Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

UDC 004.62:629.3 DOI: https://doi.org/10.30837/0135-1710.2026.189.109

Dmytro Diachenko, Oleg Mikhal

METHOD FOR ASSESSING THE RISK OF DEFECTS IN SOFTWARE COMPONENTS OF DISTRIBUTED COMPUTER SYSTEMS FOR AUTOMATED TEST PRIORITIZATION

The subject of research is methods, models, and tools for assessing the risk of defects in software components of distributed computer systems to support the process of automated test prioritization. **The aim of the work** is to develop a method for assessing the risk of defects in software components of distributed computer systems for automated test prioritization based on machine learning, considering structural characteristics of components, change intensity, and testing results. In accordance with the stated aim, **the following tasks** are to be accomplished: to analyze current approaches to software defect prediction and automated test prioritization in distributed computer systems; to identify informative characteristics of software components that reflect their structural properties, change intensity, and relationships between software components; to develop a method for assessing the risk of defects in software components based on a machine learning model with the formation of an integral risk indicator; to develop an approach to automated test prioritization based on the obtained integral risk indicator; to implement the proposed method and to conduct an experimental evaluation of its effectiveness using classification and ranking metrics. **Methods**. The study employs system analysis methods to investigate the features of functioning of distributed computer systems and the process of automated testing of software components. To construct the feature space, methods for analyzing structural metrics of source code, characteristics of change intensity, relationships between software components, and the construction of derived indicators of complexity and coupling are applied. The development of the defect risk assessment method is carried out using machine learning methods, classification, feature selection, formation of an integral risk indicator, and risk-oriented ranking. For experimental validation of the proposed approach, computational modeling, comparative analysis, and model quality evaluation methods are used, including ROC-AUC, Precision–Recall, F1-score, and Balanced Accuracy metrics. The software implementation is carried out in Python using the Google Colab environment. **Results**. A method for assessing the risk of defects in software components of distributed computer systems has been developed, which ensures the integration of structural, process, and contextual characteristics, as well as the computation of an integral risk indicator and ranking of components for automated test prioritization. Experimental validation has confirmed the effectiveness of the proposed method and its ability to accurately distinguish between defective and non-defective software components, as well as to identify high-risk components for priority testing. **Conclusions**. The results obtained confirm the feasibility of using machine learning methods and an integrated feature space for assessing the risk of defects in software components of distributed computer systems. The proposed method can be used as a basis for improving the justification of decisions on automated test prioritization, timely identification of high-risk components, and further development of intelligent software quality assurance tools.

Keywords: software components, distributed computer systems, automated testing, machine learning, defect prediction, integral risk indicator, prioritization, testing, Python, Google Colab.

1. Introduction

Modern distributed computer systems form the foundation for a significant number of software platforms that operate through the continuous interaction of a large number of interconnected components. Such systems include cloud services, microservice architectures, enterprise information platforms, data processing systems, and mission-critical hardware-software complexes. They are characterized by a high level of structural complexity, intensive data exchange between components, frequent changes to the software code, and a significant dependence of the system's overall performance on the reliability of individual software modules. Under these conditions, even local defects in individual components can lead to the disruption of significant parts of the system, reduction in productivity, loss of data consistency, or service failures. This results in increased demands on the organization of automated software testing, which must ensure timely defect detection and the rational allocation of testing resources.

Traditional approaches to automated software testing are largely based on fixed test execution rules, standardized verification procedures, or the uniform distribution of test resources among software components. This approach to testing does not account

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189)
ISSN 3083-7650 (0135-1710) (print) Management Information System and Devices. 2026. No. 2 (189)

for the actual defect risk of individual modules, their structural characteristics, interdependencies with other components, the rate of change, or the results of previous tests. As a result, test resources are often spent on verifying components with a relatively low probability of defects, while high-risk software modules do not receive the appropriate priority. This is particularly critical for distributed computer systems, where the complexity of interactions between components increases the likelihood of defects, and delays in their detection can have significant functional and operational consequences.

One promising approach to improving the effectiveness of automated testing is the application of machine learning methods to assess the risk of defects in software components. The use of such methods allows for the analysis of a set of structural characteristics of the software code, connectivity metrics, change intensity, and test results to form an integrated assessment of defect risk. The resulting metric can be used as a basis for ranking components by potential defect severity and subsequently prioritizing automated testing. This enables a shift from universal testing schemes to a risk-oriented approach, in which testing resources are directed primarily toward the most critical parts of the software system. Existing approaches to software defect prediction are predominantly focused either on the use of individual software code metrics or on the analysis of historical defect data without sufficient integration of the diverse characteristics of software components into a unified assessment framework. At the same time, the task of planning automated testing in distributed computer systems requires an approach that combines the analysis of components' structural properties, the characteristics of their interrelationships, the intensity of changes, and test execution results. This necessitates the development of a method for assessing the defect risk of software components in distributed computer systems, which would ensure the formation of an integrated risk indicator and its use for prioritizing automated testing. Solving this problem is a relevant scientific and applied task aimed at improving testing efficiency, earlier defect detection, and enhancing the overall quality of software for distributed computer systems.

2. Analysis of current scientific publications and identification of the research problem

Current research explores various approaches to defect prediction, ranging from classical statistical models to ensemble algorithms and deep learning methods. Currently, there is a need to identify scientific approaches that can serve as a basis for developing a method for assessing the risk of defects in software components of distributed computer systems, focused not only on predicting the occurrence of defects but also on supporting the prioritization of automated testing. That is why the subsequent analysis of scientific publications is aimed at identifying the achievements and limitations of existing solutions, as well as at justifying the research problem, which lies in the need to construct a method capable of generating an integrated defect risk metric based on a machine learning model and using it for more effective planning of automated testing.

Work [1] examines an approach to software defect prediction within the framework of the Just-In-Time Software Defect Prediction concept [2], which is based on the use of deep learning models. The authors justify the feasibility of early defect detection at the level of individual changes in the source code and demonstrate the possibility of improving software quality through the use of historical data and change metrics. The proposed multi-layer perceptron model achieves a prediction accuracy of 82.08% and identifies the most influential metrics, including the volume of changes, developer experience, and the degree of change distribution across the codebase. However, despite the effectiveness of the approach, the study focuses primarily on classifying changes as either defective or non-defective and does not consider the formation of an integrated defect risk metric for software components as a basis for prioritizing automated testing. Furthermore, the features used primarily reflect the characteristics of code changes without comprehensively accounting for the structural properties of software components and a generalized representation of risk in the context of distributed computer systems. This limits the applicability of the obtained results to test planning tasks and the effective allocation of verification resources.

The paper [3] considers an approach to software defect prediction in which the task of predicting defects in software code is proposed to be interpreted not only as a classification task but as a task of ranking software modules by defect severity. The authors argue that for practical test planning, it is not enough to simply determine whether a module contains a defect, since it is also important to establish the order of module verification given limited testing resources. To this end, the study investigates an approach that allows modules

Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

to be ranked by the number of errors or defect density and the quality of the ranking to be assessed by the average error rate. A comparative analysis of ranking models was also conducted, and the impact of data imbalance and feature selection on prediction quality was analyzed. The results showed that using the number of errors as the target variable yields higher and more stable ranking results than using error density, and that training on unbalanced data and feature selection do not provide universal improvements across all scenarios. Special attention was paid to the use of both static metrics of the program code and metrics characterizing the history of changes and the evolution of software modules. At the same time, the study focuses primarily on improving the accuracy of classifying modules with defects, while the issues of developing a generalized defect risk metric and its use to support the prioritization of automated testing remain insufficiently explored. Furthermore, the study does not pay specific attention to the characteristics of distributed computer systems, for which not only individual characteristics of software modules are important, but also the relationships between software components and their impact on the overall risk of defects. This confirms the relevance of developing an approach focused on assessing the defect risk of software components in distributed computer systems, with the subsequent use of the obtained assessments for more informed prioritization of automated testing.

Article [4] investigates the problem of software defect prediction based on machine learning and deep learning methods. The authors consider the prediction of software component defects as a tool for early detection of errors in software classes with the aim of reducing software maintenance costs and improving its reliability. The study utilizes the large Unified Bug Dataset, which combines data from five open sources and contains information on 47,618 classes and approximately 60 software code metrics reflecting complexity, connectivity, cohesion, size, and other characteristics of the classes' internal quality. The study also compared eight machine learning approaches. Before building the models, the target variable was binarized and the features were standardized, and the results were evaluated using several metrics. The LSTM model showed the best results, confirming the feasibility of using machine learning. However, the study focuses primarily on improving the accuracy of defect class detection based on a set of source code metrics, while the use of prediction results for controlling the automated testing process remains outside the scope of consideration. Furthermore, the proposed approach is primarily focused on the level of individual classes and does not emphasize the specifics of distributed computer systems, where the relationships between software components and the integrated assessment of defect risk are of significant importance.

