UDC 004.453:519.876:519.2 **doi:** https://doi.org/10.20998/2522-9052.2026.1.10

Sergii Liubarskyi[1], Alina Yanko[2], Yurii Zdorenko[2], Bakhtiyar Khudayarov[3]

[1] Kruty Heroes Military Institute of Telecommunications and Information Technology, Kyiv, Ukraine
[2] National University "Yuri Kondratyuk Poltava Polytechnic", Poltava, Ukraine
[3] Tashkent Institute of Irrigation and Agricultural Mechanization Engineers" National Research University, Tashkent, Uzbekistan

AN ADAPTIVE MODEL FOR SOFTWARE CODE QUALITY ASSESSMENT IN REFACTORING TASKS BASED ON FUZZY LOGIC

Abstract. The article's objective is to develop a hybrid adaptive model for assessing software code quality based on code smell characteristics by combining fuzzy logic and machine learning methods to enhance the objectivity and efficiency of refactoring. **The methodology** underlying this research is aimed at developing a hybrid adaptive model for software code quality assessment. It combines fuzzy logic and artificial intelligence methods, specifically an adaptive neuro-fuzzy inference system (ANFIS). The multi-layered ANFIS implements the Takagi-Sugeno fuzzy inference with the ability to learn using gradient methods. The methodology is based on a hybrid approach that integrates expert knowledge with the automated training of the model on real data. Results. The research resulted in the development of a hybrid adaptive model for software code quality assessment based on fuzzy logic and the ANFIS. This model allows for automated, objective, and flexible code quality assessment in refactoring tasks. The model uses eight key code smell metrics: WMC, DIT, RFC, LCOM, NOA, NOC, CBO, and FANOUT. Their normalization and processing are performed using fuzzy logic based on the Takagi-Sugeno algorithm. This ensures that the uncertainty and subjectivity of expert evaluations are taken into account. The ANFIS architecture allows the model to learn from real data, with subsequent automated adjustment of the membership function parameters and rule weights. This enables the model to adapt to various technology stacks and projects. The use of trapezoidal membership functions increases the accuracy of modeling critical code smell zones, while the hybrid learning algorithm based on gradient descent ensures high precision in determining code quality, ultimately contributing to improved software efficiency, maintainability, scalability, and security. **The scientific novelty** of the research lies in the development of a hybrid adaptive model for software code quality assessment. Unlike existing models, this one is based on fuzzy logic and an ANFIS, which combines expert knowledge with automated training on real data to enhance the objectivity and efficiency of the refactoring process. The proposed ANFIS architecture with trapezoidal membership functions is used to process eight key code smell metrics (WMC, DIT, RFC, LCOM, NOA, NOC, CBO, FANOUT) within the context of Takagi-Sugeno fuzzy inference. This provides a flexible, interpretable, and adaptive assessment of code quality with the ability to automatically tune model parameters based on gradient learning, which significantly increases the accuracy of code quality determination and the model's suitability for various technology stacks and projects. **The practical significance** of the research lies in the direct implementability and integration of the developed hybrid adaptive model for software code quality assessment into existing static analysis tools and DevOps processes, specifically as plugins for Continuous Integration/Continuous Delivery (CI/CD) systems. This will enable automated, objective, and adaptive monitoring of code quality in real time. In addition, the model has significant potential for extension to various programming languages and technology stacks by analyzing large datasets from open-source repositories, which will enhance its universality and accuracy. A promising direction for future work is to improve the ANFIS architecture by incorporating deep learning methods, which would allow for the automatic detection of new code smells and their interdependencies. The development of interpretable mechanisms to explain the model's decisions will increase developer trust in the system and promote its widespread adoption in both industrial software development and educational processes in software engineering and cybersecurity.

Keywords: refactoring; code smells; fuzzy logic; ANFIS; software code quality; software cybersecurity; artificial intelligence; Takagi-Sugeno fuzzy inference.

Introduction

Problem relevance. The rapid development of information technologies and their global use in various fields demand higher requirements for ensuring the high quality of software.

One way to improve software quality is through the use of comprehensive methods [1]. In today's environment, methods for formalized assessment of software product reliability and functional compliance are becoming particularly relevant, as they allow for the minimization of risks associated with errors in mission-critical applications [2], while also considering that diagnostic errors can directly affect system security [3]. Furthermore, a comprehensive methodology for software quality management (SQM) involves the application of mathematical modeling, system analysis, and automated testing technologies [4].

To effectively solve the tasks of designing, developing, and maintaining information systems, it is crucial to integrate software code quality assurance mechanisms at all stages of the software development lifecycle.

Undoubtedly, one of the most promising directions in addressing this challenge is the application of the software code refactoring process.

Software code refactoring is a process aimed at making changes to existing code to improve its structure, readability, and maintainability without altering its external behavior.

The use of refactoring within the framework of code optimization can prevent potential attacks on information systems [5] and reduce their information security risks. This action is particularly relevant in the context of growing cyber threats and digital risks to economic security [6, 7].

Existing approaches to software code refactoring are characterized by certain limitations, including:

– Subjectivity: The refactoring process depends on the developer's knowledge, experience, and personal views, which can lead to incomplete or redundant code modifications. Each developer may define their own formal criteria for assessing the need for this process.

– Labor-intensiveness: Since manual refactoring requires significant time and human resources, the automation of this process is inevitable.

– Insufficient scalability: Most existing refactoring methods are effective only for small systems or individual modules. Extending these methods to large systems leads to the complexity of analyzing interconnections between components.

– Lack of adaptability: Traditional refactoring techniques have a fixed set of rules and patterns that do not account for the specifics of a particular project.

– Lack of deep contextual understanding: Refactoring systems often fail to consider business logic, architectural constraints, and code semantics. This can result in changes that are syntactically correct but semantically incorrect.

The aforementioned limitations can, to varying degrees, affect the task of ensuring software code quality [8–10]. The need for a formalized approach to managing the refactoring process is evident, as similar methods for structuring management tasks are successfully applied in other complex technical systems, such as telecommunication networks [11]. One promising direction for code improvement is to account for secondary software defects that may arise during refactoring [12]. This problem can be solved by using predictive systems. General approaches for their implementation are presented in [13]. However, models that allow for the consideration of such defects are still in the development stage and require improvement. In particular, neurocomputers that operate based on efficient machine arithmetic can be used to solve similar problems [14].

The application of artificial intelligence (AI) methods to refactoring tasks can significantly help overcome these limitations by providing automated, objective, and adaptive solutions to improve code quality, security, and maintainability. There are numerous examples [15, 16] where AI methods have been effectively applied to synthesize (generate) new code, analyze existing code, detect vulnerabilities, programming errors, style violations, architectural flaws, automate code improvement, and obtain secure and optimized code based on identified shortcomings. It is also worth noting that similar approaches to error control based on mathematical procedures have found applications in other areas, such as systems with modular arithmetic [17].

The model MovePerf proposed in [18] focuses on predicting a program's execution time after a specific type of refactoring called "Move method". The authors investigate how moving methods impacts performance using a hybrid deep learning model that accounts for feature interaction. However, this process does not include a comprehensive procedure for evaluating the overall quality of the source code. This procedure is crucial for a thorough check to show how effectively the code's internal organization has been improved without losing functionality. It is an important part of supporting the long-term viability of the software. To address this challenge, fuzzy logic models can be applied, which are successfully used, for instance, in assessing risks within information security management [19].

