This work investigates fuzzy algorithmic models for predicting software defects. An issue related to conventional machine (deep) learning models is complexity and interpretability. In addition, cross-project learning of such models requires solving the problem of heterogeneity in the initial and target data distributions.

A fuzzy algorithmic model for predicting software defects with an integrated distribution of implementation options for development stages has been proposed. A genetic-neural method for tuning a fuzzy algorithmic model to cross-project data has been devised.

Unlike machine learning models, the interpretability of the prediction model was achieved by assessing the correctness (defectivity) of development stages using fuzzy rules. The model is built on the basis of "work-control-refinement" algorithmic structures, which is an analog of a fuzzy knowledge base. Fuzzy estimates of the defectivity of the execution of work, control, and refinement operators are subject to tuning, where defect ranks model the distribution of resources. The training sample is formed on the basis of estimates of the correctness of implementation options for algorithmic structures.

The integration of indicators for implementation options for work, control, and refinement operators has made it possible to solve the issue of heterogeneity in the initial and target data distributions. Unlike known methods for training neural-fuzzy models of software reliability, simplification of the tuning process was achieved by transferring cross-project data.

The scope of practical application includes predicting the quality of new software based on the experience of completed projects. The condition of use is a discrete algorithmic model of the development process

Keywords: software defect prediction, fuzzy algorithmic model, cross-project transfer learning

UDC 004.41
DOI: 10.15587/1729-4061.2026.362735

DEVISING A CROSS-PROJECT LEARNING METHOD FOR SOFTWARE DEFECT PREDICTION BASED ON FUZZY ALGORITHMIC MODELS

Hanna Rakytyanska
Corresponding author
Candidate of Technical Sciences*
E-mail: rakit@vntu.edu.ua
ORCID: https://orcid.org/0000-0001-5863-3730
Bohdan Prus
PhD Student*
ORCID: https://orcid.org/0009-0008-7214-0949
*Department of Software Engineering
Vinnytsia National Technical University
Khmelnytske highway, 95, Vinnytsia, Ukraine, 21021

Received 09.03.2026
Received in revised form 18.05.2026
Accepted date 27.05.2026
Published date 30.06.2026

How to Cite: *Rakytyanska, H., Prus, B. (2026). Devising a cross-project learning method for software defect prediction based on fuzzy algorithmic models. Eastern-European Journal of Enterprise Technologies, 3 (2 (141)), 45–55. https://doi.org/10.15587/1729-4061.2026.362735*

1. Introduction

Software defect prediction is aimed at identifying problems in the code at early stages of the life cycle, which could reduce development costs. Conventionally, this task is solved based on statistical methods, machine learning methods, and evolutionary algorithms [1]. The model classifies modules as potentially defective (defect-free) based on code quality metrics. Building a prediction model includes selecting features (code metrics); selecting projects for the training sample; selecting models for the classifier ensemble. The complexity and interpretability of such models is still a problem. In addition, there is an issue of heterogeneity of the source and target data distributions for cross-project learning [2].

Modeling processes associated with introducing, detecting, and correcting errors can be carried out based on the theory of fuzzy reliability of algorithmic processes [3, 4]. The development process is described by means of algorithm algebra. A fuzzy reliability model connects estimates of the correctness (defectiveness) of the performance of work, control, and refinement operations. Adhering to the principles of fuzzy modeling of algorithmic processes, a method of software reliability analysis based on algorithm algebra and fuzzy logic was proposed in [5]. This approach makes it possible to implement the process of constructing and training a fuzzy algorithmic model by distilling knowledge [6]. As a result of transfer learning, reliable elements of completed projects are transferred to the logical-algorithmic model of the current project. In this case, the training sample is formed based on data from completed projects with different options for performing work, control, and refinement operations [3, 4]. This will ensure the alignment of the initial and target data distribution and simplify the process of cross-project learning.

In practice, this will make it possible to predict software defects depending on the implementation options for the development stages and resource allocation. Therefore, research on constructing and configuring a fuzzy algorithmic model for predicting software defects is relevant.

2. Literature review and problem statement

Count-based code quality metrics identify object-oriented features. However, such metrics are unable to reflect the structural features of the code. In [7], structural metrics were investigated that take into account class complexity, connectivity, task dependences, abstraction, inheritance, and polymorphism. The accuracy of such models is low due to the lack of relevant data, feature correlation, and class imbalance. To

improve accuracy, ensemble models are used [8–11]. In [8, 9], logistic regression, decision trees, support vector machines, and naive Bayesian methods were investigated as basic classifiers. However, the accuracy of basic models remains low due to the inability to extract hierarchical features. In [10], deep learning algorithms are used to search for defect patterns in data. However, conventional ensemble models are unable to provide accuracy due to the fixed weights of the basic classifiers. In [11], a new ensemble learning method with dynamic weights of the base classifiers depending on the cross-project data fusion method was proposed. However, low accuracy remains a common problem of machine learning methods due to the instability of features.

Software defect prediction involves the representation of the source code in a form suitable for training. In [12, 13], prediction models are based on features selected based on expert knowledge. The robustness of estimates remains an open question since the use of machine (deep) learning models has led to problems of reproducibility. Therefore, in [14–16], representation learning is used, when the machine learning model automatically creates a code representation by selecting relevant features. In [14], natural language processing models are used for representation training, which make it possible to detect semantic features of the code at the level of language constructs. The task of substantiating the decisions of the prediction model remains unsolved because one language model cannot effectively operate with different data formats. Therefore, in [15, 16], ensemble learning methods are used to create a structured representation of the code. In work [15], basic deep models explore complex feature spaces, and categorical boosting is used to detect defects. In [16], the meta-model forms a set of code representations at the syntactic and semantic levels, taking into account the formats of the input and output data. However, the resulting representation is not universal for a wide class of tasks because it is limited to the classes of tasks on which the model was trained.

Cross-project data transfer technologies are used to predict software defects [17–19]. Machine learning algorithms assume that the training and test samples have similar data distribution. However, for cross-project learning, when the training and test data belong to different projects, this requirement is not met. To solve this problem, measures of code metric correspondence were proposed in [17]. However, this approach is not justified. In practice, only a small set of source (training) data can fully correspond to the target (test) data set. Therefore, in [18], a method for adapting a weighted balanced distribution was devised, which makes it possible to assess the importance of differences in data distribution. This method cannot be applied to deep learning algorithms because it works only with counted code metrics. Deep transfer learning technologies are used for semantic features of code in [19, 20]. In [19], a deep model transfers features from the source projects to the target project using a meta-estimator that minimizes differences in data distributions. For the case of many source projects, a transfer ensemble learning method is used in [20]. The problem is the interpretability of cross-project learning models, i.e., the assessment of the amount of knowledge that can be transferred to the current project from the source data sets [20]. The issue arises due to the knowledge transfer mechanism that generates weights for the source data sets depending on the importance of the features and the difference in the data distribution.

For machine (deep) learning models, there is still a problem of uncertainty in the data. In [21, 22], a neural-fuzzy classifier is included in the ensemble of basic models to deal with uncertainty in code metrics. An open question is the formalization of code metrics by membership functions. The construction of membership functions under the condition of heterogeneous small samples is problematic, which can significantly affect the accuracy of the model. To solve the issue of data uncertainty, the authors of [23, 24] use fuzzy measures of similarity of projects. In work [23], scalar measures based on distance are used. In [24], fuzzy relationship matrices, built into the deep learning model, model the partial membership of data to the target sample. However, the ambiguity of predictions due to incomplete data does not make it possible to correctly estimate the growth of reliability function. Therefore, in [25], recurrent neural-fuzzy networks were developed that model the growth of the reliability function of dynamic systems with feedback. However, neural-fuzzy classifiers are not able to obtain hierarchical features of the code because such classifiers have a simplified structure. In [26], input features are obtained using an ensemble of deep neural networks. The meta-model is built on the basis of an expert rule base, where fuzzy estimates of code metrics are formalized by interval membership functions. The problem is the complexity of constructing fuzzy process-oriented reliability models. This issue is due to the lack of mechanisms for obtaining hierarchical features for constructing interpretable deep fuzzy classifiers.

Thus, none of the machine (deep) learning models is universal for predicting software defects. The development of interpretable models for predicting software defects remains an unresolved issue. Ensuring accuracy requires a more complex hierarchical machine learning model, the interpretability of which is problematic. For cross-project transfer learning, this means granulation and assessment of the amount of knowledge that can be transferred from the original projects to the current project. The issue of lack of data for cross-project learning due to the heterogeneity and uncertainty of the original and target distributions is still unresolved. Projects that were implemented under different resource constraints may use different options for completing tasks. However, process-oriented models for predicting software defects that integrate resource allocation depending on the implementation options for development stages are absent.