Article [5] addresses the problem of software defect prediction based on optimized machine learning models, with a focus on the impact of hyperparameter tuning and feature dimensionality reduction. The authors emphasize that the effectiveness of the models depends significantly not only on the type of algorithm but also on the correct selection of parameters and relevant features used for training. The research methodology includes feature dimensionality reduction using the principal component analysis method, optimization of the number of components and model hyperparameters using random search, as well as a comparison of several machine learning algorithms. Model performance was evaluated using accuracy, precision, recall, and F1-score metrics, which allowed for a comprehensive assessment of the models' ability to detect defective modules. The experimental results showed that the k-nearest neighbors method is the most effective algorithm in most cases, and that hyperparameter optimization and data preprocessing significantly improved prediction accuracy compared to baseline models without tuning. It is worth noting that the study focuses primarily on improving the accuracy of classifying defective modules and does not consider the subsequent use of prediction results in test management or resource allocation processes. Furthermore, the approach is model-specific and does not provide for the integration of results from different algorithms into a single, generalized defect risk metric. This highlights the relevance of developing an approach that would combine defect prediction results with the formation of an integrated risk assessment of software components and ensure their use for intelligent prioritization of automated testing in complex software systems.

Article [6] addresses the problem of software defect prediction in the context of large-scale complex information systems, for which traditional testing approaches are resource-intensive and insufficiently effective.

The authors emphasize that the increasing complexity of systems necessitates a transition to intelligent analysis methods, in particular the use of machine learning for defect prediction based on historical development and testing data. The paper proposes a generalized approach to static defect prediction, which

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189)
ISSN 3083-7650 (0135-1710) (print) Management Information System and Devices. 2026. No. 2 (189)

involves forming a set of features for software modules, building a classification model, and subsequently applying it to determine the risk of defects in new components. Particular attention is paid to the formation of informative metrics that include not only software code characteristics but also development process indicators, specifically the frequency of changes, the complexity of modifications, and organizational aspects of the team's work. The prediction model is implemented as a binary classification task using machine learning methods, specifically logistic regression, which allows for the assessment of the probability of defects in a software module. At the same time, the approach proposed by the authors is primarily focused on classifying modules based on defect risk and does not provide for the formation of a generalized integrated risk indicator that would account for the cumulative impact of diverse characteristics of software components. Furthermore, the prediction results are not directly integrated into test management processes. This justifies the development of an approach that would combine machine learning with the formation of an integrated assessment of software component defect risk and ensure its use to improve testing efficiency in distributed computer systems.

Work [7] is devoted to solving the problem of defect prediction in a dynamic software development environment, specifically in the context of [2], where data arrives sequentially over time. The authors highlight the limitations of traditional approaches, which primarily operate in a static mode and do not account for dynamic changes in data characteristics caused by the evolution of software systems. The paper presents a comprehensive comparison of online and offline approaches to training defect prediction models. The BORB method is proposed, which allows offline models to be adapted to an online scenario through periodic retraining on updated data, taking into account the delay in defect verification. The results of the experimental study showed that offline models in an online scenario can demonstrate slightly better prediction quality compared to classical online methods, although this is achieved at the cost of greater computational complexity. At the same time, the approach is primarily focused on classifying changes as either defective or non-defective and does not provide for the formation of an integrated defect risk metric for software components. Furthermore, the prediction results are not directly integrated into the processes of optimizing and prioritizing automated testing. This justifies the need to develop a method for assessing defect risk in distributed computer systems that would combine machine learning with the formation of a generalized risk-oriented indicator and ensure its use to improve testing efficiency.

Article [8] addresses the problem of software defect prediction using machine learning methods aimed at improving the quality and reliability of software systems and reducing development costs. The authors emphasize that early detection of defective modules allows for significant optimization of the testing process and reduction of software maintenance costs, which is particularly relevant for modern complex software systems. The paper proposes an approach to generating software project metrics used as input parameters for prediction models. These metrics form a generalized representation of the software project's state and allow for the consideration of the impact of different life cycle phases on defect occurrence. The results of the experimental study demonstrate that the model based on a decision tree regression provides higher prediction accuracy compared to the k-nearest neighbors method. However, the study focuses primarily on predicting the number of defects and does not account for the formation of an integrated risk metric that could be used to support decision-making in the testing or quality management process. Furthermore, the model does not account for dynamic changes in data over time, which can be critical in distributed software systems.

Article [9] addresses the problem of software defect prediction using machine learning methods, with an emphasis on improving accuracy through ensemble approaches. The author emphasizes that the reliability of software systems directly depends on the timely detection of defects, and the application of intelligent methods allows for optimizing the testing process and reducing resource consumption. The paper provides a detailed analysis of approaches to defect prediction, specifically regression and classification models. Particular attention is paid to ensemble learning methods, which allow for improved prediction accuracy by combining several base models. An experimental study was conducted using the open-source NASA PROMISE and GitHub datasets, which contain information about software modules and their defects. In most cases, the ensemble model demonstrated better results compared to individual algorithms and also provided more stable productivity for both small and large datasets. It should be noted that this study focuses primarily on classical machine learning

Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

algorithms and does not consider the capabilities of deep learning or adaptive models, which opens up prospects for further research in this direction.

Article [10] discusses modern approaches to software defect prediction using machine learning methods. The authors emphasize that early defect detection makes it possible to improve software quality and reduce maintenance costs. The paper analyzes the main groups of machine learning methods used in software component defect prediction tasks. It summarizes current research using various datasets and highlights the importance of data preprocessing and the proper selection of metrics for evaluating model quality. The authors note that accuracy, precision, recall, F1-score, and ROC curves are most commonly used to evaluate model performance. Overall, the review confirms the high effectiveness of machine learning methods in defect prediction tasks and demonstrates a trend toward the expanding use of ensemble and deep learning models. It should be noted that this work is primarily review-based and does not propose a specific approach to developing an integrated defect risk metric for software components. Furthermore, the study does not consider the use of prediction results to support the prioritization of automated testing.

Article [11] examines the problem of software defect prediction using traditional and ensemble machine learning methods. The authors emphasize the importance of early defect detection for improving the quality of software systems and reducing maintenance costs. The paper investigates the performance of many machine learning algorithms and applies ensemble approaches. Experimental results obtained on NASA datasets demonstrate the high performance of ensemble models, particularly Gradient Boosting, confirming the feasibility of their use in defect prediction tasks. At the same time, it should be noted that the research is primarily focused on improving classification accuracy and does not involve the development of an integrated defect risk metric for software components, nor its use for prioritizing automated testing in distributed computer systems.

In current research, significant attention is paid to improving the effectiveness of software defect prediction based on machine learning methods, driven by the increasing complexity of software systems and the need to optimize testing processes. One of the key areas is ensuring the quality of training data and its representativeness, as these factors significantly influence the accuracy and generalization ability of models. Article [12] addresses the limitation of existing defect prediction models caused by insufficient volume and heterogeneity of training samples. A large and diverse dataset, Defectors, is proposed, which covers a significant number of software projects across various domains and enables improved effectiveness in applying machine learning and deep learning models. Particular attention is paid to class balancing and reducing noise in the data, which positively impacts prediction quality. At the same time, the work focuses primarily on building a high-quality training dataset and does not address the integration of the obtained results into the decision-making process regarding testing prioritization. An analysis of current scientific publications has shown that existing approaches to software defect prediction based on machine learning provide a solid theoretical and applied foundation for further research in this field. In most of the works reviewed, the main focus is on improving the accuracy of classifying defective modules, optimizing the feature space, using ensemble algorithms, deep learning, and adapting models to the characteristics of specific datasets. At the same time, such studies are predominantly oriented either toward identifying components with defects or toward ranking modules by the severity of the defects themselves, without fully integrating the prediction results into the decision-making process regarding the organization of automated testing. At the same time, the issues of developing a method capable of combining the structural characteristics of software components, the intensity of changes, the relationships between software components, and test execution parameters into a single integrated defect risk metric remain insufficiently addressed. Furthermore, there is limited coverage of the specifics of applying such approaches specifically to distributed computer systems, for which complex component interactions, data heterogeneity, and the need for well-founded prioritization of automated testing are of significant importance. In this regard, it is advisable to develop a method for assessing the defect risk of software components in distributed computer systems based on machine learning, which will ensure the formation of an integrated risk indicator and its use to improve the effectiveness of automated test prioritization.

3. Research objectives and tasks

The aim of this work is to develop a method for assessing the risk of defects in software components of distributed computer systems in order to prioritize

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189)
ISSN 3083-7650 (0135-1710) (print) Management Information System and Devices. 2026. No. 2 (189)

automated testing based on machine learning, taking into account the structural characteristics of the components, the rate of change, and test results. Given this objective, the following tasks must be performed: analyze current approaches to software defect prediction and the prioritization of automated testing in distributed computer systems; identify informative characteristics of software components that reflect their structural properties, change intensity, and interdependencies between software components; develop a method for assessing the risk of software component defects based on a machine learning model with the formation of an integrated risk indicator; develop an approach to prioritizing automated testing based on the obtained integrated risk indicator; implement the proposed method and conduct an experimental evaluation of its effectiveness in terms of classification and ranking metrics.