Therefore, a fuzzy logic-based model for software code quality assessment is proposed. Fuzzy logic serves as an effective mathematical tool for solving decision-making problems [20], particularly under complex conditions of uncertainty and incomplete input data. In the proposed model, software code quality is evaluated based on a set of features (discrepancy characteristics) used as input. For example, these inputs can be code smell characteristics [21], such as: the degree of code duplication, method size, class complexity, number of variables, comments, and the number of known vulnerabilities. The use of fuzzy logic for code quality analysis allows for the effective combination of code smells and quality metrics into a unified evaluation system. This approach provides a reasoned, flexible, and interpretable assessment that can be adapted to specific projects and technologies.

Literature review. An analysis of publications [22–24] on fuzzy logic-based models for software code quality assessment shows that this approach effectively handles the ambiguity and uncertainty present in code quality metrics. Unlike sharp threshold values or binary rules, fuzzy logic considers the degree of membership to sets, which better reflects real-world software development practices. Most scientific studies indicate that the following key features, presented in Table 1, are used for the fuzzy assessment of software code quality.

Table 1 – **Code smell characteristics for fuzzy software code quality assessment**

Feature	Description
Degree of code duplication	Measures the proportion of identical or similar code in different parts of a software system. A high level of duplication increases code interpretability and maintenance complexity.
Method size	The number of lines of code in a method. Very long methods are an anti-pattern that complicates code reading and testing.
Complexity of user or base library classes	Metrics that include cyclomatic complexity, the number of methods/fields, and the depth of the inheritance tree.
Number of variables	Many variables in a method can indicate a suboptimal structure and reduced code readability.
Number of comments	The ratio of comments to code. An insufficient number of comments often indicates poor code documentation.
Known vulnerabilities	Security issues found through static analysis tools (e.g., SonarQube), such as SQL injection, XSS, etc.

These features form the input space for a fuzzy system, where each characteristic has a membership function that determines the degree of "negative consequence" or "positive influence" on the software code.

The application of these features using fuzzy logic for code analysis involves the following steps:

1. Fuzzification. Each feature is converted into a fuzzy variable using membership functions [23]. For example:

– for lines of code: Low (0–30), Medium (20–60), High (>50);

– for code duplication: None (<5%), Moderate (5–20%), High (>20%).

2. Fuzzy inference rules. A set of rules is formed based on expert knowledge, such as:

IF (Code Duplication IS High) AND (Method Length IS Long) THEN (Code Quality IS Poor)

Such rules can be defined by experts or automatically generated and trained on data [24].

3. Fuzzy Inference. The Mamdani or Sugeno algorithm is applied to compute the result – the overall code quality assessment.

4. Defuzzification. The resulting fuzzy score is converted into a numerical value, for example, from 0 to 10, where 0 is the lowest code quality and 10 is the highest.

Eliminating these code smell characteristics from the software code is a task for subsequent refactoring. However, the precise definition of fuzzy categories like "too little," "moderate," "sufficient," or "too much" is quite subjective. Therefore, the mathematical apparatus of fuzzy logic [25, 20] can be used to formalize the membership of code smell characteristics to these fuzzy subsets, and a fuzzy production rule base can describe the relationship between code smell features and the output characteristic – software code quality.

The use of fuzzy system-based models for software code quality assessment is proposed in [21, 26, 27]. However, they are based on the Mamdani fuzzy inference algorithm, which does not allow for automated adjustment of membership function parameters or the generation of production rules. Therefore, working with them requires manual, expert-driven parameter definition, which can lead to incorrect results.

To automate the tasks of parameter tuning and adaptation, established approaches based on a combination of artificial neural networks and fuzzy logic can be used. It is proposed that the values of selected code smell characteristics be used as input parameters. The output parameter, the current software code quality value, will be determined through the fuzzy inference process. This fuzzy system is proposed to be trained using a hybrid algorithm based on backpropagation and gradient descent. Training data can be collected by analyzing code from open-source repositories (e.g., GitHub). The training of the created model is proposed to be conducted in the Matlab environment [28]. Thus, the objective of this research is to develop a hybrid adaptive model for software code quality assessment based on code smell discrepancy characteristics by combining fuzzy logic and machine learning methods to enhance the objectivity and efficiency of refactoring.

1 The developed hybrid adaptive model

Architecturally, it is proposed to use a hybrid model based on the adaptive neuro-fuzzy inference system (ANFIS) platform. ANFIS is a multi-layered neural network that performs Takagi-Sugeno fuzzy inference [27] with the ability to learn using gradient methods.

ANFIS represents an adaptive neuro-fuzzy inference system. The use of such systems has proven effective for solving various problems. For example, in [29], such a system is used for DDoS attack detection. On one hand, ANFIS is a neural network with a single output and multiple inputs, which represent fuzzy linguistic variables. The terms of the input linguistic variables are described by standard membership functions, while the terms of the output variable are represented by linear or constant functions. On the other hand, ANFIS is a fuzzy inference system in which each of the fuzzy production rules has a constant weight equal to 1.

The model's architecture structurally involves the implementation of the following layers:

1. Input layer (Layer 1): This layer handles the representation of normalized code smell values. Code smells are indicators of poor structure or design in software code that suggest the need for refactoring. For the model, we propose using quantitative characteristics that can be measured automatically, namely:

– WMC (Weighted Methods per Class);

– DIT (Depth of Inheritance Tree);

– RFC (Response for a Class), the number of unique methods a class can invoke;

– LCOM (Lack of Cohesion in Methods), the measure of a class's methods not being interconnected;

– NOA (Number of Attributes);

– NOC (Number of Children);

– CBO (Coupling Between Objects);

– FANOUT, the number of outgoing dependencies of the function/method.

Each of these metrics is normalized (Fig. 1) to the range [0, 1], where 0 is the ideal value (no code smell) and 1 is a critical value (a strong code smell).

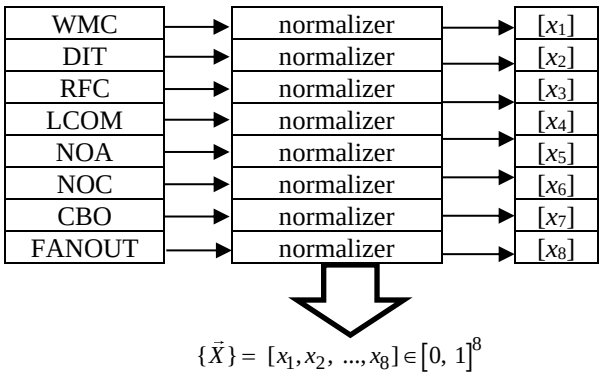

Fig. 1. Normalization process of the model's input layer

To bring the values of the code features to a single scale, it is proposed to apply linear normalization:

$$x_i^{norm} = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}, \quad (1)$$

where x_i is the initial value of the i-th code smell; $x_{\min}$, $x_{\max}$ are the minimum and maximum expected values (extrema) for this metric.

The aforementioned extremum values can be: theoretical (for example, for DIT: [0, ∞], but in practice

[0, 10]); statistical (obtained from a dataset); or expert-defined.

Thus, after normalization, an input vector $\{\vec{X}\}$ is formed, where each x_i is the normalized value of the corresponding code smell. This vector is fed into the ANFIS system, where each component is used for fuzzification (conversion into fuzzy sets), which then activates specific fuzzy rules and, ultimately, influences the code quality assessment.

2. Fuzzification layer (Layer 2). The fuzzification layer of the hybrid ANFIS model is the second layer in its architecture and is responsible for converting crisp values into fuzzy values, which is the process of fuzzification. This is a key step that allows the system to operate with fuzzy sets, which are used to implement Takagi-Sugeno fuzzy inference [27].