The above allows us to argue that it is advisable to conduct a study aimed at devising a method for cross-project transfer training of a fuzzy model of the software development process.

3. The aim and objectives of the study

The aim of our study is to devise a cross-project learning method for predicting software defects based on fuzzy algorithmic models. This will make it possible to simplify the tuning process by transferring knowledge from the original projects to the current project.

To achieve the goal, the following tasks were set:

– to build a fuzzy algorithmic model of software defects with an integrated distribution of implementation options for development stages;

– to devise a genetic-neural method for tuning a fuzzy algorithmic model of software defect prediction on cross-project data;

– to verify the performance of the method for predicting defects in a software system for media content aggregation.

4. The study materials and methods

4. 1. The object and hypothesis of the study

The object of our study is fuzzy algorithmic models of software defect prediction. Such models describe the software development process, which is associated with the introduction, detection, and correction of errors. This allows us to formulate the principal hypothesis of the work, according to which the software development process can be considered a multidimensional discrete process with a m-ary concept of defects [3, 4].

The assumptions adopted in the study are based on modeling the software development process using algorithm algebra and fuzzy logic [3, 4]. The structure of the model is determined by the sequence of algorithmic structures "work-control-refinement". It is assumed that the logical-algorithmic model of the development process is an analog of a fuzzy knowledge base. Then the system of fuzzy logical equations connects fuzzy assessments of the correctness (defectivity) of working, controlling, and refinement operators with the possibility of correct (with defects) performance of the task.

The fuzzy algorithmic model requires tuning the structure and parameters to cross-project data. The simplifications adopted in the study are based on the formalization of improving transformations of the logical-algorithmic model using control variables. The incorporation of indicators of techniques for performing working, controlling, and refinement operators into the model makes it possible to solve the issue of heterogeneous data. Knowledge distillation is carried out at the level of elements of the logical-algorithmic model [6]. The selection of implementation options is formalized by improving transformations of the system of fuzzy gradation rules [27, 28]. Thus, the tuning process is simplified due to the transfer of cross-project data. As a result, the model determines the level of preference of options depending on the ranks (criticality levels) of defects.

4. 2. Conditions for conducting the experiment

A fuzzy algorithmic model predicts the quality of a new software product based on the experience of completed projects that form the training sample. The model is tested at the stages of development of the current project. The distribution of the training and test data sets is aligned by taking into account possible options for performing tasks. The implementation of the method assumes the presence of expert or experimental assessments of the correctness (defectivity) of the implementation options for working, controlling, and refinement operations. The structure of the logical-algorithmic model is configured using a genetic algorithm that transfers reliable elements of completed projects to the current project. The parameters of the logical-algorithmic model are configured using a neural-fuzzy network that determines the ranks of defects for the selected structure of the development process. The justification of predictive solutions is based on the ranking of implementation options for working, controlling, and refinement operations by the level of impact on the quality of the software.

The performance of the method was tested for predicting defects in a media content aggregation software system. The software implementation of the method is written in Python and is designed to process expert and experimental data coming from the development environment of the target project. The developed software is synchronized with the current state of the development process to refine the predictive defect estimates. The software implementation of the method does not require specialized equipment because the analysis is based on structured tables of inter-project indicators.

5. Results of investigating the cross-project transfer learning method

5. 1. Fuzzy algorithmic model of software defect prediction

In the algebra of algorithms, the software development process with the sequential execution of stages A_i, $i = 1, ..., n$, is described by a linear structure of the following form [3, 4]

$$C = A_1(\mathbf{d}_1, \mathbf{W}_1), ..., A_n(\mathbf{d}_n, \mathbf{W}_n), \quad (1)$$

where $\mathbf{d}_i = \{d_{i1}, \dots, d_{iq_i}\}$ is the set of defects that are the result of errors at stage A_i; $\mathbf{W}_i = (w_{i1}, \dots, w_{iq_i})$ is the vector of ranks (criticality levels) of defects $d_{i1}, \dots, d_{iq_i}$; C is the operator corresponding to the linear algorithmic structure.

Work processes at stage A_i are described by the iterative algorithmic structure "work-control-refinement" [3, 4]

$$B_i = R_i \underset{\Omega_i}{\{U_i\}}^{\lambda_i}, \quad (2)$$

where R_i is the work operator; Ω_i is the control operator; U_i is the refinement operator; λ_i is the criticality level of repeated defects; B_i is the operator corresponding to the iterative algorithmic structure.

During the execution of the work operator, errors can be introduced. During the execution of the control operator, the errors introduced are subject to detection. During the execution of the refinement operator, the detected errors are subject to removal.

Let $\mathbf{r}_i = \{r_{i1}, \dots, r_{iL_i}\}$, $\boldsymbol{\omega}_i = \{\omega_{i1}, \dots, \omega_{iK_i}\}$, $\mathbf{u}_i = \{u_{i1}, \dots, u_{iP_i}\}$ be the set of ways to execute operators R_i, Ω_i, U_i; $\mathbf{g}_i = \{g_{i1}, \dots, g_{iH_i}\}$ be the set of options for the software implementation of operators R_i and U_i; $\mathbf{X}_i = (x_{i1}, \dots, x_{iL_i})$, $\mathbf{Y}_i = (y_{i1}, \dots, y_{iK_i})$, $\mathbf{Z}_i = (z_{i1}, \dots, z_{iP_i})$, x_{il}, y_{ik}, $z_{ip} \in \{0, 1\}$, – vectors of indicators of the ways of performing operators R_i, Ω_i, U_i; $\boldsymbol{\alpha}_i = (\alpha_{i1}, \dots, \alpha_{iH_i})$, $\boldsymbol{\beta}_i = (\beta_{i1}, \dots, \beta_{iH_i})$, α_{ih}, $\beta_{ih} \in \{0, 1\}$, – vectors of indicators of the options for the software implementation of operators R_i, U_i.

The choice of techniques of performing and the options for the software implementation of the work, control, and refinement operators is formalized by means of improving transformations of the logical-algorithmic model (2)

$$B_i = R_i(\mathbf{X}_i, \boldsymbol{\alpha}_i) \underset{\Omega_i(\mathbf{Y}_i)}{\{U_i(\mathbf{Z}_i, \boldsymbol{\beta}_i)\}}^{\lambda_i}, \quad (3)$$

where:

$$R_i = \left[\bigvee_{l=1}^{L_i}(x_{il} r_{il})\right] \wedge \left[\bigvee_{h=1}^{H_i}(\alpha_{ih} g_{ih})\right],$$

$$\Omega_i = \bigvee_{k=1}^{K_i}(y_{ik} \omega_{ik}),$$

$$U_i = \left[\bigvee_{p=1}^{P_i}(z_{ip} u_{ip})\right] \wedge \left[\bigvee_{h=1}^{H_i}(\beta_{ih} g_{ih})\right]. \quad (4)$$

The correctness of execution of the logical-algorithmic model (3), (4) is determined by a system of fuzzy rules:

IF operators R_i and $\Omega_i$are executed correctly
OR operator R_i is executed with defects
AND operators Ω_i, U_i are executed correctly,
THEN stage A_i is executed correctly,
ELSE stage A_i is executed with defects. (5)

Let $\mathbf{v} = \{v_1, \ldots, v_{N_i}\}$ be the set of execution options for operator $F_I \in \{R_i, \Omega_i, U_i\}$.

The correctness of execution of option $v_I \in \mathbf{v}$ is evaluated as follows [4]

$$\mu^1_{v_I} = \bigwedge_{j=1}^{q_j}\left[1-\left(\mu^{0(j)}_{v_I}\right)^{w_{ij}}\right],\ I=1,...,N_i, \quad (6)$$

where $\mu^1_{v_I}\left(\mu^{0(j)}_{v_I}\right)$ is the possibility of correct execution (with defects of the jth type) of variant v_I of operator F_I.

The ranks (levels of criticality) of defects λ_i, $w_{ij} \in \{1, 3, 5, 7, 9\}$are chosen according to the Saati scale and model the distribution of resources [4].

The variant for which the degree of belonging (6) is maximal should be considered as the optimal variant of the implementation of operator $F_I \in \{R_i, \Omega_i, U_i\}$.