4. Presentation of the main material

Improving the effectiveness of automated testing in distributed computer systems requires not only identifying defects in individual software components but also quantitatively assessing their risk level. In real-world conditions, testing resources are limited; therefore, the task of ranking components by the expected probability of a defect occurring and identifying the most critical objects for early test control becomes of paramount importance. It is for this purpose that this article proposes a method for assessing the risk of defects in software components of distributed computer systems, which combines machine learning models, feature engineering, and the integration of structural and process characteristics, and justifies the introduction of an integrated risk indicator.

The basic idea behind the method is that each software component is described by a set of quantitative characteristics that reflect both its internal properties and the context of changes and the development history of the corresponding project. Let the set of software components of a distributed system be defined as follows:

$$C = \{c_1, c_2, \ldots, c_n\}. \quad (1)$$

Each component c_i is associated with a multidimensional feature vector

$$\mathbf{x}_i = \left(\mathbf{x}_i^{(hist)}, \mathbf{x}_i^{(test)}, \mathbf{x}_i^{(str)}, \mathbf{x}_i^{(dep)}, \mathbf{x}_i^{(ops)}\right), \quad (2)$$

where $\mathbf{x}_i^{(hist)}$ is a vector of historical characteristics of component changes, $\mathbf{x}_i^{(test)}$ is a vector of metrics related to test results or test control parameters, $\mathbf{x}_i^{(str)}$ is a vector of structural characteristics of the program code, $\mathbf{x}_i^{(dep)}$ is a vector of characteristics of dependencies and relationships between software components, and $\mathbf{x}_i^{(ops)}$ is a vector of operational or contextual parameters. This division of the feature space allows for a formal account of the multifactorial nature of software component defects. Unlike classical approaches, which are often limited to code metrics alone, this representation allows for the simultaneous integration of information about structure, development history, the context of changes, and component interactions.

We define the target variable as

$$y_i = \begin{cases} 1, & \text{if the component has a defect,} \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

At the first level, the method generates an estimate of the probability of defects in a software component

$$p_i = f(\mathbf{x}_i), \quad (4)$$

where $f(\ )$ – a machine learning model. The value p_i is interpreted as an estimate of the probability that component c_i is defective or has an increased susceptibility to defects. Thus, function f is the core analytical component of the method, and its result is used not as a final decision but as one of the components of the integrated risk assessment. The integral risk indicator for the i-th component is given by the relation

$$R_i = \alpha p_i + (1-\alpha)\sum_{j=1}^{5} \lambda_j \hat{x}_{ij}, \quad \sum_{j=1}^{5} \lambda_j = 1. \quad (5)$$

In this expression, R_i is the overall risk assessment for a software component, α is the coefficient determining the contribution of the machine-based prediction p_i to the overall risk assessment, λ_j are the weighting coefficients for individual groups of features, and $\hat{x}_{ij}$ is the normalized generalized value of the j-th group of characteristics for the component c_i. The condition $\sum_{j=1}^{5} \lambda_j = 1$ ensures the normalization of the weight scheme and allows λ_j to be interpreted as the relative contribution of the corresponding group of factors to risk formation. The presence of two components in the risk formula is of fundamental importance. The first component αp_i reflects the contribution of the machine learning model, which accounts for nonlinear dependencies between features

Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

and the target variable. The second component, $(1-\alpha)\sum_{j=1}^{5}\lambda_j \hat{x}_{ij}$, ensures a structured consideration of group risk factors and enhances the interpretability of the method. As a result, the risk assessment combines both the predictive potential of the model and the analytical transparency of the weighted representation of factors.

To compare the components, we will use the values $R \in [0,1]$. However, this approach effectively reduces the method to a standard binary classification followed by ranking; therefore, a more complex risk index structure was introduced, which takes into account not only the probabilistic forecast but also generalized group characteristics. To this end, after filling in the gaps and standardizing the data, all features were divided into three groups: structural, process-related, and contextual. A normalized group index was formed for each group. If S_i is the structural index, P_i is the process index, and G_i is the contextual index, then the integrated risk indicator in its improved form was defined as

$$R_i^{v1} = \alpha p_i + (1-\alpha)\left(w_1 S_i + w_2 P_i + w_3 G_i\right), \quad (6)$$

where $\alpha \in [0,1]$ specifies the contribution of the predicted probability of defects, and the coefficients w_1, w_2, w_3 determine the relative importance of the structure, process, and context indices, subject to the condition:

$$w_1 + w_2 + w_3 = 1. \quad (7)$$

A situation is possible in which the risk model relies not only on individual raw metrics but also on interaction features. In this case, the component's risk is formed by combining local risk, which reflects the internal properties of the component itself, and context risk, which characterizes the conditions of its development and the operational context. As a result, the final risk index takes the form:

$$R_i^{v2} = \alpha p_i^{stack} + (1-\alpha)\left(\gamma R_i^{(local)} + (1-\gamma) R_i^{(context)}\right), \quad (8)$$

where p_i^{stack} is the defect probability obtained from the model, $R_i^{(local)}$ is the local structure risk, $R_i^{(context)}$ is the context risk, and $\gamma \in [0,1]$ defines the ratio between the local and context components. This scheme best addresses the task of risk-oriented prioritization of automated testing, as it allows not only classifying components as defective or non-defective but also forming an ordered risk scale for planning tests.

Test prioritization is performed by sorting components in descending order of risk:

$$\pi = \text{argsort}\left(R_i, \downarrow\right), \quad (9)$$

where π is the ordered sequence of components.

For the Top-k approach [13], the following is used:

$$\text{Class}(c_i) = \begin{cases} low, & R_i < \tau_1, \\ medium, & \tau_1 \le R_i < \tau_2, \\ high, & R_i \ge \tau_2. \end{cases} \quad (10)$$

where τ and τ are threshold values that divide a continuous risk scale into three interpretable levels: low, medium, and high. This representation is important from an applied perspective, as it allows not only for ranking components but also for grouping them by criticality. Components in the high class are of the greatest interest for early testing prioritization; components in the medium class can be considered second-priority items for verification; and components in the low class have the lowest expected risk of defects.

To determine the optimal threshold, the following quality function is used:

$$F1(\tau) = \frac{2 \cdot \text{Precision}(\tau) \cdot \text{Recall}(\tau)}{\text{Precision}(\tau) + \text{Recall}(\tau)}, \quad (11)$$

The optimal threshold is determined by:

$$\tau^* = \max_{\tau} F1(\tau). \quad (12)$$

It is proposed to evaluate the effectiveness of the method using the following metrics:

a) area under the ROC curve (ROC-AUC):

$$AUC = \int_0^1 TPR(FPR)\,dFPR; \quad (13)$$

b) Precision-Recall curve:

$$AP = \int_0^1 Precision(Recall)\,dRecall; \quad (14)$$

c) F1-score:

$$F1 = \frac{2TP}{2TP + FP + FN}; \quad (15)$$

d) Balanced Accuracy:

$$BA = \frac{TPR + TNR}{2}. \quad (16)$$

We also introduce feature importance as a term in the function:

$$\text{Importance}_j = \frac{1}{K}\sum_{k=1}^{K} I_{jk}, \quad (17)$$

where I_{jk} is the importance of the j-th feature in the k-th model.

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189)
ISSN 3083-7650 (0135-1710) (print) Management Information System and Devices. 2026. No. 2 (189)

5. Justification for the selection, preparation, and integration of datasets

To implement the method, it was decided to combine classic open-source software defect datasets with more modern process and project characteristics. The foundation consisted of regenerated PROMISE and BPD datasets in Excel format, as well as an additional dataset, csv_result-JDT.xls. This data provided a reliable foundation at the software component level, as it contained structural code metrics and a target feature indicating the presence of defects. Specifically, the data included metrics such as cbo, dcc, exportcoupling, importcoupling, nom, wmc, and the defect label. However, structural metrics alone proved insufficient for the method to work effectively. Therefore, to expand the feature space, additional open tables from the SQuaD dataset [14] were incorporated, namely process_metrics.csv, github_metrics.csv, and release_data.csv. The selected tables allowed us to add such important parameters to the structural level as change intensity, number of commits, number of fixes, author activity, number of files with defects, overall repository activity, and release characteristics. The choice of this specific combination of data sources was driven by the desire to build not just a defect classifier, but to develop a method that integrates several types of signals. PROMISE/BPD and JDT provided a detailed description of components, while the SQuaD tables provided a measure of processes and contexts.