For each input x_i and each fuzzy term A_j the fuzzification layer calculates:

$$O_{ij} = \mu_{A_j}(x_i), \quad (2)$$

where O_{ij} – the degree of membership of the i-th input to the j-th term; $\mu_{A_j}(x_i)$ – the value of the membership function for x_i.

The conversion of crisp values into fuzzy ones is performed using membership functions of various types (e.g., triangular, trapezoidal).

The fuzzification procedure involves the process of converting a numerical value (for example, a normalized code smell value) into a degree of membership to fuzzy sets such as: low, medium, or high. This process shows how much the numerical code smell value corresponds to the given linguistic terms.

Each input feature (e.g., WMC, CBO) has its own group of membership functions that determine how much the value corresponds to each fuzzy term.

For example, for WMC, the fuzzy terms "Low", "Medium", or "High" can be defined. Thus, if WMC = 0.7, the system calculates: $\mu_{Low}(0.7)$; $\mu_{Medium}(0.7)$; $\mu_{High}(0.7)$, where $\mu(x)$ is the corresponding membership function. In the ANFIS fuzzification layer (Layer 2), parameterized membership functions are most commonly used, for example:

– triangular membership function $\mu(x)$:

$$\mu(x;a,b,c) = \begin{cases} 0, & x \le a, \\ (x-a)/(b-a), & a < x \le b, \\ (c-x)/(c-b), & b < x \le c, \\ 0, & x > c, \end{cases} \quad (3)$$

where x is the input value of the code smell characteristic; a is the left boundary of the fuzzy term (degree of membership $\mu = 0$); b is the center of the term (degree of membership $\mu = 1$); c is the right boundary of the term (degree of membership $\mu = 0$).

The parameters a, b, c determine the shape of the triangular function, which reflects the degree to which the value of the input feature (e.g., WMC, CBO) corresponds to a specific linguistic term (Low, Medium, High). They allow for the clear definition of the boundaries of fuzzy sets for each feature:

– trapezoidal membership function $\mu(x)$:

$$\mu(x;a,b,c,d) = \begin{cases} 0, & x \le a \\ (x-a)/(b-a), & a < x \le b \\ 1, & b < x \le c, \\ (d-x)/(d-c), & c < x \le d \\ 0 & x > d \end{cases} \quad (4)$$

where x is the input value of the code smell characteristic; a is the start of the rising slope of the membership function; b is the start of the full membership plateau ($\mu = 1$); c is the end of the full membership plateau; d is the end of the falling slope of the membership function.

The trapezoidal membership function $\mu(x)$ differs from the triangular one by having a plateau of full membership ($\mu = 1$) between b and c. The trapezoidal function is also useful for modeling clearer zones of high impact (e.g., "the class has a critical code smell") and allows for more flexible description of linguistic terms in cases where it is necessary to clearly define a high-risk range.

– sigmoid membership function $\mu(x)$:

$$\mu(x;f,c) = \frac{1}{1+e^{-a(x-c)}}, \quad (5)$$

where x is the input value (e.g., a normalized code smell value);

a – is the slope coefficient that determines the steepness of the function (the larger $|a|$, the sharper the transition from 0 to 1);

c – is the shift parameter that defines the point of intersection with 0.5, i.e., the value of x at which the degree of membership equals 0.5.

This membership function can be useful in software code quality assessment tasks, especially when it is necessary to model the monotonically increasing or decreasing influence of a particular code smell on the overall code quality score [30].

In ANFIS, the parameters of these functions (e.g., a, b, c, d) are learnable and are optimized using gradient descent or another learning algorithm.

For assessing software code quality based on code smell characteristics within the hybrid ANFIS model, the choice of the membership function has a significant impact on the accuracy and interpretability of the fuzzy inference.

The advantages and disadvantages of the presented membership functions, in accordance with the model's objective, are provided in Table 2.

Although other types of membership functions exist (e.g., Gaussian, S-shaped, Z-shaped), their use is not recommended for this task due to: difficulty of interpretation (they are less understandable to experts, making it harder to explain their correlation with code smells); increased complexity (they complicate the learning process and increase the risk of overfitting due to a larger number of parameters). Lack of a clear connection to linguistic terms [31].

Table 2 – **Advantages and disadvantages of membership functions**

Type of function	Advantages	Disadvantages
Triangular	Simplicity, clarity, fast training, interpretability.	Lack of a clear high-risk zone.
Trapezoidal	Presence of a full membership zone ($\mu = 1$) high flexibility, well-suited for modeling critical zones.	More complex interpretation, more parameters.
Sigmoid	Smooth change in the degree of membership, analytical derivative, well-suited for gradient learning.	Less interpretable, more difficult for experts, risk of overfitting.

The trapezoidal membership function is considered optimal for implementing the hybrid adaptive model for software code quality assessment because it:

– best suited for modeling critical code smell zones, where the feature's value (e.g., WMC, CBO) reaches a dangerous level. The trapezoidal function allows for the definition of a plateau of full membership ($\mu = 1$);

– has clear boundaries and is easily explained by experts, unlike the sigmoid function. This is crucial since the model combines expert knowledge and machine learning;

– offers flexible configuration due to its four parameters. This ensures better adaptability to different code smells and projects. In the hybrid ANFIS model, these parameters (*a*, *b*, *c*, *d*) are optimized by gradient descent, which provides high prediction accuracy;

– is suitable for modeling linguistic terms.

It is proposed to choose 3-5 terms for each code smell characteristic based on Table 3.

Table 3 – **Selection of linguistic terms for code smell characteristics**

Terms	Details
3 terms	*Low – Medium – High* (Most commonly used in practice)
4 terms	*Very Low – Low – Medium – High*
5 terms	*Very Low – Low – Medium – High – Very High* (Used when a detailed analysis is required)

The parameter tuning for the membership functions should be performed using alternative methods:

– based on expert knowledge. Experts indicate which code smell values correspond to "Low", "Medium", or "High" levels;

– based on data (statistically). Quantiles (e.g., 25%, 50%, 75%) are calculated for each metric. These values are used as the intersection points of the membership functions.

3. The rule layer (Layer 3). This layer calculates the firing strength of each rule, which is the product of the membership function values for all sub-conditions of the rule.

The fuzzy rule layer in the hybrid ANFIS model is a main element that implements the Takagi-Sugeno fuzzy inference. This layer provides the model with semantic interpretability, as each rule can be understood by a human expert, and it also supports parameter learning using gradient methods.

The main task of this layer is to compute the firing strength of each rule based on the degrees of membership of the inputs to the fuzzy sets.

Each rule has the form:

$$\begin{gathered}\text{“}If\ x_1 \in A_1,\ x_2 \in A_2,\ \ldots\ x_n \in A_n,\\ than\ y = p_1x_1 + p_2x_2 + \ \ldots\ + p_nx_n + q\text{”},\end{gathered} \tag{6}$$

where $A_1,\ldots,A_n$ are the fuzzy terms corresponding to the linguistic variables (e.g., Low, Medium, High); $x_1, x_2, \ldots, x_n$ are the input code smell features (e.g., WMC, CBO, LCOM, DIT, NOC, RFC, CBO, MFA, etc.); *y* is the output value (code quality assessment based on code smell characteristics); $p_1, p_2, \ldots, p_n$ are the weighting coefficients of the input variables (influence weights). These parameters determine the influence of each input feature (e.g., a specific code smell) on the output value (code quality assessment) within a particular rule. If p_i has a large magnitude, it means that the corresponding feature x_i significantly affects the output in this rule. If p_i is close to zero, the influence of this feature is minimal; *q* is the free term (bias). This is a constant value that is added to the sum of the weighted input features. It acts as a baseline or offset for a specific rule (*q* can reflect a systematic contribution of the rule that is independent of the input data). This representation is typical for a first-order Takagi-Sugeno inference algorithm rule [32].

