advanced hypothesis generation in astronomy through the implementation of a mathematical model of linguistic neural networks. *CEUR Workshop Proceedings*. 2023. No. 3687. P. 121–128.

ANTONENKO ARTEM
Candidate of Technical Sciences, Associate Professor,
Department of Standardization and Certification of Agricultural Products,
National University of Life and Environmental Sciences of Ukraine, Kyiv, Ukraine
https://orcid.org/0000-0001-9397-1209
Scopus Author ID: 57207861964
Researcher ID: AAM-7380-2021
E-mail: artem.v.antonenko@gmail.com

GOLUBENKO OLEKSANDR
Candidate of Technical Sciences, Associate Professor,
Department of Information and Communication Technologies, Higher Education Institution "Academician Yuri Bugay International Science and Technical University", Kyiv, Ukraine
https://orcid.org/0000-0002-1776-5160
Scopus Author ID: 59155637100 (57552544800)
E-mail: o.golubenko@istu.edu.ua

ONYSKO ANDRII
Candidate of Military Sciences, Associate Professor,
Department of Digital Technologies in Energy,
National Technical University of Ukraine "Igor Sikorskykyiv Polytechnic Institute", Ukraine
https://orcid.org/0000-0001-7178-1417
E-mail: oniskoandrij2020@gmail.com

CHECHYK SERHII
Postgraduate Student,
Department of Computer Engineering,
State University of Information and Communication Technologies, Kyiv, Ukraine
https://orcid.org/0009-0009-9293-5156
E-mail: kardinal5700@ukr.net

VOSTRIKOV SERGII
Postgraduate Student,
Department of Computer Engineering,
State University of Information and Communication Technologies, Kyiv, Ukraine
https://orcid.org/0009-0008-8425-8872
E-mail: s.vostrikov@stud.duikt.edu.ua

Артем АНТОНЕНКО[1], Олександр ГОЛУБЕНКО[2], Андрій ОНИСЬКО[3], Сергій ЧЕЧИК[4], Сергій ВОСТРІКОВ[4]

[1] *Національний університет біоресурсів і природокористування України, Київ, Україна*
[2] *ЗВО "Міжнародний-науково технічний університет імені академіка Юрія Бугая", Київ, Україна*
[3] *Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Україна*
[4] *Державний університет інформаційно-комунікаційних технологій, Київ, Україна*

ПЕРСПЕКТИВИ ВИКОРИСТАННЯ ШІ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета. *Метою дослідження є аналіз перспектив використання штучного інтелекту для подолання ключових обмежень традиційного автоматизованого тестування програмного забезпечення, зокрема ламкості тестів, низької адаптивності, великих обсягів тестових сценаріїв та невідповідності швидкості тестування темпам сучасної розробки, а також обґрунтування переходу до інтелектуальних, самонавчальних систем забезпечення якості.*

Методика. *У роботі застосовано метод системного огляду та синтезу сучасної наукової літератури й публікацій провідних технологічних компаній, порівняльний аналіз традиційних і ШІ-орієнтованих підходів до тестування, узагальнення практичних прикладів використання великих* *мовних моделей, reinforcement learning, графових нейронних мереж та self-healing технологій, а також ілюстративне моделювання переваг ШІ через схематичні діаграми та описові приклади.*

Результати. *Дослідження виявило шість основних проблем класичного тестування та продемонструвало, що інтеграція ШІ дозволяє автоматично генерувати тестові сценарії (в тому числі з природної мови), прогнозувати дефекти, реалізовувати self-healing механізми, оптимізувати пріоритизацію тестів у CI/CD, виявляти flaky-тести та суттєво підвищувати покриття коду й стабільність процесів при одночасному скороченні витрат на підтримку автотестів.*

Наукова новизна. *Новизна полягає в системному синтезі та актуалізації тенденцій застосування великих мовних моделей для генерації тестів на основі природної мови, графових нейронних мереж для аналізу архітектурних залежностей, а також методів навчання з підкріпленням для адаптивної навігації в інтерфейсах, що разом сприяють формуванню концепції проактивного, інтелектуального та контекстно-орієнтованого тестування програмного забезпечення.*

Практична значимість. *Результати роботи мають безпосереднє прикладне значення для QA-команд і DevOps-процесів: запропоновані ШІ-підходи дозволяють суттєво скоротити час і ресурси на підтримку автоматизованих тестів, прискорити цикли CI/CD, підвищити надійність релізів, оптимізувати фокус тестування на ризикових ділянках коду та зробити процеси забезпечення якості доступнішими навіть для команд із обмеженими технічними ресурсами.*

Ключові слова: *тестування програмного забезпечення; автоматизація тестування; штучний інтелект; машинне навчання; генерація тестів; self-healing; прогнозування дефектів; забезпечення якості; нейронні мережі; CI/CD.*