Data preparation was a separate, essential stage of implementation. First, all sheets of the Excel file from PROMISE/BPD and the JDT file were read. Column names were normalized to a single format to avoid issues with case, spaces, delimiters, and inconsistent variable names. Next, the project name and release were automatically extracted from the sheet names, which allowed the creation of the project and release fields. At this stage, it was discovered that certain sheets and rows did not contain actual data about components but served a utility or reference function. That is why rows without a class name, sheets of the Main type, as well as other parts unsuitable for modeling were removed from the base dataset. This cleaning allowed us to transition from a mixed source to a component-oriented dataset. After the initial merge, an intermediate dataset named base_df was formed, in which each row corresponded to a separate software component. Its key fields were the project name, component name, structural metrics, and defect label.

Next, integration with the SQuaD tables was performed. This stage proved to be the most challenging. It became clear that a direct merge based on the project + release fields did not work, since the project and release identifiers were not consistent across different sources. For example, in the base dataset, the project was labeled as "ant", whereas in the SQuaD tables it appeared as "apache#ant". Similar inconsistencies were observed for "camel" and other projects. To address this issue, a project name normalization procedure was implemented, which mapped designations such as "apache#ant" to the unified value "ant". Another significant problem turned out to be poor-quality or incomplete information about releases in the base set. That is why, instead of merging at the project + release level, a decision was made to perform integration at the project level. As a result, process and project characteristics were attached to all components of the corresponding project as contextual information. Formally, this meant that the same set of process attributes was duplicated for components of a single project. For the purpose of risk assessment, this is acceptable, since the context risk can indeed be shared by components developed under the same conditions.

After integration, the final intermediate dataset modeling_dataset_merged_v2.csv was obtained. A check of feature completeness showed that the basic structural characteristics were fully available; that is, for indicators such as cbo, dcc, exportcoupling, importcoupling, nom, and wmc, the proportion of non-empty values was 1.0. Additional characteristics of processes and projects had a completeness of approximately 0.3649, indicating a partial but meaningful enrichment of the data. Based on this dataset, two final datasets were formed. The first, final_structural_dataset.csv, contained only structural features and was used to build the baseline method. The second, final_extended_dataset.csv, contained characteristics of structures, processes, and projects and was used for the extended and improved versions of the method.

6. Experimental implementation

In implementing the proposed method for assessing the risk of defects in software components, the Python programming language was used as one of the most widely used platforms for data analytics and machine learning tasks. The choice of Python is due to the

Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

availability of a well-developed ecosystem of libraries for data processing, model building, and result visualization, including NumPy, pandas, scikit-learn, and matplotlib, which allow for the effective implementation of all stages of data processing: from preprocessing to model quality assessment. The software prototype of the method was implemented in the Google Colab environment, which provides access to computational resources without the need for local infrastructure setup. Using this environment allows working with large datasets, performing complex computational operations for training machine learning models, and provides a convenient mechanism for interactive analysis of results.

Experimental validation was conducted using a fixed split of the data into training and test sets in a 75/25 ratio, with stratification by the target feature. This approach ensured that the ratio of defective to non-defective components was correctly preserved in each subsample. For all models, missing values were imputed with median values, and for scale-sensitive algorithms, feature standardization was also applied. Quality assessment was performed using ROC-AUC, Average Precision, F1-score, Balanced Accuracy, Precision-Recall, as well as top-k analysis, which is particularly important for the task of test prioritization. A separate threshold selection was performed to maximize the F1-score or Balanced Accuracy. This allowed us not only to compare methods based on probabilities but also to find the optimal operating point for the practical application of the method.

In the first stage of the experiments, a basic version of the method was implemented, which used only structural metrics of the components. The main goal of this stage was to establish a baseline for comparison. The following served as input features: cbo, dcc, exportcoupling, importcoupling, nom, and wmc. Concurrently, an extended version was implemented, in which process and project characteristics were added to the structural features. In practice, this made it possible to verify whether the new information signal actually improves the quality of risk assessment. In the early stages, CatBoost [15] was used for this task, as it is well-suited for tabular data. Figure 1 shows the ROC curves for two approaches to assessing software component defects: the baseline method, which uses only structural metrics, and the extended method, which integrates additional characteristics. The x-axis plots the False Positive Rate (FPR), and the y-axis plots the True Positive Rate (TPR).

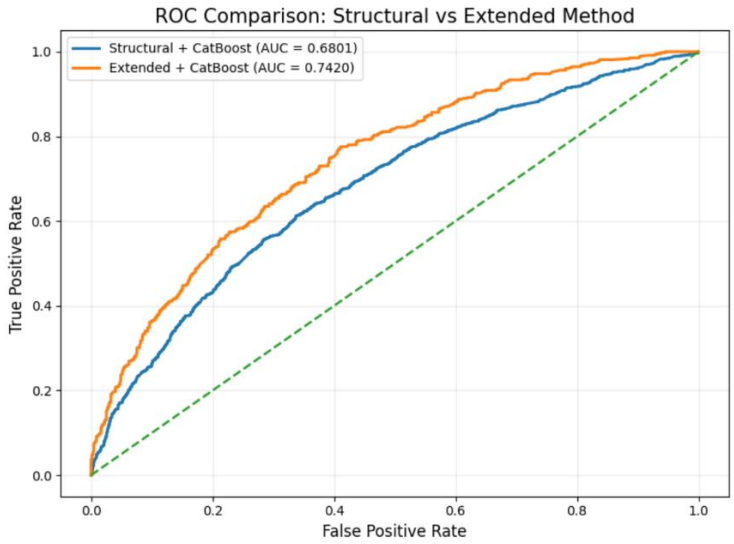

Fig. 1. ROC curves for the structural and extended methods

The extended method consistently outperforms the baseline approach across the entire range of values, as evidenced by the larger area under the ROC curve (AUC = 0.7420 vs. 0.6801). This indicates the model's superior ability to correctly distinguish between components with and without defects, confirming the validity of integrating structural and process characteristics. Thus, incorporating additional features improves classification quality and ensures more effective defect detection in software components. Fig. 2 presents a comparison of the Precision–Recall curves for the baseline method (structural metrics only)

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189)
ISSN 3083-7650 (0135-1710) (print) Management Information System and Devices. 2026. No. 2 (189)

and the extended method. Recall is plotted on the x-axis, and precision on the y-axis. The extended method demonstrates higher precision values across most of the recall range, as evidenced by the larger area under the curve (AUC = 0.6718 vs. 0.5062). This means that when detecting component defects, it produces fewer false positive rate (FPR) triggers.

The results obtained indicate a significant improvement in the method's performance under conditions of unbalanced data, which is critically important for test prioritization tasks.

After developing the extended method, the first refinement was implemented. In this version, risk was no longer equated solely with the probability of defects and began to be calculated as a combination of a probabilistic forecast with group indices. To this end, all features were divided into structural, process, and contextual groups, and after standardization, corresponding group indices were formed. Figure 3 shows the ROC curves for the baseline extended method and the improved risk assessment method. It can be seen that both methods demonstrate similar results; however, the improved method has a slightly higher area under the curve.

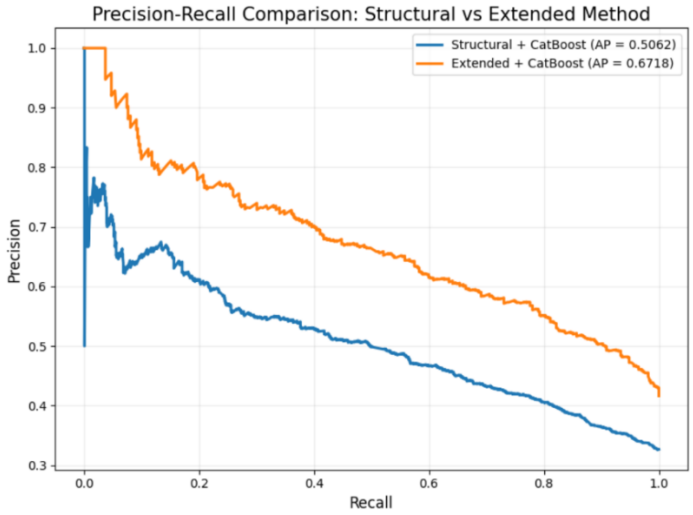

Fig. 2. Precision-Recall for the structural and advanced methods

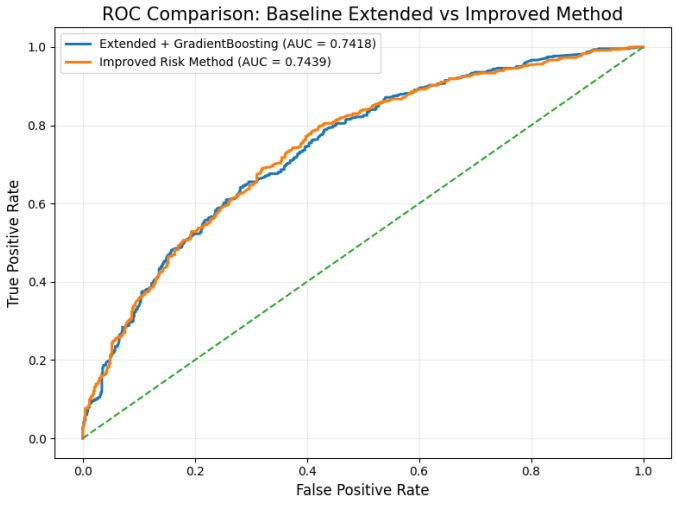

Fig. 3. ROC curves for the extended and improved methods

Figure 4 shows the dependence of the F1-score and Balanced Accuracy metrics on the classification threshold for the improved method. The x-axis plots the threshold values, while the y-axis plots the corresponding quality metric valuesBoth metrics reach their maximum values at a threshold of approximately 0.33, as indicated by the corresponding vertical lines. As the threshold is further increased, a significant

Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

decrease in the F1-score is observed, which is associated with an increase in the number of missed defects. Thus, the selected threshold value provides an optimal balance between classification accuracy and completeness and is suitable for use in the task of test prioritization.