So, if we have the following input conditions:

– number of code smell characteristics: 8 (WMC, DIT, RFC, LCOM, NOA, NOC, CBO, FANOUT);

– number of terms: 4 ("Low", "Medium", "High", "Critical") corresponding to the trapezoidal membership function;

– number of Takagi-Sugeno rules will be $4^8 = 65536$ (a full combination).

In general, Takagi-Sugeno rules have the following form:

$$\begin{gathered}\text{“}If\ \left(WMC\ is\ x_1\right) and \left(DIT\ is\ x_2\right) and\ \ldots\\ and \left(FANOUT\ is\ x_8\right),\ than\ Quality = p_1 * WMC + \\ + p_2 * DIT + \ \ldots + p_8 * FANOUT + q\text{”}.\end{gathered} \tag{7}$$

A fragment of the rule base range is presented in Table 4. The operation algorithm of this layer assumes that the degrees of membership from the fuzzification layer for each input and each term are passed to its input. For each rule, the logical "AND" (conjunction) of the degrees of membership of all its antecedents is computed:

$$\omega_i = \mu A_1(x_1) \cdot \mu A_2(x_2) \cdot \ldots \cdot \mu A_n(x_n), \tag{8}$$

where ω_i is the degree of activation (firing strength) of the *i*-th fuzzy rule (6); $\mu A_j(x_j)$ is the degree of membership of the *j*-th input x_j to the fuzzy term A_j; *n* is the number of input features (code smells), for example: WMC, CBO, LCOM, etc.

Table 4 – **A fragment of the Takagi-Sugeno rules**

Rule #	Conditions (8 features)	Influence weights $p_1, p_2, \dots, p_8$	Bias q	Inference formula (1st-order Takagi-Sugeno)
1	*WMC=Low, DIT=Low, RFC=Low, LCOM=Low, NOA=Low, NOC=Low, CBO=Low, FANOUT=Low*	Initially, they are initialized with small random values ([0, 1]) or are set by an expert. The $p_i \in [0,1]$ values are normalized, but they can also be negative (if an increase in a *code smell* improves quality, although this is rare).	$q \approx 0.05..0.5$ depending on the rules	$y = p_1*WMC + p_2*DIT + p_3*RFC +$ $+ p_4*\text{LCOM} + p_5*NOA + p_6*NOC+$ $+ p_7*CBO + p_8*FANOUT + q$
2	*WMC=Medium, DIT=Low, RFC=Low, LCOM=Low, NOA=Low, NOC=Low, CBO=Low, FANOUT=Low*			$y = p_1*WMC + p_2*DIT + p_3*RFC +$ $+ p_4*\text{LCOM} + p_5*NOA + p_6*NOC$ $+$ $+ p_7*CBO + p_8*FANOUT + q$
…	…			…
65536	*WMC=Critical, DIT=Critical, RFC=Critical, LCOM=Critical, NOA=Critical, NOC=Critical, CBO=Critical, FANOUT=Critical*			$y = p_1*WMC + p_2*DIT + p_3*RFC +$ $+ p_4*\text{LCOM} + p_5*NOA + p_6*NOC$ $+$ $+ p_7*CBO + p_8*FANOUT + q$

The firing strength (activation) is a number that shows how strongly a particular rule applies to the current set of code smells [33].

Therefore, the operation algorithm of the rule layer can be described mathematically as follows:

Let there be given:

- $x = [x_1, x_2, \dots, x_n]$ is the normalized set of code smell parameters;
- for each x_i, M fuzzy terms are defined;
- the number of possible rules is $R = M^n$.

Then, for each rule $r = \overline{1,R}$, the following holds true:

$$\omega_r = \prod_{i=1}^{n} \mu_{A_i^r}(x_i), \quad (9)$$

where $\mu_{A_i^r}(x_i)$ is the degree of membership of the *i*-th input to the *j*-th term within the *r*-th rule.

The output of the rule layer is a vector $\vec{\omega} = [\omega_1, \omega_2, \dots, \omega_R]$, where each element corresponds to the activation of a separate rule. The layer has high interpretability due to the clear semantics of the rules, supports the learning of inference parameters (using gradient descent), and is the result of combining fuzzy logic and neural networks.

4. Normalization layer (Layer 4). This layer ensures the calculation of the normalized firing strength of each rule. The Normalization Layer of the hybrid ANFIS model is a key element for the fuzzy inference stage, which provides the relative weighting of rule activations before they are used in defuzzification. This layer allows the system to react more stably to different input combinations and improves the model's training for optimizing its parameters using gradient methods.

This layer is aimed at calculating the normalized firing strengths of the rules. Its task is to determine how important each rule is relative to the others for the current set of inputs. This is due to the fact that: the absolute values of the rule activations ω_i can be large or small simultaneously; for an accurate output calculation, it is necessary to consider the relative weight of each rule.

Based on the vector of rule activations $\vec{\omega} = [\omega_1, \omega_2, \dots, \omega_R]$, the layer calculates as:

$$\bar{\omega}_i = \omega_i \Big/ \sum_{i=1}^{R} \omega_i, \quad (10)$$

where ω_i is the degree of activation of the *i*-th rule; *R* is the total number of rules; $\bar{\omega}_i$ is the normalized *i*-th rule activation.

This approach guarantees that the sum of all normalized activations equals 1. The Normalization Layer is a part of the architecture that ensures learning stability and increases the accuracy of fuzzy inference. Normalizing the rule activations prevents rules with a large number of inputs from having an undue advantage. This is especially important in a hybrid learning algorithm, where both the inference parameters (of the linear rule part) and the membership function parameters are optimized in a single pass (when using gradient descent).

Therefore, at the output of the layer, each rule has a weight that will be used in defuzzification.

5. Defuzzification layer (Layer 5). This layer calculates the weighted average of the results, taking into account the rules' activity. The defuzzification layer of the hybrid ANFIS model is designed for the final computational stage of fuzzy inference. It provides the conversion of fuzzy results into a crisp numerical value, which can be interpreted as a software code quality assessment, or any other quantitative indicator.

In this layer, the final result of the fuzzy inference of the proposed system is calculated using the weighted average of the rule conclusions, taking into account their normalized activations. This layer uses the results from the rule layer (Layer 3) and the normalization layer (Layer 4) and calculates the resulting parameter – the code quality assessment, *Q*. At the same time, it supports gradient learning, and the result of the fuzzy inference depends on the model's parameters.

Mathematically, this process can be described as follows. The input data are:

- the normalized rule activations $\bar{\omega}_i$:

$$\vec{\bar{\omega}} = [\bar{\omega}_1, \bar{\omega}_2, \dots, \bar{\omega}_R];$$

– the Takagi-Sugeno rule conclusions:

$$f_i = p_1 x_1 + p_2 x_2 + ... + p_n x_n + q_i,$$
$$\vec{f} = [f_1, f_2, \ ... \ , f_R].$$

The final stage of fuzzy inference (software code quality assessment) is calculated using the formula:

$$Q = \sum_{i=1}^{R} \bar{\omega}_i \cdot f, \qquad (11)$$

where Q is the final software code quality assessment ($Q \in [0, 1]$, where 0 corresponds to high code quality and 1 to low code quality); R is the total number of rules; $\bar{\omega}_i$ is the normalized i-th rule activation; f_i is the output value of the i-th rule (a linear combination of the inputs).