From the base of rules (5), a system of fuzzy logic equations follows, which connects the evaluations of the correctness of the execution of work, control, and refinement operators with the evaluation of the correctness (defectiveness) of the execution of stage A_i

$$\mu^1_{B_i} = 1-\left(\mu^{00}_{R_i}\right)^{\lambda_i},\ \ \mu^0_{B_i} = 1-\mu^1_{B_i}, \quad (7)$$

where

$$\mu^{00}_{R_i} = 1-\begin{bmatrix}\mu^1_{R_i}\left(\mathbf{X}_i,\boldsymbol{\alpha}_i\right)\cdot\mu^1_{\Omega_i}\left(\mathbf{Y}_i\right)+\\ +\mu^0_{R_i}\left(\mathbf{X}_i,\boldsymbol{\alpha}_i\right)\cdot\mu^1_{\Omega_i}\left(\mathbf{Y}_i\right)\cdot\mu^1_{U_i}\left(\mathbf{Z}_i,\boldsymbol{\beta}_i\right)\end{bmatrix}.$$

Here $\mu^1_{R_i}\left(\mu^0_{R_i}\right)$, $\mu^1_{\Omega_i}\left(\mu^0_{\Omega_i}\right)$, $\mu^1_{U_i}\left(\mu^0_{U_i}\right)$ is the possibility of correct (with defects) execution of work (R_i), control (Ω_i), and refinement (U_i) operators; $\mu^{00}_{R_i}$ is the possibility of skipping errors during control and introducing new errors during refinement; $\mu^1_{B_i}\left(\mu^0_{B_i}\right)$ is the possibility of correct (with defects) execution of the iterative algorithmic structure B_i.

The estimates of correct (with defects) execution of operators R_i, Ω_i, U_i are determined based on the estimates of the correctness of their implementation options according to the ranks (criticality levels) of defects:

$$\mu^1_{R_i} = \left[\bigvee_{l=1}^{L_i}\left(x_{il}\cdot\mu^1_{r_{il}}\left(\mathbf{W}_i\right)\right)\right]\wedge\left[\bigvee_{h=1}^{H_i}\left(\alpha_{ih}\cdot\mu^1_{g_{ih}}\left(\mathbf{W}_i\right)\right)\right],$$

$$\mu^0_{R_i} = 1-\mu^1_{R_i},$$

$$\mu^1_{\Omega_i} = \bigvee_{k=1}^{K_i}\left(y_{ik}\cdot\mu^1_{\omega_{ik}}\left(\mathbf{W}_i\right)\right),\ \ \mu^0_{\Omega_i} = 1-\mu^1_{\Omega_i},$$

$$\mu^1_{U_i} = \left[\bigvee_{p=1}^{P_i}\left(z_{ip}\cdot\mu^1_{u_{ip}}\left(\mathbf{W}_i\right)\right)\right]\wedge\left[\bigvee_{h=1}^{H_i}\left(\beta_{ih}\cdot\mu^1_{g_{ih}}\left(\mathbf{W}_i\right)\right)\right],$$

$$\mu^0_{U_i} = 1-\mu^1_{U_i}, \quad (8)$$

The correctness (defectivity) of the development process is assessed as follows:

$$\mu^1_C = \prod_{i=1}^{n}\left(\mu^1_{B_i}\left(\mathbf{X}_i,\boldsymbol{\alpha}_i,\mathbf{Y}_i,\mathbf{Z}_i,\boldsymbol{\beta}_i,\mathbf{W}_i,\lambda_i\right)\right),$$

$$\mu^0_C = 1-\mu^1_C,$$

where $\mu^1_C\left(\mu^0_C\right)$ is the possibility of correct (with defects) execution of the linear algorithmic structure (1).

5. 2. A method for setting up a fuzzy algorithmic model for predicting software defects

5. 2. 1. The task of setting up the forecasting model

Let $\boldsymbol{\mu}^0_v$ be a fuzzy matrix of evaluations of defectiveness of variants v_I of the implementation of operator $F_I \in \{R_i, \Omega_i, U_i\}$, where:

$$\boldsymbol{\mu}^0_v = \begin{array}{c} \\ v_1 \\ \ldots \\ v_{N_i}\end{array}\begin{array}{c} \begin{array}{ccc} d_{i1} & \ldots & d_{iq_i}\end{array} \\ \begin{bmatrix}\mu^{0(1)}_{v_1} & \ldots & \mu^{0(q_i)}_{v_1}\\ \ldots & \ldots & \ldots\\ \mu^{0(1)}_{v_{N_i}} & \ldots & \mu^{0(q_i)}_{v_{N_i}}\end{bmatrix}\end{array}.$$

Relations (6) to (8) define a fuzzy algorithmic model for predicting software defects as follows:

$$\mu^1_{R_i} = f^1_{R_i}\left(\mathbf{X}_i,\boldsymbol{\alpha}_i,\boldsymbol{\mu}^0_{r_i},\boldsymbol{\mu}^0_{g_i},\mathbf{W}_i\right),\ \ f^0_{R_i} = 1-f^1_{R_i};$$

$$\mu^1_{\Omega_i} = f^1_{\Omega_i}\left(\mathbf{Y}_i,\boldsymbol{\mu}^0_{\omega_i},\mathbf{W}_i\right),\ \ f^0_{\Omega_i} = 1-f^1_{\Omega_i};$$

$$\mu^1_{U_i} = f^1_{U_i}\left(\mathbf{Z}_i,\boldsymbol{\beta}_i,\boldsymbol{\mu}^0_{u_i},\boldsymbol{\mu}^0_{g_i},\mathbf{W}_i\right),\qquad f^0_{U_i} = 1-f^1_{U_i};$$

$$\mu^1_{B_i} = f^1_{B_i}\left(f^1_{R_i},f^1_{\Omega_i},f^1_{U_i},\lambda_i\right),\ \ f^0_{B_i} = 1-f^1_{B_i}, \quad (9)$$

where $\boldsymbol{\mu}^0_{r_i},\boldsymbol{\mu}^0_{\omega_i},\boldsymbol{\mu}^0_{u_i}$ – fuzzy matrices of defectivity estimates of the methods for executing operators R_i, Ω_i, U_i; $\boldsymbol{\mu}^0_{g_i}$ – fuzzy matrices of defectivity estimates of the software implementation options of operators R_i, U_i; $f^1_{R_i}\left(f^0_{R_i}\right)$, $f^1_{\Omega_i}\left(f^0_{\Omega_i}\right)$, $f^1_{U_i}\left(f^0_{U_i}\right)$ – fuzzy correctness (defectiveness) functions of operators R_i, Ω_i, U_i; $f^1_{B_i}\left(f^0_{B_i}\right)$ – fuzzy correctness (defectiveness) function of operator structure B_i.

Let the training sample be obtained from the initial projects in the form of M pairs of cross-project data

$$\left\langle\hat{\mathbf{X}}^s_i,\hat{\boldsymbol{\alpha}}^s_i,\hat{\mathbf{Y}}^s_i,\hat{\mathbf{Z}}^s_i,\hat{\boldsymbol{\beta}}^s_i,\hat{\mu}^{1,s}_{B_i}\right\rangle,\ i=1,...,n,\ \ s=1,...,M,$$

where $\hat{\mathbf{X}}^s_i = \left(\hat{x}^s_{i1},...,\hat{x}^s_{iL_i}\right)$, $\hat{\mathbf{Y}}^s_i = \left(\hat{y}^s_{i1},...,\hat{y}^s_{iK_i}\right)$, $\hat{\mathbf{Z}}^s_i = \left(\hat{z}^s_{i1},...,\hat{z}^s_{iP_i}\right)$ – vectors of indicators of ways for performing operators R_i, Ω_i, U_i at stage A_i in project s; $\hat{\boldsymbol{\alpha}}^s_i = \left(\hat{\alpha}^s_{i1},...,\hat{\alpha}^s_{iH_i}\right)$, $\hat{\boldsymbol{\beta}}^s_i = \left(\hat{\beta}^s_{i1},...,\hat{\beta}^s_{iH_i}\right)$– vectors of indicators of options of software implementation of operators R_i and U_i at stage A_i in project s; $\hat{\mu}^{1,s}_{B_i}$ – assessment of correctness of execution of operator structure B_i in project s.

The assessment of $\hat{\mu}^{1,s}_{B_i}$ correctness is interpreted as the percentage of correctly performed tasks at stage A_i in project s.

It is necessary to find fuzzy matrices of defectivity estimates ($\boldsymbol{\mu}^0_{r_i},\boldsymbol{\mu}^0_{\omega_i},\boldsymbol{\mu}^0_{u_i}\ \boldsymbol{\mu}^0_{g_i}$) and vectors of defect ranks ($\mathbf{W}_i, \lambda_i$), which provide the minimum distance between model and experimental estimates of correctness of execution of operator structures (B_i)

$$\sum_{s=1}^{M}\sum_{i=1}^{n}\left[f^1_{B_i}\begin{pmatrix}f^1_{R_i}\left(\hat{\mathbf{X}}^s_i,\hat{\boldsymbol{\alpha}}^s_i\right),\\ f^1_{\Omega_i}\left(\hat{\mathbf{Y}}^s_i\right),f^1_{U_i}\left(\hat{\mathbf{Z}}^s_i,\hat{\boldsymbol{\beta}}^s_i\right)\end{pmatrix}-\hat{\mu}^{1,s}_{B_i}\right]^2 = \min_{\boldsymbol{\mu}^0_v,\mathbf{W}_i,\lambda_i}. \quad (10)$$

To solve the optimization problem (10), a genetic-neural approach is proposed. The genetic algorithm is used to adjust the structure of the logical-algorithmic model. The neural-fuzzy network is used to adaptively adjust the model parameters as new cross-project data is received.