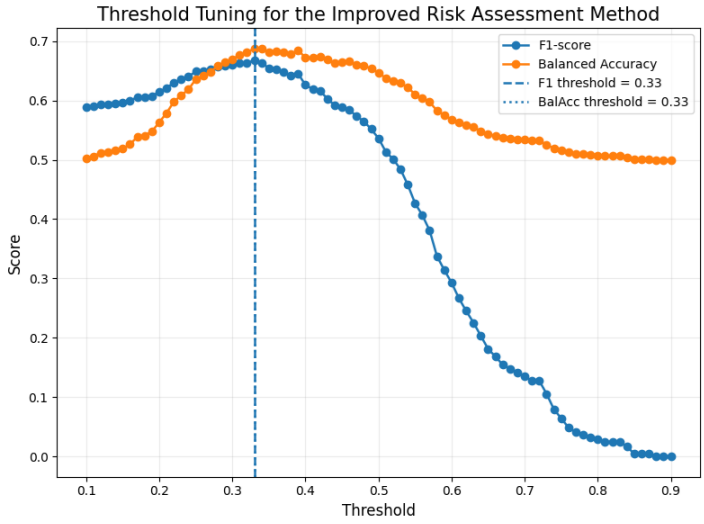

Fig. 4. The dependence of the F1-score and Balanced Accuracy metrics on the classification threshold

In practice, this option proved correct but yielded only a very slight improvement compared to the extended version. The difference in ROC-AUC was only a few thousandths, and Average Precision increased only marginally. Thus, aggregating the existing signal without introducing new informative features does little to improve the method. In other words, the first modification was useful more from the perspective of risk interpretation than from the perspective of improving predictive quality. The next step was to develop a second, significantly improved version of the method. It was here that feature interactions, a stack ensemble, and a two-level risk logic were introduced. It was found that the main limitation of the previous version was that additional process characteristics merely duplicated information at the project level but did not interact with the structure of the component itself. To address this shortcoming, new integrated features were developed. In particular, the following characteristics were introduced: complexity_plus_coupling, methods_times_coupling, coupling_gap, coupling_sum, export_import_ratio, complexity_per_method, wmc_x_log_churn, cbo_x_log_buggy_files, max_change_set_x_cbo, age_x_complexity_per_method. These features allowed us to explicitly account for the interaction between complexity, coupling, change history, and defect context. They became the primary source of the new information signal. The first layer consisted of Logistic Regression, Random Forest, and Gradient Boosting [16], while the second layer featured a meta-model based on logistic regression, which was trained to combine the probabilities of the base models. Thus, the value p_i^{stack} was obtained, which served as the basis for the final assessment. To calculate the overall risk, local and contextual risks were introduced. Local risk was formed based on structural and structural-process interactions of features, while contextual risk reflected the background of processes and projects. The final risk index was calculated as a combination of a stack-ensemble forecast and these two components.

The extended method is characterized by the dominance of classical structural metrics, specifically cbo, wmc, and nom, which determine the complexity and connectivity of software components. Dependency metrics (importcoupling, exportcoupling, dcc), which reflect the interaction between modules, also play an important role. Process and historical characteristics, such as max_change_set, latest_release_month, num_bugs, and n_auth, have a secondary influence, indicating the method's predominant focus on static code properties.

ISSN 3083-7715 (online)
ISSN 3083-7650 (0135-1710) (print)
Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189)
Management Information System and Devices. 2026. No. 2 (189)

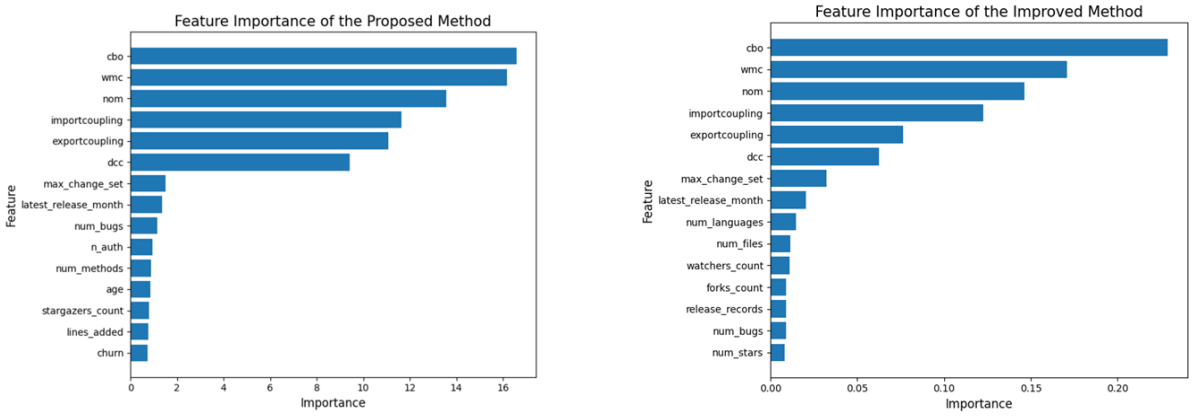

Fig. 5. The Importance of Features for the Extended and Improved (v1) Methods

For the first improved method, structural features remain dominant, but a more balanced distribution of importance is observed across different types of characteristics. Although cbo, wmc, and nom remain key, the contribution of process and repository metrics such as num_languages, num_files, watchers_count, and forks_count increases. This indicates an expansion of the model's information base and better consideration of the development context, which allows for improved accuracy in defect risk assessment compared to the baseline method. Figure 6 shows a comparison of ROC curves for three methods: the baseline extended method, the ensemble approach (v1), and the improved method (v2). The x-axis plots the proportion of FPR hits, and the y-axis plots the proportion of TPR hits. The baseline extended method yields an AUC of 0.7347, which corresponds to acceptable classification quality but is inferior to more complex models. Using a stack ensemble improves the quality to AUC = 0.7542, demonstrating the effectiveness of the ensemble combination of models and a better ability to account for nonlinear dependencies in the data. The improved method v2 demonstrates a similar AUC value of 0.7542, effectively matching the level of the ensemble approach.

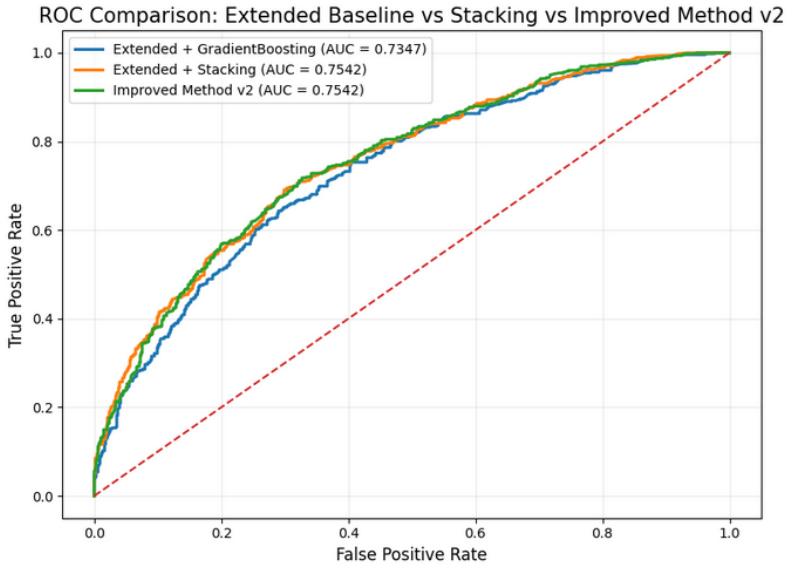

Fig. 6. Comparison of ROC Curves

Compared to previous results (where the AUC for the baseline extended approach was approximately 0.742 and for the first modification of the method was approximately 0.744), it is evident that further refinement has yielded a more significant improvement in performance. In particular, the improvement relative to the initial structural approach (AUC ≈ 0.68) is substantial, whereas compared to the previous version of the method, the improvement is incremental. Figure 7 shows the Precision–Recall curves for the baseline

Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

extended approach, the stack ensemble, and the improved method v2. The extended approach has the lowest performance (AP = 0.6646), while the other demonstrates the best result (AP = 0.6986). The improved method v2 (AP = 0.6940) nearly reaches the ensemble's level and significantly exceeds the baseline.