The defuzzification layer (Layer 5) in ANFIS implements the conversion of fuzzy results into a crisp number, using a weighted average that accounts for rule activations. It also supports gradient learning and is a key link in the model optimization process.

6. Output layer (Layer 6). The Output Layer returns the software code quality assessment in the range [0, 1]. The quality assessment is formed in the previous layer based on fuzzy inference. At this step, it is possible to perform post-processing, such as normalization, rounding, conversion to a discrete category (e.g., "Low" quality ($Q \in [0.7-1]$, refactoring is needed), "Medium" quality ($Q \in [0.3-0.7]$, refactoring is possible), "High" quality (($Q \in [0-0.3]$, no need for refactoring)), or comparison with a threshold for decision-making.

The architecture of the adaptive model for software code refactoring based on fuzzy logic is shown in Fig. 2.

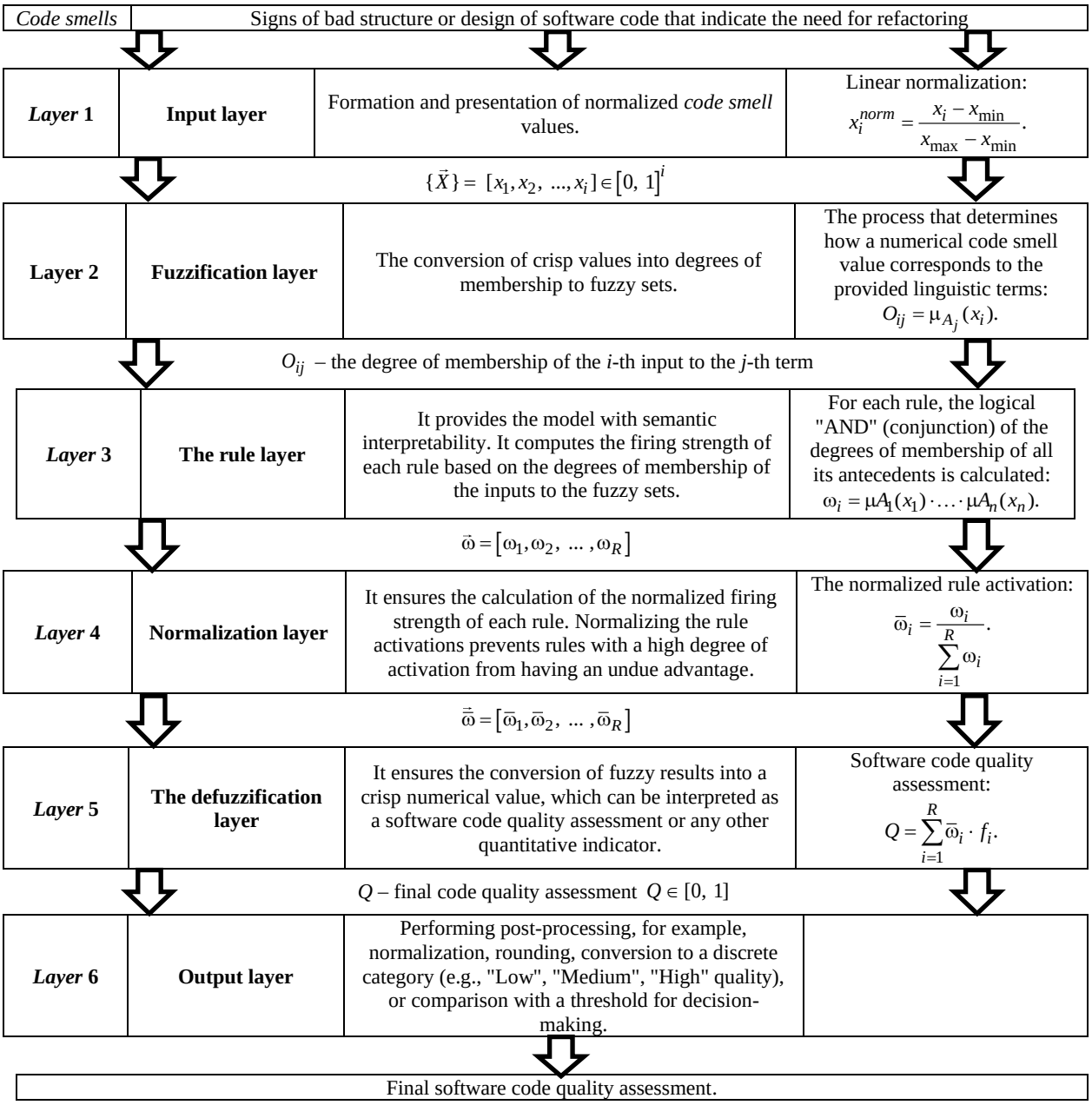

Fig. 2. Architecture of an adaptive model for software code refactoring based on fuzzy logic

2 Hybrid learning algorithm

The hybrid learning algorithm of the adaptive model for software code refactoring based on fuzzy logic combines forward pass and backward pass procedures, as well as the gradient descent algorithm.

The training involves the following stages:

1. Initialization of fuzzy rules. Initial rules are created based on expert knowledge or analysis of the data from the subject area of the research.

2. Forward pass procedure. This is the implementation of the forward computational process through all the layers of the ANFIS model. Data passes through all layers to the Defuzzification Layer, and the expected quality assessment $Q_{expected}$ is calculated using formula (11).

3. Calculation of the training error E. This stage involves comparing the predicted quality value $Q_{expected}$ with the actual value Q_{actual} (expert evaluation or prior refactoring) using the formula:

$$E = \left(Q_{actual} - Q_{expected}\right)^2 / 2, \quad (12)$$

where Q_{actual} is the actual, real value of software code quality that is known at the time of model training. It is the known value of the output parameter that the model is trying to predict. It can be determined either based on an expert evaluation ([0, 1], where 0 is ideal code, and 1 is code that requires refactoring) or based on an automatic assessment (using static analysis tools); $Q_{expected}$ is the expected code quality value that the model returns after processing the code smells. The corresponding value is calculated through the Takagi-Sugeno fuzzy inference algorithm, depends on the rule activations, their conclusions, and the model's parameters, and is formed through defuzzification in ANFIS.

4. Backward pass. This procedure is aimed at minimizing the error in determining (predicting) the software code quality by adapting the model's parameters. That is, the model adjusts its internal parameters to improve the accuracy of the $Q_{expected}$ determination, making it closer to the actual value Q_{actual}.

In the hybrid ANFIS model, two categories of parameters are adapted:

– membership function parameters (*a*, *b*, *c* in triangular functions or *a*, *b*, *c*, *d* in trapezoidal functions) are updated using gradient descent, as they influence the nonlinear part of the model;

– rule weighting coefficients $(p_1, p_2, \ldots, p_n)$, which correspond to the linear conclusion of the Takagi-Sugeno rule, are updated using the least squares method.

This hybrid approach ensures a significant increase in the efficiency and speed of training.

The backward pass procedure allows the error from the output of the adaptive model for software code refactoring to be directed backward toward the model's inputs. This is done to evaluate how each model parameter influenced the error and, accordingly, to adjust these parameters.

The backward pass procedure involves:

– performing the forward pass procedure;

– calculating the expected code quality value $Q_{expected}$ and the training error *E*;

– calculating the gradients. This very process allows for determining how much each model parameter influences the prediction error, and accordingly, adjusting these parameters to improve accuracy.