5. 2. 2. Genetic-neural method

The genetic algorithm implements the idea of transferring reliable elements of completed projects to the current project. The transfer can be carried out both at the level of implementation options for individual operators and at the level of implementation of operator structures. At the coding stage, elements $\hat{\mathbf{X}}_i$, $\hat{\boldsymbol{\alpha}}_i$, $\hat{\mathbf{Y}}_i$, $\hat{\mathbf{Z}}_i$, $\hat{\boldsymbol{\beta}}_i$ of the training sample are transferred to the current population of chromosomes. This makes it possible to rank the options for performing work, control, and refinement operators. Then the genetic algorithm generates fuzzy matrices of defect estimates $\mu^0_{r_i}$, $\mu^0_{\omega_i}$, $\mu^0_{u_i}$ and $\mu^0_{g_i}$ based on the correctness estimates $\hat{\mu}^1_{B_i}$ of completed projects. The best chromosome determines the level of preference of options v_I for the requirements of the current project, which are determined by defect ranks $\mathbf{W}_i$.

A chromosome is defined as a vector-string of binary codes of elements of fuzzy matrices of defect estimates $\mu^{0(j)}_{r_{il}}$, $\mu^{0(j)}_{\omega_{ik}}$, $\mu^{0(j)}_{u_{ip}}$, $\mu^{0(j)}_{g_{ih}}$, and elements of defect rank vectors w_{ij}, λ_i (Fig. 1).

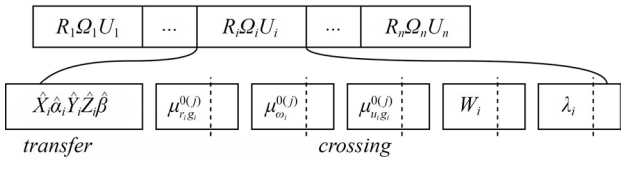

Fig. 1. Chromosome structure for transfer genetic algorithm

The crossover operation is performed by exchanging parts of chromosomes in matrices $\mu^0_{r_i}$, $\mu^0_{\omega_i}$, $\mu^0_{u_i}$, $\mu^0_{g_i}$, and vectors $\mathbf{W}_i$, λ_i. The correspondence function is built on the basis of criterion (10). Selection consists in selecting chromosomes that provide a threshold level of reliability $\mu^1_{B_i} \geq \underline{\mu}^1_{B_i}$ with minimal costs.

Neural-fuzzy algorithmic prediction model, isomorphic to the fuzzy knowledge base (5), is shown in Fig. 2. The network has a hierarchical structure for predicting the correctness (defectivity) of the execution of operators and operator structures. For operators R_i, Ω_i, U_i, matrices of fuzzy defectivity estimates $\mu^0_{r_i}$, $\mu^0_{\omega_i}$, $\mu^0_{u_i}$, $\mu^0_{g_i}$ are implanted in the neural network. Then the defectivity membership functions and their ranks (concentration parameters) are subject to training. For operator structures B_i, fuzzy rules are implanted in the neural network. Then the criticality levels of repeated defects are subject to training.

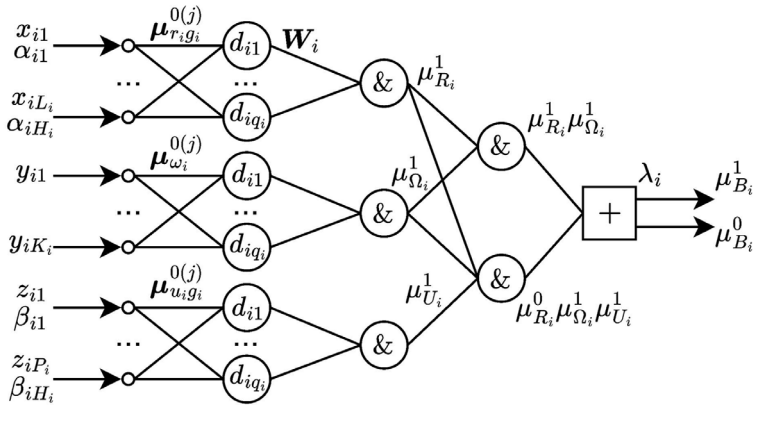

Fig. 2. Neural-fuzzy algorithmic model for predicting software defects

The network inputs are indicators of the implementation options of the work (x_{il}, α_{ih}), control (y_{ik}) and refinement (z_{ip}, β_{ih}) operators. In the intermediate layers, estimates of the correctness of the implementation options are formed $\mu^1_{r_{il}}$, $\mu^1_{\omega_{ik}}$, $\mu^1_{u_{ip}}$, $\mu^1_{g_{ih}}$, which determine the correctness of the execution of the work, control, and refinement operations $\mu^1_{R_i}$, $\mu^1_{\Omega_i}$, $\mu^1_{U_i}$. At the network outputs, an estimate of the correctness (defectivity) of the control $\mu^1_{R_i}\mu^1_{\Omega_i}$ and refinement $\mu^0_{R_i}\mu^1_{\Omega_i}\mu^1_{U_i}$, stages is formed, which determine the correctness (defectivity) of the execution of algorithmic structures B_i, $i = 1, ..., n$.

To set the parameters of the neural-fuzzy network, the following recurrent relations are used:

$$\mu^{0(j)}_{v_I}(t+1) = \mu^{0(j)}_{v_I}(t) - \eta\frac{\partial\varepsilon^{B_i}_t}{\partial\mu^{0(j)}_{v_I}(t)},$$

$$w_{ij}(t+1) = w_{ij}(t) - \eta\frac{\partial\varepsilon^{B_i}_t}{\partial w_{ij}(t)},$$

$$\lambda_i(t+1) = \lambda_i(t) - \eta\frac{\partial\varepsilon^{B_i}_t}{\partial\lambda_i(t)}, \quad (11)$$

which minimize the criterion

$$\varepsilon^{B_i}_t = \frac{1}{2}\left(\hat{\mu}^1_{B_i}(t) - \mu^1_{B_i}(t)\right)^2,$$

where $\hat{\mu}^1_{B_i}(t)$, $\mu^1_{B_i}(t)$ – experimental and theoretical estimates of the correctness of the execution of operator structures B_i at the t-th training step; $\mu^{0(j)}_{v_I}(t)$ – fuzzy estimates of defectiveness in the execution of variant v_I of operator $F_I \in \{R_i, \Omega_i, U_i\}$ at the t-th training step; $w_{ij}(t)$ – ranks of d_{ij} defects at the t-th training step; $\lambda_i(t)$ – criticality level of repeated defects at the t-th training step; η – training parameter.

Partial derivatives in relation (11) characterize the sensitivity of error $\varepsilon^{B_i}_t$ to changes in network parameters and are calculated according to [4].

5. 3. Verifying the performance of a method for predicting defects in a software system for media content aggregation

The task of predicting defects in a software system for media content aggregation is considered [5, 29]. The process of developing a software system includes the following stages

$$C = A_1[\mathbf{d}_1]...A_{10}[\mathbf{d}_{10}],$$

where the following types of defects are possible at the output of stages A_i:

A_1 – design of a media content aggregation system;

d_{11} – incomplete consideration of requirements;

d_{12} – missing dependences between modules;

d_{13} – inconsistency with the system's specifics;

d_{14} – critical scenarios not taken into account;

d_{15} – violation of component consistency;

A_2 – development of a media content acquisition module;

d_{21} – incomplete processing of formats;

d_{22} – loss of part of the data;

d_{23} – incorrect metadata;

d_{24} – critical scenarios without tests;

A_3 – development of a media content preprocessing module;
d_{31} – loss of data quality;
d_{32} – missing anomalies;
d_{33} – deletion of relevant data;
d_{34} – failure under specific conditions;
d_{35} – reduced data processing performance;
A_4 – design of the media content aggregation module;
d_{41} – incorrect grouping;
d_{42} – missing details;
d_{43} – low performance;
d_{44} – failures under special conditions;
d_{45} – compatibility conflicts;
A_5 – design of the data storage module;
d_{51} – violation of data integrity;
d_{52} – data loss;
d_{53} – incorrect display;
d_{54} – reduced performance;
d_{55} – compatibility violations;
A_6 – design of the user interface module;
d_{61} – inconsistency;
d_{62} – incorrect interaction;
d_{63} – incorrect response to changes;
d_{64} – customization limitations;
d_{65} – errors on different devices;
A_7 – module integration;
d_{71} – module incompatibility;
d_{72} – data loss;
d_{73} – delays;
d_{74} – stability violations;
A_8 – system testing;
d_{81} – incomplete coverage;
d_{82} – missed logic errors;
d_{83} – false positives;
d_{84} – incomplete verification;
d_{85} – undetected integration errors;
d_{86} – missed edge-cases;
d_{87} – undetected conflicts;
A_9 – refinement and optimization;
d_{91} – recurrence;
d_{92} – compatibility violation;
d_{93} – stability violation;
d_{94} – new errors;
d_{95} – functional degradation;
A_{10} – monitoring and support;
$d_{10.1}$ – insufficient coverage;
$d_{10.2}$ – low performance;
$d_{10.3}$ – important details missed;
$d_{10.4}$ – critical errors not detected;
$d_{10.5}$ – bug recurrence.