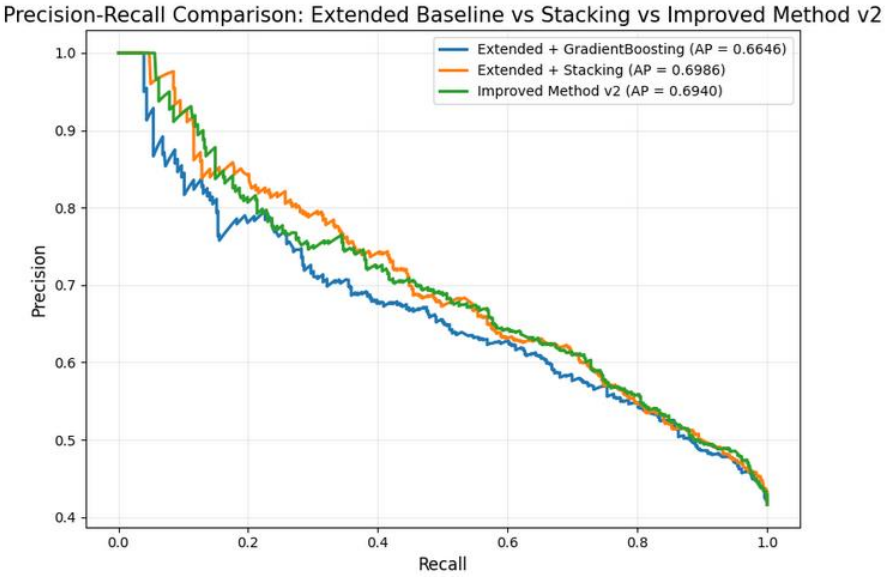

Fig. 7. Comparison of Precision–Recall Curves

Compared to previous versions of the method, there is a steady improvement in performance, particularly in terms of average recall values, which is crucial for defect detection tasks involving imbalanced data.

Figure 8 shows the dependence of the F1-score and Balanced Accuracy on the threshold value for the improved v2 method. The maximum F1-score is achieved at a threshold of approximately 0.33, while Balanced Accuracy is achieved at a higher value, approximately 0.41.

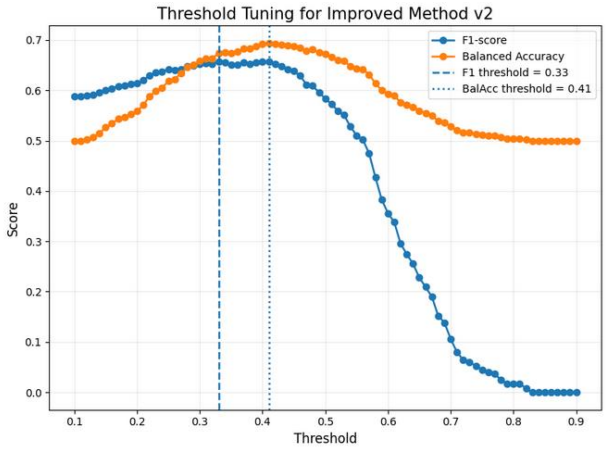

Fig. 8. Error matrix for the improved method

The results indicate a trade-off between classification completeness and accuracy: a lower threshold provides better defect detection (higher F1 score), while a higher threshold improves the balance between classes. Compared to the previous version of the method, the optimal thresholds remain similar, but the Balanced Accuracy curve is more stable, indicating improved model consistency. Figure 9 shows the confusion matrix for the improved v2 method.

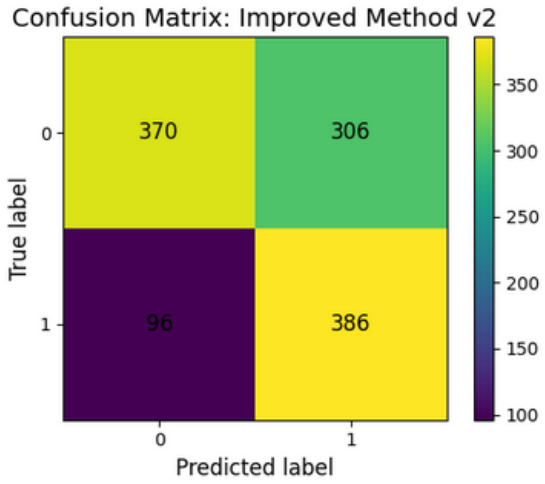

Fig. 9. Error matrix for the improved method

ISSN 3083-7715 (online) *Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189)*
ISSN 3083-7650 (0135-1710) (print) *Management Information System and Devices. 2026. No. 2 (189)*

The method correctly classifies 386 defective components and 370 non-defective ones, while having 96 missed defects and 306 false positive rate (FPR) events.

There is a slight decrease in the number of missed defects, which is a positive result for risk detection tasks. At the same time, the number of FPR decisions has increased slightly, indicating a shift in the model toward a more "conservative" strategy – with an emphasis on maximizing defect detection even at the cost of additional checks. The results obtained demonstrate the method's sufficiently high ability to identify defective components, which is critically important for test prioritization tasks, although the number of FPRs remains significant.

Figure 10 shows the results of Top-k ranking for the improved v2 method. A similar trend to the previous version is observed: as the proportion of selected components increases, the proportion of detected defects rises (to ≈ 0.49 at $k = 0.30$), while accuracy gradually decreases.

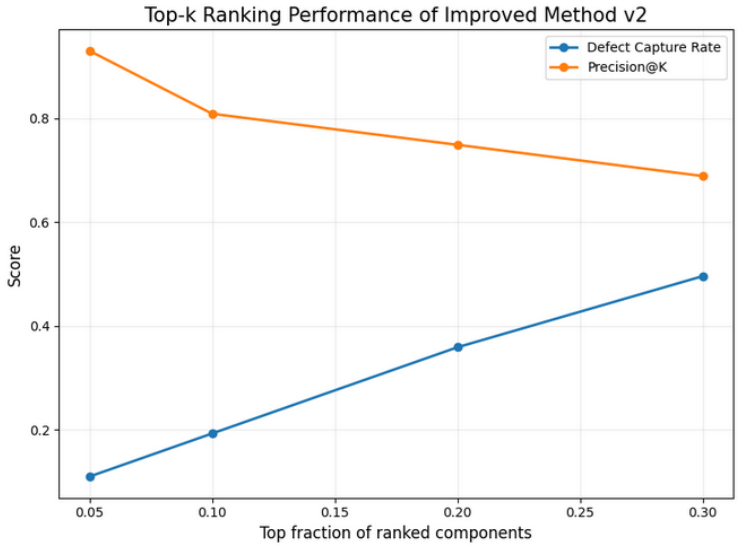

Fig. 10. Top-k ranking results

Compared to the previous version of the method, the improved version demonstrates higher accuracy at small values of k (particularly for $k = 0.05 – 0.10$), indicating a better concentration of defective components at the top of the ranking. This confirms the increased effectiveness of the method specifically for testing prioritization tasks, where it is critically important to correctly identify the highest-risk components at early stages.

Fig. 11 shows the distribution of the integral defect risk index for the improved v2 method. The index values are concentrated primarily in the range of 0.2–0.6, with a gradual decrease in frequency for high risk values.

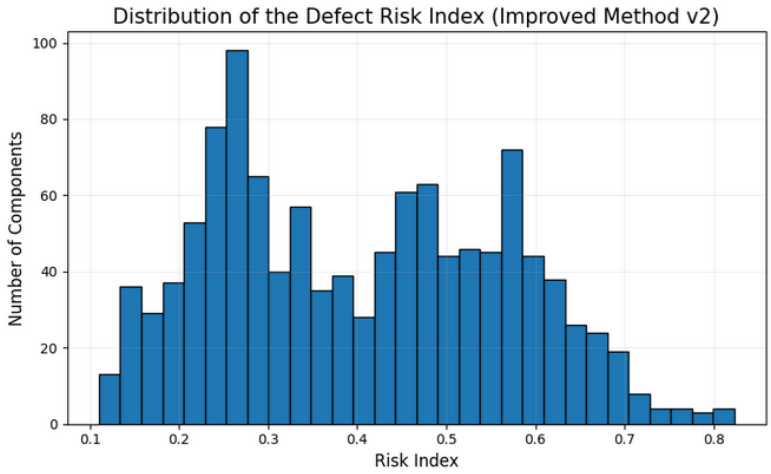

Fig. 11. Distribution of the integral defect risk index

Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

The distribution of the integral defect risk index appears natural and is easily interpreted. Risk values are concentrated primarily in the range of 0.15–0.65, with fewer components in the very low and very high risk zones. The presence of values above 0.6–0.8 indicates the identification of a limited number of high-risk components, which is important for effective test prioritization. This indicates that the index is not degenerate and actually forms a gradation of components by defect level and can be used for planning automated tests.