The gradient is a vector of partial derivatives with respect to all model parameters:

$$\nabla E = \left[\frac{\partial E}{\partial \theta_1}, \frac{\partial E}{\partial \theta_2}, \ldots, \frac{\partial E}{\partial \theta_n}\right], \quad (13)$$

where ∇E is the gradient vector (of partial derivatives). It allows for determining how the parameters should be changed to reduce the error;

∂E is the partial derivative of the error with respect to each individual model parameter θ_i. It demonstrates the influence of a single parameter on the model's error;

$\theta_1, \theta_2, \ldots, \theta_n$ are the model parameters (for example, fuzzy rule coefficients, membership function parameters, neural network weights).

Each derivative is calculated using the chain rule of differentiation, taking into account the influence of each parameter through all previous layers (fuzzification and defuzzification).

5. Execution of the parameter update procedure using the gradient descent algorithm. The gradient descent algorithm is aimed at optimizing parameters to minimize functions (MSE, MAE, etc.) [34].

Parameters are updated using the formula:

$$\theta_{new} = \theta_{old} - \eta \frac{\partial E}{\partial \theta}, \quad (14)$$

where θ is the parameter (for example, b in a triangular function); η is the learning rate; $\frac{\partial E}{\partial \theta}$ is the error gradient (the derivative of the error with respect to this parameter).

The process is repeated over a set of cycles until the error reaches a predefined level for model convergence. With a large number of inputs and rules, the training process can be lengthy. Poor or contradictory data can lead to inaccurate predictions.

However, the undeniable advantages of the presented model are: the ability to improve rule settings based on new data; a proper level of interpretability due to the fuzzy rules; an organic combination of expert knowledge and machine learning; the ability to work with various technologies and languages; and support for integration into the software development process.

3 Discussion of results

The developed hybrid ANFIS model demonstrates significant potential for the automated and objective assessment of software code quality. Unlike existing approaches, which are often based on static rules or subjective expert evaluations, our model combines the interpretability of fuzzy logic with the adaptability of machine learning. The use of eight key code smell metrics as input parameters allows for covering a wide

range of potential problems in code. The proposed architecture, with its trapezoidal membership functions and a hybrid learning algorithm, ensures not only high accuracy but also flexibility, which is critically important for adapting the model to various programming languages and technology stacks. This allows the model to independently optimize its parameters based on real data, which minimizes the reliance on manual tuning.

The achieved accuracy in code quality determination indicates the effectiveness of applying this hybrid approach to solve complex problems that have uncertainty and fuzziness in their input data. The model can be used not only for detecting existing code smells but also for predicting possible secondary defects that may arise during refactoring. Implementing this model into development processes, particularly into CI/CD tools, will provide developers with valuable, objective, and fast feedback on code quality, which will ultimately increase the overall efficiency of development, maintainability, and security of software.

Conclusions

Based on the results of the conducted research, the following conclusions were formulated.

A hybrid adaptive model for software code quality assessment was developed based on fuzzy logic and the ANFIS architecture, which allows for overcoming the limitations of classical refactoring approaches, particularly their subjectivity, labor-intensity, and lack of adaptability.

The proposed model integrates eight key code smell metrics (WMC, DIT, RFC, LCOM, NOA, NOC, CBO, FANOUT) using Takagi-Sugeno fuzzy inference. This ensures a flexible, interpretable, and objective assessment of code quality.

The choice of trapezoidal membership functions was justified as optimal for modeling critical code smell zones. The hybrid learning algorithm, which combines forward and backward propagation of error, ensures the automatic tuning of the model's parameters, significantly increasing assessment accuracy.

The obtained results provide a solid foundation for further scientific and practical developments. The prospects for future research lie in the practical implementation and integration of the developed hybrid adaptive model into existing static analysis tools and DevOps processes. Specifically, it can be implemented as plugins for continuous integration and continuous delivery (CI/CD) systems, which would provide automated, objective, and adaptive real-time code quality monitoring.

Furthermore, future work should focus on improving and expanding the model. It is worth separately investigating its potential application for different programming languages and technology stacks by collecting and analyzing large datasets from open repositories. It is also promising to enhance the ANFIS architecture by introducing deep learning methods or hybridizing it with other artificial intelligence approaches. This would allow for the automatic detection of new code smells and their interrelationships. Additionally, developing interpretable mechanisms that would explain the model's decisions will significantly increase developers' trust in the system and facilitate its widespread adoption in industrial software development and educational processes for software engineering and cybersecurity.

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, as well as the results reported in this paper.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

REFERENCES

1. Laktionov, A. (2021), “Improving the methods for determining the index of quality of subsystem element interaction”, *Eastern-European Journal of Enterprise Technologies*, vol. 6, no. 3 (114), pp. 72–82, doi: https://doi.org/10.15587/1729-4061.2021.244929
2. Buriak, A., and Maslii, O. (2025), “Minimization of digital risks and threats to the economic security of the state through the use of generative artificial intelligence”, *Eastern-European Journal of Enterprise Technologies*, vol. 4, no. 13 (136), pp. 17–25, doi: https://doi.org/10.15587/1729-4061.2025.336640
3. Ponochovniy, Y., Bulba, E., Yanko, A. and Hozbenko, E. (2018), “Influence of diagnostics errors on safety: Indicators and requirements”, *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies* (DESSERT), 24-27 May 2018, Kyiv, Ukraine, pp. 53–57, doi: https://doi.org/10.1109/DESSERT.2018.8409098
4. Onyshchenko, S., Zhyvylo, Ye., Hlushko, A., and Bilko, S. (2024), “Cyber risk management technology to strengthen the information security of the national economy”, *Naukovyi Visnyk Natsionalnoho Hirnychoho Universytetu*, vol. 5, pp. 136–142, doi: https://doi.org/10.33271/nvngu/2024-5/136 7
5. Krasnobayev, V., Yanko, A., Hlushko, A., Kruk, O., Gakh, V., Onyshchenko, S., Maslii, O., Kivshyk, O., Potapova, K., Nalyvaichuk, M., Meliukh, V., Gurynenko, S., Ostapenko, T., and Hrashchenko I. (2023), *Economic And Cyber Security*, Monographs, PC TECHNOLOGY CENTER, doi: https://doi.org/10.15587/978-617-7319-98-5
6. Kudinova, A., Maslii, O., Smokvina, V., and Tsyhanenko, K. (2025), “The impact of digitalization on the financial institutions' economic security in the face of growing cyber threats”, *Financial and Credit Activity Problems of Theory and Practice*, vol. 4(63), pp. 466–483, doi: https://doi.org/10.55643/fcaptp.4.63.2025.4790
7. Maslii, O., Buriak, A., Chaikina, A., and Cherviak, A. (2025), “Improving conceptual approaches to ensuring state economic security under conditions of digitalization”, *Eastern-European Journal of Enterprise Technologies*, vol. 1, no. 13 (133), pp. 35–45, doi: https://doi.org/10.15587/1729-4061.2025.319256
8. Tsantalis, N., Angelopoulos, T., Herraiz, I., Mazinanian D. and Dig D. (2018), “Accurate and efficient refactoring detection in version histories”, *Proceedings of the 40th International Conference on Software Engineering* (ICSE), pp. 939–950, doi: https://doi.org/10.1145/3180155.3180206