The logical-algorithmic model of the development process takes the following form

$$C = B_1 \ldots B_{10} = R_1 \underset{\Omega_1}{\{U_1\}} \ldots R_7 \underset{\Omega_7}{\{U_7\}} \underset{\Omega_8}{\{U_8\}} \ldots \underset{\Omega_{10}}{\{U_{10}\}},$$

where techniques for performing development stages are given in Tables 1, 2.

Table 1

Techniques to perform development stages

Stage	Work	Control	Refinement
A_1	r_{11} – manual architecture formation	ω_{11} – architecture analysis	u_{11} – correction without full revision
	r_{12} – using template architecture	ω_{12} – validation of architectural solutions	u_{12} – architecture refactoring
A_2	r_{21} – using SDK (Software Development Kit) to access media	ω_{21} – file import verification	u_{21} – correction without full analysis
	r_{22} – manual file processing	ω_{22} – format testing	u_{22} – fixes after deployment
A_3	r_{31} – data normalization	ω_{31} – checking the correctness of processing	u_{31} – algorithm correction
	r_{32} – content filtering	ω_{32} – edge-case testing	u_{32} – processing optimization
A_4	r_{41} – manual aggregations	ω_{41} – analysis of results	u_{41} – optimization without analysis
	r_{42} – standard algorithms	ω_{42} – edge-case testing	u_{42} – refactoring
A_5	r_{51} – direct access to the database	ω_{51} – transaction verification	u_{51} – fix without tests
	r_{52} – using ORM (Object-Relational Mapping)	ω_{52} – load testing	u_{52} – database optimization
A_6	r_{61} – manual UI (User Interface) development	ω_{61} – usability testing	u_{61} – UI fixes
	r_{62} – use of components	ω_{62} – adaptability testing	u_{62} – UI refactoring
A_7	r_{71} – manual integration	ω_{71} – interaction check	u_{71} – integration fixes
	r_{72} – integration using API (Application Programming Interface)	ω_{72} – testing	u_{72} – optimization

Table 2

Techniques for performing testing and refinement stages

Stage	Control	Refinement
A_8	ω_{81} – automated testing	u_{81} – test generation
	ω_{82} – regression testing	u_{82} – localization and correction of defects
	ω_{83} – manual testing	u_{83} – test coverage optimization
	ω_{84} – static analysis	–
A_9	ω_{91} – check after changes	u_{91} – correction without analysis
	ω_{92} – regression	u_{92} – refactoring
	–	u_{93} – optimization
A_{10}	$\omega_{10.1}$ – manual monitoring	$u_{10.1}$ – manual log analysis
	$\omega_{10.2}$ – post-release analysis	$u_{10.2}$ – correction after feedback

Variants of software implementation of modules:
g_{11} – code generation via template;
g_{12} – manual code writing;
g_{13} – copying code from examples;
g_{14} – development from scratch.
Types of defects at the software implementation stage:
e_1 – violation of system logic;
e_2 – incorrect integration of components;
e_3 – syntax errors;
e_4 – logical errors;
e_5 – incomplete functionality;
e_6 – incompatibility with architecture;
e_7 – unforeseen execution scenarios;
e_8 – compatibility violations;
e_9 – dependence conflicts.

To build a forecasting model, expert assessments of the defectivity of options for performing work $\left(\mu_{r_i g_i}^{0(j)}\right)$, control $\left(\mu_{\omega_i}^{0(j)}\right)$, and refinement $\left(\mu_{u_i g_i}^{0(j)}\right)$ operations (Table 3) were used, which are interpreted as the percentage of problems occurring because of the selected implementation option.

Table 3

Software defect prediction before training a fuzzy model

Stage	Work		Control		Refinement		Prediction	
	R_i	$\mu_{r_i g_i}^{0(j)}$	Ω_i	$\mu_{\omega_i}^{0(j)}$	U_i	$\mu_{u_i g_i}^{0(j)}$	$\mu_{B_i}^1$	Possible defects
A_1	r_{11}	0.07–0.21	ω_{11}	0.05–0.09	u_{11}	0.12–0.18	0.87	d_{11}, d_{14}, d_{15} e_1, e_3, e_4, e_7
	r_{12}	0.12–0.18	ω_{12}	0.01–0.04	u_{12}	0.03–0.09		
A_2	r_{21}	0.09–0.22	ω_{21}	0.06–0.10	u_{21}	0.11–0.17	0.86	d_{21}, d_{23}, d_{24} e_2, e_4, e_7
	r_{22}	0.13–0.19	ω_{22}	0.02–0.05	u_{22}	0.04–0.08		
A_3	r_{31}	0.08–0.20	ω_{31}	0.04–0.08	u_{31}	0.10–0.16	0.89	d_{31}, d_{32}, d_{35} e_4, e_5, e_7
	r_{32}	0.10–0.17	ω_{32}	0.02–0.05	u_{32}	0.03–0.07		
A_4	r_{41}	0.10–0.23	ω_{41}	0.06–0.10	u_{41}	0.12–0.18	0.86	d_{41}, d_{43}, d_{45} e_1, e_2, e_8, e_9
	r_{42}	0.14–0.20	ω_{42}	0.03–0.06	u_{42}	0.05–0.09		
A_5	r_{51}	0.11–0.24	ω_{51}	0.07–0.11	u_{51}	0.13–0.19	0.85	d_{51}, d_{52}, d_{54}, d_{55} e_2, e_6, e_8, e_9
	r_{52}	0.15–0.22	ω_{52}	0.03–0.06	u_{52}	0.05–0.09		
A_6	r_{61}	0.08–0.19	ω_{61}	0.05–0.09	u_{61}	0.10–0.15	0.88	d_{61}, d_{63}, d_{65} e_1, e_4, e_5, e_7
	r_{62}	0.11–0.17	ω_{62}	0.02–0.05	u_{62}	0.03–0.07		
A_7	r_{71}	0.12–0.25	ω_{71}	0.07–0.11	u_{71}	0.14–0.20	0.86	d_{71}, d_{72}, d_{74} e_2, e_6, e_8, e_9
	r_{72}	0.16–0.22	ω_{72}	0.03–0.06	u_{72}	0.05–0.10		
A_8	–	–	ω_{81}	0.02–0.06	u_{81}	0.04–0.09	0.83	d_{81}, d_{82}, d_{85}, d_{86}, d_{87} e_3, e_4, e_7, e_9
	–	–	ω_{82}	0.03–0.07	u_{82}	0.08–0.14		
	–	–	ω_{83}	0.09–0.15	u_{83}	0.05–0.10		
	–	–	ω_{84}	0.01–0.04	–	–		
A_9	–	–	ω_{91}	0.04–0.08	u_{91}	0.10–0.16	0.89	d_{91}, d_{92}, d_{94}, d_{95} e_1, e_4, e_8
	–	–	ω_{92}	0.03–0.07	u_{92}	0.04–0.09		
	–	–	–	–	u_{93}	0.03–0.08		
A_{10}	–	–	$\omega_{10.1}$	0.06–0.10	$u_{10.1}$	0.05–0.09	0.89	$d_{10.1}$, $d_{10.3}$, $d_{10.4}$, $d_{10.5}$, e_4, e_7, e_9
	–	–	$\omega_{10.2}$	0.04–0.08	$u_{10.2}$	0.08–0.13		