Values above 0.6–0.8 indicate the identification of a limited number of high-risk components, which is important for effective test prioritization. This indicates that the index is not degenerate and actually ranks components by defect severity, making it suitable for planning automated tests.

Figure 12 shows an assessment of the importance of features for the improved v2 method. The greatest contribution comes from the aggregated feature methods_times_coupling, which reflects the interaction between the complexity and coupling of components. Also significant are cbo_x_log_buggy_files, complexity_per_method, and age_x_complexity_per_method, which combine historical data with characteristics of structures and processes.

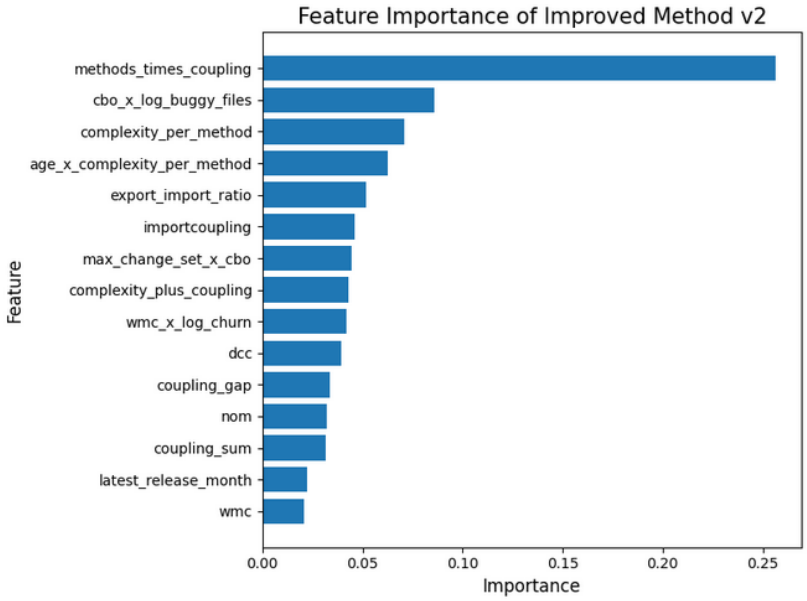

Fig. 12. Assessment of the importance of characteristics

The graph shows that the role of derived and composite attributes is growing, confirming the value of using them to improve the accuracy of defect risk assessment and better reflect complex relationships in the source code.

7. Discussion of the results

The results of the experimental study confirm the effectiveness of the developed method for assessing the risk of defects in software components within distributed computer systems. The study was conducted sequentially for several versions of the method, which differed in the composition of features and the method of calculating the integrated risk indicator, while defect probability was estimated using machine learning models.

A comparison of the basic (structural) and extended versions of the method showed that considering only the structural characteristics of the software code does not provide a sufficiently high-quality risk assessment. When process and context characteristics were added, the method's ability to distinguish between components with and without defects significantly improved. This indicates that defects in components are formed under the influence not only of the code's internal complexity and connectivity, but also of the change history, development intensity, and the overall state of the project.

In the first improved version of the method, an integrated risk index was implemented as a combination of the model's probabilistic forecast and generalized characteristics of various groups of features. The results showed that this approach does not lead to a significant

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189)
ISSN 3083-7650 (0135-1710) (print) Management Information System and Devices. 2026. No. 2 (189)

improvement in standard quality metrics compared to the extended version. At the same time, it allows moving from simple classification to a more meaningful representation of results in the form of a continuous risk indicator, which is fundamentally important for component ranking and testing prioritization tasks.

Further development of the method, implemented in version v2, is based on two key elements. First, an ensemble-based defect probability assessment scheme is used, which combines the results of several machine learning models. Second, derived features have been introduced that reflect the interaction between the characteristics of structures, processes, and contexts. This combination has improved the quality of the assessment and provided a more in-depth representation of the factors influencing the occurrence of defects in software components.

Analysis of ROC curves showed that the transition from the extended version of the method to the use of an ensemble model results in an increase in the ROC-AUC value. At the same time, the improved version of the method (v2) demonstrates the same level of generalization ability as the stack ensemble model, confirming the correctness of integrating the ensemble prediction into the method's structure. This means that the proposed risk index formulation scheme does not degrade prediction quality while preserving the advantages of the ensemble approach. A similar trend is observed for the Precision–Recall analysis. The use of an expanded set of features improves quality compared to the structural version of the method, while the application of the ensemble model allows for even better results. The improved version of the method, v2, demonstrates nearly identical values, which indicates the effectiveness of the chosen scheme for integrating the prediction and features.

Analysis of the error matrices shows that the improved versions of the method are geared toward more complete detection of component defects. The reduction in the number of missed defects is achieved at the cost of a certain increase in FPR triggers. This behavior is justified for automated test prioritization tasks, since missing a defective component is more critical than performing additional testing on a defect-free component. Thus, the method implements an evaluation strategy aimed at minimizing the risk of missing defects.

Particular attention should be paid to the component ranking results. The Top-k analysis showed that the largest proportion of defective components is concentrated at the top of the generated list. The improved version of the method, v2, provides higher accuracy at small values of k, indicating a better ability to identify components with the highest risk. This directly addresses the practical task of determining a limited set of components for priority testing.

The distribution of the integral risk index confirms the method's ability to differentiate components by risk level. The improved v2 version is characterized by a more uniform and stretched distribution of values, allowing for a clearer identification of groups of components with low, medium, and high risk. This creates a foundation for the method's further use in risk classification and decision-making tasks.

Feature importance analysis shows that in the early stages of the method's development, the main contribution came from classical structural characteristics, such as component complexity and connectivity. In the improved v2 version, the greatest weight is given to derived features that combine different types of characteristics. This indicates that the machine learning model is beginning to utilize more complex patterns related to the interaction of factors, rather than just their individual values. It is worth noting that the improvement in evaluation quality is achieved not only by changing the algorithm but also by changing the structure of the feature space. Summarizing the results obtained, we can conclude that the most effective version of the method is the one that combines an ensemble assessment of defect occurrence probability using derived features and an integral risk index. This scheme simultaneously ensures high prediction quality, interpretability of results, and suitability for automated testing prioritization tasks. This demonstrates the advisability of using a comprehensive approach to risk assessment that takes into account both the properties of the program code and the context of its development and evolution.

8. Conclusions

This paper addresses a scientific and applied problem involving the development of a method for assessing the risk of defects in software components of distributed computer systems for the purpose of prioritizing automated testing. An analysis of current scientific publications has shown that existing approaches are predominantly focused either on the use of individual structural metrics of program code or on the application of machine learning models without proper consideration

Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

of process characteristics and contexts, while the issue of integrating heterogeneous factors into a unified risk assessment system remains insufficiently addressed.

Within the scope of this study, a method was developed based on the formation of an integral defect risk indicator for software components using machine learning models, an extended feature space, and their generalization. The proposed scheme involves representing each component as a multidimensional feature vector that encompasses parameters of structures, processes, and contexts, as well as calculating defect probabilities using machine learning models and forming a risk index suitable for ranking components in automated testing tasks.

The use of an integrated risk index allows moving from binary classification to a ranking task, which is more relevant for automated testing practice. The Top-k analysis showed that a significant proportion of components with defects is concentrated at the top of the ranking, which enables effective prioritization of test checks when resources are limited. Feature importance analysis confirmed the feasibility of using derived characteristics that account for the interaction between code complexity, change intensity, and defect context.

The developed method can be used as an analytical support tool for the automated testing process in distributed computer systems. Prospects for further research include improving risk assessment models, reducing the number of FPR triggers, adapting threshold values to the specifics of particular software projects, and integrating the method into real-world software lifecycle management systems.

Conflict of Interest

The authors of this article declare that they have no conflicts of interest regarding this study, including financial, personal, authorship, or other conflicts that could influence the study and its results presented in this article.

Funding

The study was conducted without financial support.

Data Availability

The manuscript has no associated data.

Use of Artificial Intelligence

The authors confirm that they did not use artificial intelligence technologies in the creation of this work.