9. Amandeep, K., Sushma, J., Shivani, G., and Gaurav, D. (2021), "A Review on Machine-learning Based Code Smell Detection Techniques in Object-oriented Software System(s)", *Recent Advances in Electrical & Electronic Engineering*, vol. 14, is. 3, pp. 290–303, doi: https://doi.org/10.2174/2352096513999200922125839
10. Manju, B.P.K. (2022), "A Survey of Static and Dynamic Metrics Tools for Object Oriented Environment", *Lecture Notes in Electrical Engineering*, vol. 790 LNEE, pp. 521–530, doi: https://doi.org/10.1007/978-981-16-1342-5_40
11. Samokhvalov, Y., Bovda, E., and Liubarskyi, S. (2024), "Structuring management tasks in the telecommunication network management system", *11th International Scientific Conference "Information Technology and Implementation"* (IT&I-2024), 20–21 November 2024, Kyiv, Ukraine, CEUR 3909, pp. 375–386, available at: https://ceur-ws.org/Vol-3909/Paper_30.pdf
12. Rudenko, O., Yanko, A., Haitan, O., Zdorenko, Y., and Rudenko, Z. (2025), "Secondary Software Faults Detection Models", *Lecture Notes in Networks and Systems*, vol. 1391 LNNS, pp. 212–221, doi: https://doi.org/10.1007/978-3-031-90735-7_17
13. Romanenkov, Y., Danova, M., Kashcheyeva, V., Bugaienko, O., Volk, M., Karminska-Bielobrova, M., and Lobach, O. (2018), "Complexification Methods of Interval Forecast Estimates in the Problems on Short-Term Prediction", *Eastern-European Journal of Enterprise Technologies*, vol. 3, no. 3 (93), pp. 50–58, doi: https://doi.org/10.15587/1729-4061.2018.131939
14. Yanko, A., Krasnobayev, V., Hlushko, A., and Goncharenko, S. (2025), "Neurocomputer operating in the residue class system", *Advanced Information Systems*, vol. 9, no. 2, pp. 84–92, doi: https://doi.org/10.20998/2522-9052.2025.2.11
15. Nanadani, H., Saad, M., and Sharma, T. (2023), "Calibrating Deep Learning-Based Code Smell Detection Using Human Feedback", *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation* (SCAM), 2–3 October 2023, Bogotá, Colombia, pp. 37–48, doi: https://doi.org/10.1109/SCAM59687.2023.00015
16. Ayadi, M., Rhazali, Y., and Lahmer, M. (2022), "A Proposed Methodology to Automate the software manufacturing through Artificial Intelligence (AI)", *Procedia Computer Science*, vol. 201, pp. 627–631, doi: https://doi.org/10.1016/j.procs.2022.03.082
17. Krasnobayev, V., Kuznetsov, A., Yanko, A., and Kuznetsova, T. (2020), "The data errors control in the modular number system based on the nullification procedure", *3rd Int. Workshop on Computer Modeling and Intelligent Systems* (CMIS-2020), 27 April – 1 May 2020, Zaporizhzhia, Ukraine, CEUR 2608, pp. 580–593, doi: https://doi.org/10.32782/cmis/2608-45
18. Yu, Y., Lu, Y., Liang, S., Zhang, X., Zhang, L., Bai, Y., and Zhang, Y. (2025), "Predicting a Program's Execution Time After Move Method Refactoring Based on Deep Learning and Feature Interaction", *Applied Sciences*, vol. 15(8), art. no. 4270, doi: https://doi.org/10.3390/app15084270
19. Zdorenko, Y., Yanko, A., Myziura, M., and Fesokha, N. (2025), "Development of a fuzzy risk assessment model for information security management", *Techn. Audit and Production Reserves*, vol. 4(84), doi: https://doi.org/10.15587/2706-5448.2025.334954
20. Levashenko, V., Liashenko, O., and Kuchuk, H. (2020), "Building Decision Support Systems based on Fuzzy Data", Advanced Information Systems, vol. 4, no. 4, pp. 48–56, doi: https://doi.org/10.20998/2522-9052.2020.4.07
21. Sehgal, R., Mehrotra, D., and Bala, M. (2018), "Prioritizing the refactoring need for critical component using combined approach", *Decision Science Letters*, vol. 7, pp. 257–272, available at: https://pdfs.semanticscholar.org/17c5/3ab8eba7114de5ce9ba595c26590f4b99835.pdf
22. Kara, M., Lamouchi, O., and Ramdane-Cherif, A. (2016), "Ontology Software Quality Model for Fuzzy Logic Evaluation Approach", Procedia Computer Science, vol. 83, pp. 637–641, doi: https://doi.org/10.1016/j.procs.2016.04.143
23. Ritu and Sangwan O.P. (2021), "Software Quality Prediction Method Using Fuzzy Logic", *Turkish Journal of Computer and Mathematics Education* (TURCOMAT), vol. 12(11), pp. 807–817, doi: https://doi.org/10.17762/turcomat.v12i11.5966
24. Chuan Y.R., Huang, T., Towey, D. and Zhou, L. (2025), "A Median-Based Fuzzy Approach to Software Quality Evaluation", Tsinghua science and technology, vol. 30, no. 5, pp. 2146−2168, doi: https://doi.org/10.26599/TST.2024.9010103
25. Qi, R., Tao, G. and Jiang, B. (2019), *Fuzzy system identification and adaptive control* (1st ed.), Springer Cham, doi: https://doi.org/10.1007/978-3-030-19882-4
26. Maddeh, M., Al-Otaibi, S., Alyahya, S., Hajjej, F. and Ayouni, S. (2023), "A Comprehensive MCDM-Based Approach for Object-Oriented Metrics Selection Problems", *Applied Sciences*, vol. 13(6), pp. 3411, doi: https://doi.org/10.3390/app13063411
27. Golosovskiy, M.S., Bogomolov, A.V., and Evtushenko, E.V. (2021), "An Algorithm for Setting Sugeno-Type Fuzzy Inference Systems", *Automatic Documentation and Mathematical Linguistics*, vol. 55, pp. 79–88, doi: https://doi.org/10.3103/S000510552103002X
28. (2024), *Fuzzy Logic Toolbox. Design and simulate fuzzy logic systems*, available at: https://www.mathworks.com/help/fuzzy/index.html
29. Onyshchenko, S., Haitan, O., Yanko, A., Zdorenko, Y., and Rudenko, O. (2024), "Method for detection of the modified DDoS cyber attacks on a web resource of an Information and Telecommunication Network based on the use of intelligent systems", *6th International Workshop on Modern Data Science Technologies Workshop* (MoDaST 2024), Lviv, Ukraine, 31 May – 1 June 2024, CEUR 3723, pp. 219–235, available at: https://ceur-ws.org/Vol-3723/paper12.pdf
30. Puja, R. S., Fatema, T., Akhter, N. and Khatun, A. (2023), "Prediction of Code Smell from Source Code: A Hybrid Approach", *2023 Int. Conf. on Information and Communication Technology for Sustainable Development* (ICICT4SD), 21-23 September 2023, Dhaka, Bangladesh, pp. 315–319, doi: https://doi.org/10.1109/ICICT4SD59951.2023.10303449
31. Lima, J. F., Patiño-León, A., Orellana, M., and Zambrano-Martinez, J. L. (2025), "Evaluating the Impact of Membership Functions and Defuzzification Methods in a Fuzzy System: Case of Air Quality Levels", *Applied Sciences*, vol. 15(4), 1934, doi: https://doi.org/10.3390/app15041934
32. Coradini, M. F., Felão, L. H. V., Lyra, S. d. S., Teixeira, M. C. M., and Kitano, C. (2025), "Takagi–Sugeno Fuzzy Nonlinear Control System for Optical Interferometry", Sensors, vol. 25(6), art. no. 1853, doi: https://doi.org/10.3390/s25061853
33. Pecorelli, F., Lujan, S., Lenarduzzi, V., Palomba, F. and de Lucia, A. (2022), "On the adequacy of static analysis warnings with respect to code smell prediction", *Empirical Software Eng.*, vol. 27, art. no. 64, doi: https://doi.org/10.1007/s10664-022-10126-5
34. Dogo, E. M., Afolabi, O. J., Nwulu, N. I., Twala, B., and Aigbavboa, C. O. (2018), "A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks", *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems* (CTEMS), 21-22 December 2018, Belgaum, India, pp. 92–99, doi: https://doi.org/10.1109/CTEMS.2018.8769211