The structure of the fuzzy algorithmic model before training takes the form

$$C = r_{11}\underset{\omega_{11}}{\{u_{11}\}}\, r_{22}g_{12}\underset{\omega_{21}}{\{u_{21}\}}\, r_{32}g_{13}\underset{\omega_{31}}{\{u_{31}\}}\, r_{41}g_{12}\underset{\omega_{41}}{\{u_{41}\}}\, r_{51}g_{12}\underset{\omega_{51}}{\{u_{51}\}}$$
$$r_{61}g_{12}\underset{\omega_{61}}{\{u_{61}\}}\, r_{71}g_{12}\underset{\omega_{71}}{\{u_{71}\}}\,\underset{\omega_{83}}{\{u_{82}\}}\,\underset{\omega_{91}}{\{u_{91}\}}\,\underset{\omega_{10.1}}{\{u_{10.1}\}}.$$

The execution options for the work (R_i), control (Ω_i), and refinement (U_i) operators were selected provided that the threshold reliability levels $\mu_{B_i}^1 \geq 0.80$ were ensured (the percentage of correctly completed tasks was not less than 80%). At the same time, the costs remained minimal ($w_{ij} = 1$; $\lambda_i = 1$). For the obtained logical-algorithmic model, the levels of correctness of $\left(\mu_{B_i}^1\right)$ stages were calculated from formula (7). The possibility of defects d_{ij}, $j = 1,\ldots, q_j$ was estimated based on formula (6).

Analysis of the distribution of defects before training reveals that both primary and secondary defects are predicted. Primary defects are caused by errors at specific stages. Secondary defects are caused by the cascading spread of problems between interdependent modules of the software system. This is due to the large number of work execution options, insufficient efficiency of control and refinement operations.

Below are the results of tuning the fuzzy algorithmic model. The initial data set was obtained based on 25 projects for media content aggregation on devices with limited computing resources. The training sample consisted of options for performing work, control, and refinement operations (Tables 1, 2) and fuzzy estimates of correctness (percentage of correctly completed tasks).

As a result of training, defect ranks w_{ij} were obtained, as well as criticality levels of repeated defects λ_i for options for performing work, control, and refinement operations (Table 4).

The structure of the fuzzy algorithmic model after training takes the following form

$$C = r_{12}\underset{\omega_{12}}{\{u_{11}\}}\, r_{21}g_{12}\underset{\omega_{22}}{\{u_{21}\}}\, r_{32}g_{13}\underset{\omega_{32}}{\{u_{31}\}}$$
$$r_{42}g_{12}\underset{\omega_{42}}{\{u_{41}\}}\, r_{51}g_{12}\underset{\omega_{52}}{\{u_{52}\}}\, r_{61}g_{12}\underset{\omega_{62}}{\{u_{61}\}}$$
$$r_{72}g_{12}\underset{\omega_{72}}{\{u_{72}\}}\,\underset{\omega_{81}}{\{u_{82}\}}\,\underset{\omega_{92}}{\{u_{91}\}}\,\underset{\omega_{10.2}}{\{u_{10.1}\}}.$$

The execution options for the work (R_i), control (Ω_i), and refinement (U_i) operators were selected on the condition of ensuring minimal defects at limited costs. The costs were estimated according to the levels of defect criticality ($w_{ij} \leq 5$; $\lambda_i \leq 2$).

Training was carried out by transferring reliable elements of completed projects into the logical-algorithmic model of the current project (Table 5). After training, changes mainly affected control (stages A_1–A_8) and partly the techniques of performing work (A_2, A_4, A_7) and refinement (A_5, A_7). Stages A_2, A_4, A_5, A_7 are critical because they were predicted to have an increased risk of defects. Due to knowledge distillation, the structure of the logical-algorithmic model was preserved without a complete redesign of the development process, which simplified the process of training the model.

Owing to fuzzy model training, the number of possible defects has significantly decreased. This became possible by eliminating critical errors associated with incomplete requirements, conflicts during module integration, data integrity violations, missed edge cases, and insufficient testing. The possibility of systemic defects that can cause chain failures or gradual degradation of system performance has decreased. Residual risk defects are due to the complexity of the subject area and the uncertainty of operating conditions. Such defects include single logical errors, unforeseen system operation scenarios, and integration problems that are difficult to predict at the design stage. After optimization, they lost their critical cascading nature, are effectively localized, and are removed at the control and refinement stages.

Table 4

Forecasting software defects after training a fuzzy model

Stage	Work		Control		Refinement			Prediction	
	R_i	w_{ij}	Ω_i	w_{ij}	U_i	w_{ij}	λ_i	$\mu^1_{B_i}$	Possible defects
A_1	r_{11}	3–5	ω_{11}	4–6	u_{11}	4–5	1	0.998	d_{11}, e_1, e_3
	r_{12}	4–5	ω_{12}	3–5	u_{12}	3–4			
A_2	r_{21}	4–5	ω_{21}	4–6	u_{21}	3–4	2	0.997	d_{21}, d_{23} e_2, e_4
	r_{22}	3–5	ω_{22}	3–4	u_{22}	3–4			
A_3	r_{31}	4–5	ω_{31}	4–5	u_{31}	3–4	1	0.995	d_{31}, d_{35} e_4, e_7
	r_{32}	4–5	ω_{32}	3–4	u_{32}	3–4			
A_4	r_{41}	4–5	ω_{41}	4–6	u_{41}	4–5	2	0.996	d_{41}, d_{43} e_1, e_2
	r_{42}	3–5	ω_{42}	3–4	u_{42}	3–5			
A_5	r_{51}	4–5	ω_{51}	4–7	u_{51}	4–5	2	0.997	d_{51}, d_{54} e_2, e_9
	r_{52}	4–5	ω_{52}	4–5	u_{52}	4–5			
A_6	r_{61}	4–5	ω_{61}	4–6	u_{61}	4–5	1	0.998	d_{61}, d_{63} e_1, e_4
	r_{62}	4–5	ω_{62}	3–4	u_{62}	3–4			
A_7	r_{71}	4–5	ω_{71}	4–5	u_{71}	5–7	2	0.999	d_{71}, d_{74} e_2, e_8
	r_{72}	3–5	ω_{72}	3–4	u_{72}	4–5			
A_8	–	–	ω_{81}	3–4	u_{81}	3–5	1	0.998	d_{81}, d_{85} e_4, e_7
	–	–	ω_{82}	3–5	u_{82}	4–5			
	–	–	ω_{83}	5–7	u_{83}	3–5			
	–	–	ω_{84}	3–4	–	–			
A_9	–	–	ω_{91}	4–5	u_{91}	4–5	1	0.997	d_{91}, d_{94} e_1, e_4
	–	–	ω_{92}	3–5	u_{92}	3–5			
	–	–	–	–	u_{93}	3–5			
A_{10}	–	–	$\omega_{10.1}$	4–5	$u_{10.1}$	3–5	1	0.995	$d_{10.1}$, $d_{10.4}$ e_4, e_7
	–	–	$\omega_{10.2}$	3–5	$u_{10.2}$	3–5			

Table 5

Knowledge transfer as a result of cross-project learning

Stage	Before training	After training	Knowledge transfer
A_1	r_{11} ω_{11} u_{11}	r_{12} ω_{12} u_{11}	Transition to template architecture and strengthened control of architectural solutions through validation
A_2	r_{22} g_{12} ω_{21} u_{21}	r_{21} g_{12} ω_{22} u_{21}	Manual file processing replaced by SDK and strengthened control of formats
A_3	r_{32} g_{13} ω_{31} u_{31}	r_{32} g_{13} ω_{32} u_{31}	Edge-case testing added
A_4	r_{41} g_{12} ω_{41} u_{41}	r_{42} g_{12} ω_{42} u_{41}	Transition to standard aggregation algorithms and increased control
A_5	r_{51} g_{12} ω_{51} u_{51}	r_{51} g_{12} ω_{52} u_{52}	Added load testing and database optimization
A_6	r_{61} g_{12} ω_{61} u_{61}	r_{61} g_{12} ω_{62} u_{61}	Enhanced control through adaptivity testing
A_7	r_{71} g_{12} ω_{71} u_{71}	r_{72} g_{12} ω_{72} u_{72}	Transition from manual to API integration and increased control and refinement
A_8	ω_{83} u_{82}	ω_{81} u_{82}	Manual testing replaced by automated
A_9	ω_{91} u_{91}	ω_{92} u_{91}	Post-change control replaced by regression control
A_{10}	$\omega_{10.1}$ $u_{10.1}$	$\omega_{10.2}$ $u_{10.1}$	Monitoring moved from manual to post-release analysis

At each stage of development, a comparison of predicted defects with actual ones was carried out (Table 6). The analysis revealed how accurate the forecast was and what measures were taken to prevent the cascading spread of defects.