References

1. Dos Santos, R. A. (2024), "Just-In-Time Software Defect Prediction using a deep learning-based model", *New Trends in Computer Sciences*, 2(2), 91–100. DOI: https://doi.org/10.3846/ntcs.2024.22274
2. Jiang, Y., Shen, B., Gu, X., (2025), "Just-in-time software defect prediction via bi-modal change representation learning", *Journal of Systems and Software*. Volume 219. DOI: https://doi.org/10.1016/j.jss.2024.112253
3. Ali, M., Mazhar, T., Al-Rasheed, A., Shahzad, T., Yasin Ghadi, Y., Amir Khan, M., (2024), "Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning", *PeerJ Computer Science* 10:e1860. DOI: https://doi.org/10.7717/peerj-cs.1860
4. Albattah, W., Alzahrani, M., (2024), "Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach", *AI*, 5, 1743–1758. DOI: https://doi.org/10.3390/ai5040086
5. Siswantoro, M., Yuhana, U.L., (2023), "Software Defect Prediction Based on Optimized Machine Learning Models: A Comparative Study". Teknika. 12, 2, 166–172. DOI: https://doi.org/10.34148/teknika.v12i2.634
6. Guo, Z., Sun, Y., Fu, B., Cheng, X., (2022), "Research on software defect prediction method based on machine learning oriented to large-scale complex information system", *Proc. SPIE 12161, 4th International Conference on Informatics Engineering & Information Science (ICIEIS2021)*. DOI: https://doi.org/10.1117/12.2627233
7. Cabral, G.G., Minku, L.L., Oliveira, A.L., Pessoa, D.A., Tabassum, S., (2023), "An investigation of online and offline learning models for online Just-in-Time Software Defect Prediction", *Journal of Systems and Software* 28, 121. DOI: https://doi.org/10.1007/s10664-023-10335-6
8. Kaur, G., Pruthi, J., Gandhi, P., (2023), "Machine learning based Software Fault Prediction models," *Karbala International Journal of Modern Science*: Vol. 9. DOI: https://doi.org/10.33640/2405-609X.3297
9. Bayramova, T., (2023), "Software defect prediction using the machine learning methods", *Problems of Information Technology*, 14(2). DOI: http://doi.org/10.25045/jpit.v14.i2.03

ISSN 3083-7715 (online) *Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189)*
ISSN 3083-7650 (0135-1710) (print) *Management Information System and Devices. 2026. No. 2 (189)*

10. Shittu, A.S., Abdulsalami, B. (2023), "Software defect prediction using machine learning approach : a contemporary review", *TechRxiv*, pp. 1–8. DOI: https://doi.org/10.36227/techrxiv.23648907.v1
11. Hasan Kabir, S.M., Tanim Rahman, Md., Aunik Hasan Mridul, (2025), "Software Defect Prediction Using Traditional Machine Learning and Ensemble Learning Algorithms", *Smart Wearable Technology*, 1, A9. DOI: https://doi.org/10.47852/bonviewSWT52025645
12. Mahbub, P., Shuvo, O., Rahman, M.M.б (2023) "Defectors: A Large, Diverse Python Dataset for Defect Prediction", *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, Melbourne, Australia, pp. 393–397. DOI: https://doi.org/10.1109/MSR59073.2023.00085
13. Robredo, M., Esposito, M., Taibi, D., Peñaloza, R., Lenarduzzi, V., (2025), "SQuaD: The Software Quality Dataset – Dataset [Data set]". *Zenodo*. DOI: https://doi.org/10.5281/zenodo.17566691
14. Chen, Z., Wang, K., Yang, C., Cao, B., Fan, J., (2025), "Learning Top-k Classification with Label Ranking. In: Taniguchi, T., et al. Neural Information Processing. ICONIP 2025", *Communications in Computer and Information Science*, vol 2755. Springer, Singapore. DOI: https://doi.org/10.1007/978-981-95-4094-5_9
15. Hamid, M., Hajjej, F., Alluhaidan, A.S., (2025), "Fine tuned CatBoost machine learning approach for early detection of cardiovascular disease through predictive modeling", *Scientific Reports* 15, 31199. DOI: https://doi.org/10.1038/s41598-025-13790-x
16. F. Setievi, F., Natalia, J., Tjhang, T.R., Edbert, I. S., Suhartono, D., (2022), "A Comparative Study of Supervised Machine Learning Algorithms for Fake Review Detection," *2022 5th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, Yogyakarta, Indonesia, 2022, pp. 306–312. DOI: https://doi.org/10.1109/ISRITI56927.2022.10052860

Received (Надійшла) 28.02.2026
Accepted for publication (Прийнята до друку) 18.04.2026
Publication date (Дата публікації) 29.05.2026

Відомості про авторів / About the Authors

Дяченко Дмитро Олександрович – Харківський національний університет радіоелектроніки, аспірант кафедри електронних обчислювальних машин, Харків, Україна;
Dmytro Diachenko – Kharkiv National University of Radio Electronics Kharkiv, Postgraduate Student of the Department of Electronic Computers, Ukraine;
e-mail: dmytro.diachenko2@nure.ua
ORCID ID: http://orcid.org/0009-0006-5751-3511

Міхаль Олег Пилипович – доктор технічних наук, доцент, Харківський національний університет радіоелектроніки, професор кафедри електронних обчислювальних машин, Харків, Україна;
Oleg Mikhal – Doctor of Technical Sciences, Associate Professor, Kharkiv National University of Radio Electronics, Professor of the Department of Electronic Computers, Kharkiv, Ukraine;
e-mail: oleg.mikhal@nure.ua
ORCID ID: http://orcid.org/0000-0002-5977-3177

МЕТОД ОЦІНЮВАННЯ РИЗИКУ ДЕФЕКТІВ ПРОГРАМНИХ КОМПОНЕНТІВ РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМ ДЛЯ ПРІОРИТИЗАЦІЇ АВТОМАТИЗОВАНИХ ВИПРОБУВАНЬ

Предметом дослідження є методи, моделі й засоби оцінювання ризику дефектів програмних компонентів розподілених комп'ютерних систем для підтримки процесу пріоритизації автоматизованих випробувань. **Мета** – розроблення методу оцінювання ризику дефектів програмних компонентів розподілених комп'ютерних систем для пріоритизації автоматизованих випробувань на основі машинного навчання з огляду на структурні характеристики компонентів, інтенсивність змін і результати тестування. Відповідно до окресленої мети необхідно виконати такі **завдання:** проаналізувати сучасні підходи до прогнозування дефектів програмного забезпечення й пріоритизації автоматизованих випробувань у розподілених комп'ютерних системах; визначити інформативні

Автоматизовані системи управління та прилади автоматики. 2026. № 2 (189) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 2 (189) *ISSN 3083-7650 (0135-1710) (print)*

характеристики програмних компонентів, що відтворюють їх структурні властивості, інтенсивність змін і зв'язки між програмними компонентами; розробити метод оцінювання ризику дефектів програмних компонентів на основі моделі машинного навчання з формуванням інтегрального показника ризику; запропонувати підхід до пріоритизації автоматизованих випробувань на основі отриманого інтегрального показника ризику; впровадити запропонований метод і експериментально оцінити його ефективність за показниками класифікації та ранжування. **Методи:** системний аналіз, машинне навчання, аналіз структурних метрик програмного коду, класифікація, відбір інформативних ознак, формування інтегрального показника ризику й ризик-орієнтованого ранжування. Експериментальну перевірку виконано з використанням методів обчислювального моделювання. **Результати дослідження.** Розроблено метод оцінювання ризику дефектів програмних компонентів розподілених комп'ютерних систем, що забезпечує інтеграцію характеристик структур, процесів і контекстів, а також обчислення інтегрального показника ризику й ранжування компонентів для пріоритизації автоматизованих випробувань. Експериментальна перевірка підтвердила працездатність запропонованого методу та його спроможність ефективно розділяти програмні компоненти з дефектами та без них, а також виділяти компоненти ризику для першочергового тестового контролю. **Висновки.** Досягнуті результати підтвердили доцільність упровадження методів машинного навчання та інтегрованого простору ознак для оцінювання ризику дефектів програмних компонентів розподілених комп'ютерних систем. Запропонований метод може бути використаний як основа для підвищення обґрунтованості рішень щодо пріоритизації автоматизованих випробувань, вчасного виявлення найбільших компонентів ризику й подальшого розвитку інтелектуальних засобів контролю якості програмного забезпечення.

Ключові слова: програмні компоненти; розподілені комп'ютерні системи; автоматизовані випробування; машинне навчання; прогнозування дефектів; інтегральний показник ризику; пріоритизація; тестування; *Python; Google Colab*.

Бібліографічні описи / Bibliographic descriptions

Дяченко Д. О., Міхаль О. П. Метод оцінювання ризику дефектів програмних компонентів розподілених комп'ютерних систем для пріоритизації автоматизованих випробувань. *Автоматизовані системи управління та прилади автоматики*. 2026. № 2 (189). С. 109–127. DOI: https://doi.org/10.30837/0135-1710.2026.189.109

Diachenko, D., Mikhal, O. (2026), "Method for assessing the risk of defects in software components of distributed computer systems for automated test prioritization", *Management Information System and Devices*, No. 2 (189), P. 109–127. DOI: https://doi.org/10.30837/0135-1710.2026.189.109