Received (Надійшла) 21.10.2025
Accepted for publication (Прийнята до друку) 14.01.2026

ABOUT THE AUTHORS / ВІДОМОСТІ ПРО АВТОРІВ

Любарський Сергій Володимирович – кандидат технічних наук, доцент, доцент кафедри, Військовий інститут телекомунікацій та інформатизації імені Героїв Крут, Київ, Україна;
Sergii Liubarskyi – Candidate of Technical Sciences, Associate Professor, Associate Professor of the Department, Military Institute of Telecommunications and Informatization Technologies named after Heroes of Kruty; Kyiv, Ukraine;
e-mail: sergii.liubarskyi@viti.edu.ua, ORCID Author ID: https://orcid.org/0000-0001-8068-1106;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=59548693300.

Янко Аліна Сергіївна – кандидат технічних наук, доцент, доцент кафедри комп'ютерних та інформаційних технологій і систем, Національний університет «Полтавська політехніка імені Юрія Кондратюка», Полтава, Україна;
Alina Yanko – Candidate of Technical Sciences, Associate Professor, Associate Professor of the Department of Computer and Information Technologies and Systems, National University "Yuri Kondratyuk Poltava Polytechnic", Poltava, Ukraine;
e-mail: al9_yanko@ukr.net; ORCID Author ID: https://orcid.org/0000-0003-2876-9316;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57094953000.

Здоренко Юрій Миколайович – кандидат технічних наук, доцент, доцент кафедри комп'ютерних та інформаційних технологій і систем, Національний університет «Полтавська політехніка імені Юрія Кондратюка», Полтава, Україна;
Yurii Zdorenko – Candidate of Technical Sciences, Associate Professor, Associate Professor of the Department of Computer and Information Technologies and Systems, National University "Yuri Kondratyuk Poltava Polytechnic", Poltava, Ukraine;
e-mail: zdorenkoviti@gmail.com; ORCID Author ID: https://orcid.org/0000-0002-5649-771X;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=57188762885&origin=resultslist.

Худаяров Бахтіяр Алімович – доктор технічних наук, професор кафедри вищої математики, Національний дослідницький університет «Ташкентський інститут інженерів іригації та механізації сільського господарства», Ташкент, Узбекистан;
Bakhtiyar Khudayarov– Doctor of Technical Sciences, Professor of the Department of Higher Mathematics, "Tashkent Institute of Irrigation and Agricultural Mechanization Engineers" National Research University, Tashkent, Uzbekistan;
e-mail: khudayaroba@gmail.com; ORCID Author ID: https://orcid.org/0000-0002-2876-8447;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=9244185400.

Адаптивна модель оцінки якості програмного коду в задачах рефакторингу на основі fuzzy-логіки

С. В. Любарський, А. С. Янко, Ю. М. Здоренко, Б. А. Худаяров

Анотація. Мета статті полягає у розробці гібридної адаптивної моделі оцінки якості програмного коду на основі характеристик невідповідності code smells шляхом поєднання методів fuzzy-логіки та машинного навчання для підвищення об’єктивності та ефективності рефакторингу. **Методологія**, що покладена в основу дослідження, спрямована на розробку гібридної адаптивної моделі оцінки якості програмного коду, поєднує у собі нечітку логіку (fuzzy logic) та методи штучного інтелекту, зокрема адаптивну систему нейро-нечіткого виведення (ANFIS – Adaptive Neuro-Fuzzy Inference System). Багатошарова адаптивна система нейро-нечіткого виведення ANFIS реалізує нечітке логічне виведення Такагі-Сугено з можливістю навчання за допомогою градієнтних методів. Методологія побудована на гібридному підході, що інтегрує експертні знання з автоматичним навчанням моделі на реальних даних. **Результати.** За результатами проведеного дослідження розроблено гібридну адаптивну модель оцінки якості програмного коду на основі нечіткої логіки та адаптивної нейро-нечіткої системи виведення ANFIS, що дозволяє автоматизовано, об’єктивно та гнучко оцінювати якість програмного коду в задачах рефакторингу. Модель використовує вісім ключових code smells-метрик (WMC, DIT, RFC, LCOM, NOA, NOC, CBO, FANOUT). Їх нормалізація та обробка здійснюється за допомогою нечіткої логіки на основі алгоритму Такагі-Сугено. Це забезпечує врахування невизначеності та суб’єктивності експертних оцінок. Архітектура ANFIS дозволяє моделі навчатися на реальних даних з подальшим автоматичним налаштуванням параметрів функцій приналежності та вагових коефіцієнтів правил. Саме це надає змогу адаптуватися до різних технологічних стеків та проєктів. Використання трапецієподібних функцій приналежності підвищує точність моделювання критичних зон code smells, а гібридний алгоритм навчання на основі градієнтного спуску забезпечує високу точність визначення якості коду, що в підсумку сприяє підвищенню ефективності, підтримуваності, розширюваності та безпеки програмного забезпечення. **Наукова новизна** дослідження полягає у розробці гібридної адаптивної моделі оцінки якості програмного коду, яка на відміну від існуючих, здійснена на основі нечіткої логіки та адаптивної системи нейро-нечіткого виведення ANFIS, що поєднує експертні знання з автоматичним навчанням на реальних даних для підвищення об’єктивності та ефективності процесу рефакторингу. Запропоновано використання архітектури ANFIS з трапецієподібними функціями приналежності для обробки восьми ключових метрик code smells (WMC, DIT, RFC, LCOM, NOA, NOC, CBO, FANOUT) у контексті нечіткого логічного виведення Такагі-Сугено, що забезпечує гнучке, інтерпретоване та адаптивне оцінювання якості коду з можливістю автоматичного налаштування параметрів моделі на основі градієнтного навчання, що значно підвищує точність визначення якості коду та придатність моделі для різноманітних технологічних стеків та проєктів. **Практичне значення** дослідження полягає у можливості прямої реалізації та інтеграції розробленої гібридної адаптивної моделі оцінки якості програмного коду в існуючі інструменти статичного аналізу та DevOps-процеси, зокрема у вигляді плагінів для систем безперервної інтеграції та доставки (CI/CD). Це дозволить забезпечити автоматизований, об'єктивний та адаптивний моніторинг якості коду в реальному часі. Окрім цього, модель має значний потенціал для розширення на різні мови програмування та технологічні стеки шляхом аналізу великих масивів даних з відкритих репозиторіїв, що підвищить її універсальність та точність. Перспективним є також удосконалення архітектури ANFIS через впровадження методів глибокого навчання, що дасть змогу автоматично виявляти нові code smells та їх взаємозв'язки. Розробка інтерпретованих механізмів пояснення рішень моделі підвищить довіру розробників до системи та сприятиме її широкому впровадженню як у промислову розробку програмного забезпечення, так і в навчальні процеси з програмної інженерії та кібербезпеки.

Ключові слова: рефакторинг; code smells; нечітка логіка; ANFIS; якість програмного коду; кібербезпека програмного забезпечення; штучний інтелект; нечітке логічне виведення Такагі-Сугено.