Table 6

Comparison of predicted and actual defects at the stages of software development

Stage	Predicted defects	Actual defects	Qualitative validation result
A_1	d_{11} e_1, e_3	d_{11} e_1, e_3	The forecast was confirmed. Defects were localized at an early stage
A_2	d_{21}, d_{23} e_2, e_4	d_{21}, d_{23} e_2, e_4	The forecast was confirmed. Timely control prevented the cascading spread of defects
A_3	d_{31}, d_{35} e_4, e_7	d_{31}, d_{35} e_4	The forecast was partially confirmed. Defect e_7 did not manifest itself after control operations
A_4	d_{41}, d_{43} e_1, e_2	d_{41}, d_{43} e_2	The forecast was partially confirmed. Defect e_1 was not confirmed as actual
A_5	d_{51}, d_{54} e_2, e_9	d_{54} e_9	The forecast was partially confirmed. Control operations made it possible to prevent the appearance of significant errors in time. Defects d_{51} and e_2 did not appear after the finishing operations
A_6	d_{61}, d_{63} e_1, e_4	d_{61}, d_{63} e_4	The prediction was partially confirmed. The defects were local in nature and were eliminated without affecting the main functionality. The defect e_1 did not actually manifest itself
A_7	d_{71}, d_{74} e_2, e_8	d_{71}, d_{74} e_2, e_8	The prediction was confirmed. The defects detected were localized within the integration stage
A_8	d_{81}, d_{85} e_4, e_7	d_{81}, d_{85} e_4, e_7, e_8	The prediction was partially confirmed. The model detected the main defects of the testing stage; however, an unforeseen defect e_8 was recorded
A_9	d_{91}, d_{94} e_1, e_4	d_{91}, d_{94} e_4	The forecast was partially confirmed. Defect e_1 did not actually manifest itself
A_{10}	$d_{10.1}$, $d_{10.4}$ e_4, e_7	$d_{10.1}$, $d_{10.4}$ e_4, e_7	The forecast was confirmed. The residual defects did not have a critical impact on the system's performance

Based on the results from comparing the predicted and actually detected defects, an error matrix was obtained (Table 7). The error matrix takes into account two levels of defects. The first level is formed by 50 types of defects d_{ij} for the outputs of A_1–A_{10} stages. The second level is formed by types of defects of the software implementation e_1–e_9, which are subject to validation at each A_1–A_{10} stage. Therefore, the number of checks for defects of e_k type is $9 \cdot 10 = 90$, and the total number of cases for constructing the error matrix is $50 + 90 = 140$.

Table 7

Software defect prediction error matrix

Number of defects		Forecast	
		Positive (with defect)	Negative (no defect)
Fact	Positive (with defect)	TP = 33	FN = 1
	Negative (without defect)	FP = 6	TN = 100

The values of prediction quality metrics are: *Accuracy* = 0.950; *Recall* = 0.971; *Precision* = 0.846; *F-score* = 0.904. A high *Recall* value indicates that the model captured the vast majority of actually detected defects. The lower *Preci-*

sion value is explained by the fact that some of the predicted defects did not actually appear after performing control and refinement operations.

6. Discussion of results based on investigating the effectiveness of the cross-project transfer learning method

The accuracy and interpretability of the software defect prediction model is ensured by building and configuring a fuzzy rule base (5). Fuzzy rules assess the correctness (defectiveness) of the software depending on the execution options of the development stages. The fuzzy knowledge base (5) corresponds to a system of fuzzy logical equations (6) to (8), which determines the correctness (defectiveness) of work, control, and refinement operators. The accuracy of the model is ensured by training fuzzy rules on cross-project data "execution technique–percentage of correctly completed tasks". The genetic algorithm transfers reliable elements (implementation options) of completed projects into the logical-algorithmic model of the current project (Fig. 1). The defectiveness functions of work, control, and refinement operators are adaptively learned using a neural-fuzzy algorithmic model (Fig. 2). Defect ranks determine the allocation of resources. As a result of training, the model predicts a significant reduction in the number of defects due to the ranking of options for performing work, control, and refinement operations (Tables 3, 4).

A feature of our study is the prediction of software defects based on a fuzzy process-oriented model. The model integrates the distribution of resources depending on the ranks of defects associated with the execution options of work, control, and refinement operators. Unlike papers [21, 22], which use code metrics as features, the proposed model applies the functions of correctness (defectivity) of development stages, which ensures the stability of features. Unlike [23, 24], which use measures of project similarity, the introduction of indicators of the methods of implementation of work, control, and refinement operators into the fuzzy model allowed us to solve the issue of heterogeneity of cross-project data. Unlike work [25], where feedback is modeled by recurrent neural-fuzzy networks, the growth of the reliability function is ensured by embedding algorithmic structures "work-control–refinement" into the fuzzy model. Unlike [26], which uses an expert meta-model to train an ensemble of classifiers, our model implements cross-project transfer learning. In this case, good practices from the original projects are transferred to the current project. This simplifies the learning process by consistently defining the model structure and parameters for each development stage.

The complexity of the tuning process is reduced by discretizing the search area. At each stage A_i, the search area is formed in the form of defect estimates for the variants of performing work, control, and refinement operations. Unlike [7–11], where an ensemble of 3–5 basic deep classifiers analyzes at least 12–14 code metrics, the proposed method uses $2q_i$ fuzzy defect estimates for q_i types of defects and their ranks. Thus, for the task of predicting defects in a mobile application, at each of $n = 10$ stages, an optimization problem with 5–12 variables for the techniques of implementing work, control, and refinement operations is solved.

The scope of practical application is predicting the quality of new software based on the experience of completed projects. The justification of decisions is carried out on the basis of a logical-algorithmic model of the development process. The condition for application is the presence of expert or empirical estimates of the correctness (defectivity) of the variants of implementing work, control, and refinement operations. Using this method at the design stages makes it possible to reduce the number of refinement cycles after quality control. The costs of testing associated with correcting errors at the final stages are reduced. At the same time, the risks of cascading defects are minimized, which ensures an increase in the reliability function of the software.

The application of the method is limited to discrete algorithmic processes. The devised method is based on a sequential algorithmic model, which involves moving to the next stage upon completion of the previous stage.

The disadvantage is that the logical-algorithmic model does not take into account the relationships between the stages of development. In this case, the system of fuzzy logical equations should reflect the dependences of the tasks.

Future advancement of this method involves developing a fuzzy cognitive map for modeling the dependences of tasks where the concepts represent the built-in "work – control – refinement" structures.

7. Conclusions

1. A fuzzy algorithmic model for predicting software defects has been constructed, depending on the implementation options for development stages. Unlike machine learning models, the interpretability of the model is ensured by assessing the correctness (defectiveness) of the software system using fuzzy rules. A fuzzy model of the development process is built on the basis of "work-control-refinement" algorithmic structures. The system of fuzzy logic equations connects the fuzzy estimates of the correctness (defectiveness) of work, control, and refinement operators. The integration of indicators into the model of options for the implementation of work, control, and refinement operators has made it possible to solve the issue of heterogeneity in cross-project data. The parameters of the model are ranks (levels of criticality) of defects, which simulate the distribution of resources.

2. A method for tuning a fuzzy algorithmic model of software defect prediction on cross-project data has been devised. Unlike conventional methods for training neural-fuzzy models, training is implemented by transferring reliable elements of completed projects into a logical-algorithmic model of the current project. The training sample consists of estimates of the correctness of the implementation options for work, control, and refinement operators. The tuning process is simplified by using a combined genetic-neural approach. The genetic algorithm performs tuning of the structure of the logical-algorithmic model. The neural-fuzzy algorithmic model provides tuning of the parameters of work, control, and refinement operators as new cross-project data is received.

3. To test the method's performance, a defect prediction was carried out for a media content aggregation software system. As a result of training on cross-project data, a logical-algorithmic model of the software system development process was generated. Unlike machine learning models, this model not only predicts defects but also provides an explanation of which implementation options pose an increased

risk of defects. Owing to the devised method, it was possible to prevent the cascading spread of defects. The remaining errors are effectively detected and corrected at the control and refinement stages, which will allow the release of software with zero defects. Using the method at design stages will reduce the costs of control and refinement. The number of refinement cycles after quality control is reduced. The testing costs associated with eliminating defects at the final stages are reduced.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Funding

The study was conducted without financial support.

Data availaility

The manuscript has associated data in the data warehouse https://github.com/StigUK/software-reliability-training-dataset.

Use of artificial intelligence

The authors declare limited use of artificial intelligence tools during manuscript preparation.

The Gemini language model (LLM) version 3 was used exclusively as an auxiliary tool for visualizing author data in the form of structured tables in Section 5 in accordance with the rules of the editorial policy.

All scientific results were obtained by the authors independently, fragments using AI were checked and edited by the authors.

The authors bear full responsibility for the final manuscript.

Generative AI tools are not indicated as authors and are not responsible for the results.

Declaration submitted by: Prus Bohdan.

Acknowledgments

The manuscript was prepared within the framework of project 0115U001119 “Technologies for building intelligent analog-digital systems for monitoring and analyzing multimedia information”.

Authors’ contributions

Hanna Rakytyanska: Methodology, Formal analysis; **Bohdan Prus:** Software, Data curation, Visualization.

References

1. Malhotra, R., Chawla, S., Sharma, A. (2023). Software defect prediction using hybrid techniques: a systematic literature review. Soft Computing, 27 (12), 8255–8288. https://doi.org/10.1007/s00500-022-07738-w
2. Pal, S., Sillitti, A. (2022). Cross-Project Defect Prediction: A Literature Review. IEEE Access, 10, 118697–118717. https://doi.org/10.1109/access.2022.3221184
3. Rotshtein, O. P., Shtovba, S. D., Kozachko, O. M. (2007). Modeliuvannia ta optymizatsiya nadiynosti bahatovymirnykh alhorytmichnykh protsesiv. Vinnytsia: «UNIVERSUM», 211. Available at: http://ir.lib.vntu.edu.ua/handle/123456789/2296
4. Rotshtein, A. P., Rakytyanska, H. B. (2012). Fuzzy Evidence in Identification, Forecasting and Diagnosis. Studies in Fuzziness and Soft Computing. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-25786-5
5. Rakytyanska, H., Prus, B. (2025). Fuzzy-algorithmic analysis of software reliability. Information Technology and Computer Engineering, 22, 136–147. https://doi.org/10.31649/vitce/3.2025.136
6. Rakytyanska, H. (2023). Knowledge Distillation in Granular Fuzzy Models by Solving Fuzzy Relation Equations. Advancements in Knowledge Distillation: Towards New Horizons of Intelligent Systems, 95–133. https://doi.org/10.1007/978-3-031-32095-8_4
7. Singh, M., Chhabra, J. K. (2024). Machine learning based improved cross-project software defect prediction using new structural features in object oriented software. Applied Soft Computing, 165, 112082. https://doi.org/10.1016/j.asoc.2024.112082
8. Abbas, S., Aftab, S., Adnan Khan, M., M. Ghazal, T., Al Hamadi, H., Yeob Yeun, C. (2023). Data and Ensemble Machine Learning Fusion Based Intelligent Software Defect Prediction System. Computers, Materials & Continua, 75 (3), 6083–6100. https://doi.org/10.32604/cmc.2023.037933
9. Sharma, T., Jatain, A., Bhaskar, S., Pabreja, K. (2023). Ensemble Machine Learning Paradigms in Software Defect Prediction. Procedia Computer Science, 218, 199–209. https://doi.org/10.1016/j.procs.2023.01.002
10. Dong, X., Wang, J., Liang, Y. (2025). A Novel Ensemble Classifier Selection Method for Software Defect Prediction. IEEE Access, 13, 25578–25597. https://doi.org/10.1109/access.2025.3537658
11. Omer, A., Rathore, S. S., Kumar, S. (2024). ME-SFP: A Mixture-of-Experts-Based Approach for Software Fault Prediction. IEEE Transactions on Reliability, 73 (1), 710–725. https://doi.org/10.1109/tr.2023.3295012
12. Wang, S., Huang, L., Gao, A., Ge, J., Zhang, T., Feng, H. et al. (2023). Machine/Deep Learning for Software Engineering: A Systematic Literature Review. IEEE Transactions on Software Engineering, 49 (3), 1188–1231. https://doi.org/10.1109/tse.2022.3173346
13. Goyal, S. R. (2026). Effective software defect prediction with deep neural networks. Results in Engineering, 29, 108378. https://doi.org/10.1016/j.rineng.2025.108378

14. Sun, T., Allix, K., Kim, K., Zhou, X., Kim, D., Lo, D. et al. (2023). DexBERT: Effective, Task-Agnostic and Fine-Grained Representation Learning of Android Bytecode. IEEE Transactions on Software Engineering, 49 (10), 4691–4706. https://doi.org/10.1109/tse.2023.3310874
15. Villoth, J. P., Zivkovic, M., Zivkovic, T., Abdel-salam, M., Hammad, M., Jovanovic, L. et al. (2025). Two-tier deep and machine learning approach optimized by adaptive multi-population firefly algorithm for software defects prediction. Neurocomputing, 630, 129695. https://doi.org/10.1016/j.neucom.2025.129695
16. Abdu, A., Zhai, Z., Abdo, H. A., Algabri, R. (2024). Software Defect Prediction Based on Deep Representation Learning of Source Code From Contextual Syntax and Semantic Graph. IEEE Transactions on Reliability, 73 (2), 820–834. https://doi.org/10.1109/tr.2024.3354965
17. Bal, P. R., Kumar, S. (2025). An Approach for Cross Project Defect Prediction Using Identical Metrics Matching and Deep Neural Network. IEEE Transactions on Reliability, 74 (2), 2678–2692. https://doi.org/10.1109/tr.2024.3435709
18. Omondiagbe, O. P., Licorish, S. A., MacDonell, S.G. (2024). Improving transfer learning for software cross-project defect prediction. Applied Intelligence, 54, 5593–5616. https://doi.org/10.1007/s10489-024-05459-1
19. Nevendra, M., Singh, P. (2025). TRGNet: a deep transfer learning approach for software defect prediction. Expert Systems with Applications, 282, 127799. https://doi.org/10.1016/j.eswa.2025.127799
20. Tong, H., Zhang, D., Liu, J., Xing, W., Lu, L., Lu, W., Wu, Y. (2024). MASTER: Multi-Source Transfer Weighted Ensemble Learning for Multiple Sources Cross-Project Defect Prediction. IEEE Transactions on Software Engineering, 50 (5), 1281–1305. https://doi.org/10.1109/tse.2024.3381235
21. Elsabagh, M. A., Emam, O. E., Gafar, M. G., Medhat, T. (2023). Handling uncertainty issue in software defect prediction utilizing a hybrid of ANFIS and turbulent flow of water optimization algorithm. Neural Computing and Applications, 36 (9), 4583–4602. https://doi.org/10.1007/s00521-023-09315-0
22. Lakshmi, P. J., Krishna, T. V. S., Kumar, N. B., Rao, D. S. N. M., Hosseinpour, A., Lenin, N. C. (2025). Improving Software Fault Prediction With a Hybrid DE-WOA Optimizer and ANFIS-Enhanced Ensemble Learning. IEEE Access, 13, 158194–158209. https://doi.org/10.1109/access.2025.3603980
23. Azzeh, M., Elsheikh, Y., Alqasrawi, Y. (2024). Software defect density prediction using grey system theory and fuzzy logic. Soft Computing, 28 (21-22), 12897–12916. https://doi.org/10.1007/s00500-024-10324-x
24. Azzeh, M., Abdel-Rahman, M. J. (2026). Cross-project software defects prediction using fuzzy embedding and deep learning. Information and Software Technology, 190, 107968. https://doi.org/10.1016/j.infsof.2025.107968
25. Behera, A. K., Panda, M., Dehuri, S. (2024). A recurrent ANFIS tuned by modified differential evolution for efficient prediction of software reliability. Evolutionary Intelligence, 17 (5-6), 3469–3482. https://doi.org/10.1007/s12065-024-00940-9
26. Chatterjee, S., Saha, D. (2024). IT2F-SEDNN: an interval type-2 fuzzy logic-based stacked ensemble deep learning approach for early phase software dependability analysis. Innovations in Systems and Software Engineering, 21 (2), 727–746. https://doi.org/10.1007/s11334-024-00563-4
27. Rakytyanska, H. (2015). Fuzzy classification knowledge base construction based on trend rules and inverse inference. Eastern-European Journal of Enterprise Technologies, 1 (3 (73)), 25–32. https://doi.org/10.15587/1729-4061.2015.36934
28. Rakytyanska, H. (2017). Optimization of fuzzy classification knowledge bases using improving transformations. Eastern-European Journal of Enterprise Technologies, 5 (2 (89)), 33–41. https://doi.org/10.15587/1729-4061.2017.110261
29. Rakytyanska, H., Prus, B. (2024). Constructing Prototype-Based Granular Fuzzy Rules for Scene Classification on Mobile Devices. Vol. 1. Lecture Notes in Data Engineering, Computational Intelligence, and Decision-Making, 194–218. https://doi.org/10.1007/978-3-031-70959-3_10