REPUBLIC OF AZERBAIJAN

On the right of the manuscript

ABSTRACT

of the dissertation for the degree of Doctor of Philosophy

DEVELOPMENT OF METHODS AND ALGORITHMS FOR IMPROVING THE RELIABILITY OF SOFTWARE SYSTEMS

Specialty: 1203.01 – Computer sciences

Field of science: Technical sciences

Applicant: **Tamilla Adil Bayramova**

Bakı – 2025

The work was performed at the Institute of Information Technology of the Ministry of Science and Education of the Republic of Azerbaijan

Scientific supervisor:	Doctor of Philosophy in physical and mathematical sciences, Associate Professor **Tofig Hasanaga Kazimov**
Official opponents:	Doctor of Technical sciences, Prof. **Naila Fuad Musayeva**
	Doctor of Technical sciences, Prof. **Bayram Ganimat Ibrahimov**
	Doctor of Technical sciences, Prof. **Valeh Azad Mutsafayev**

Dissertation council ED 1.20 of Supreme Attestation Commission under the President of the Republic of Azerbaijan operating at the Institute of Control Systems of Ministry of Science and Education of the Republic of Azerbaijan

Chairman of the Dissertation Council:	Academician of ANAS, Doctor of Technical Sciences, Professor **Ali Mahammad Abbasov**
Scientific Secretary of Dissertation Council:	Doctor of Philosophy in Engineering, Associate Professor **Tahir Ali Alizada**
Chairman of the Scientific Seminar:	Doctor of Technical Sciences, Associate Professor **Rovshan Agakishi Guliyev**

GENERAL CHARACTERISTICS OF THE WORK

The relevance of the topic and the degree of development. Today, software systems drive innovation, efficiency, and transformation, ensuring economic growth, improving social well-being, and solving global problems. These systems have permeated all areas, from smartphones to business, healthcare, education, government administration, and critical infrastructure, and are applied to solve large-scale scientific, financial, military, and manufacturing issues. A software system failure can lead to significant economic losses, as well as human casualties, environmental disasters, and social unrest.

The use of low-quality program code in the analysis and processing of big data can lead to serious errors in calculations and the adoption of business solutions or other important decisions, causing major problems. The transition to green programming, prompted by climate change, has increased the demand for optimizing algorithms and selecting energy-efficient programming languages and infrastructure. Errors in software systems that cause production shutdowns or customer data leaks create significant financial risks, seriously damage the trust of any company, and tarnish its reputation. The increasing complexity of software has led to a rise in the number of errors and failures, which in turn reduces the effectiveness of debugging and testing methods used to eliminate these problems.

Information and Communication Technologies (ICT), one of the main priorities of Azerbaijan's sustainable development policy, have rapidly integrated into all areas of society's socio-economic system and people's daily lives, becoming an inseparable part of socio-economic relations. The development and implementation of national software products is a long-term strategic goal aimed at strengthening the country's technological independence and ensuring information security. This approach plays a vital role in protecting national interests by reducing dependence on foreign technologies.

The "Strategy of the Republic of Azerbaijan on Information Security and Cybersecurity for 2023–2027," approved by the Decree of the President of the Republic of Azerbaijan dated August 28, 2023,

emphasizes the importance of forming an information security and cybersecurity industry, minimizing and controlling direct dependence on foreign countries in the field of information protection technologies, developing software engineering and the digital economy, and providing organizational and scientific support for the production of information protection tools and the development of the national digital economy, especially software engineering[1].

The role of subjective factors in the development of reliable software systems is also significant. Those who work on software systems must have deep knowledge of programming, software system architecture, algorithms, databases, modern technologies, etc. Experience with various programming languages, tools, and relevant domain expertise is also of great importance.

To succeed in the highly competitive software market, companies that develop software systems must deliver them on time, with high quality, and with guaranteed reliability. The development of high-quality software systems and the improvement of their reliability require methods and algorithms based on artificial intelligence technologies. Therefore, the development of intelligent methods and algorithms that warn about errors in software systems, ensure their prevention, and allow the software system to continue functioning during a failure is one of the important issues to be solved.

Systematic research on the reliability of software systems began in the 1970s. In this field, prominent researchers include B. Boehm, T. DeMarco, T. McCabe, P. Roy, J. Schmidhuber, T. Shooman, M. Khoshgoftaar, H. Pham, Q. Wang, V. Glushkov, V. Lipayev, C. Chatal, and others.

The object and subject of the research. The research object in the dissertation is software systems. The subject of this dissertation is the

[1] Decree of the President of the Republic of Azerbaijan on the approval of the "Strategy of the Republic of Azerbaijan on Information Security and Cybersecurity for 2023–2027." [Electronic resource] / – Approved by Decree No. 4060 of the President of the Republic of Azerbaijan dated August 28, 2023. – Baku, August 28, 2023. URL: https://president.az/az/articles/view/60949

development of methods and algorithms to enhance the reliability of software systems.

Purpose and responsibilities of the work. The aim is to develop new models, methods, and algorithms for detecting and eliminating errors, predicting failures, ensuring fault tolerance, and assessing the reliability of software systems.

To achieve the goals set in the dissertation, the following tasks were defined:

- Analyzing existing methods and algorithms for ensuring the reliability of software systems and identifying scientific-theoretical and technological problems.
- Developing a model and architectural-technological principles for enhancing the reliability of software systems.
- Conducting a comparative analysis of existing methods for calculating software complexity and developing a new hybrid method.
- Analyzing machine learning algorithms and developing a method and algorithm for detecting faulty modules in software systems.
- Analyzing subjective factors affecting the reliability of a software system and developing a method for their elimination.
- Developing the architectural-technological principles and a model for ensuring the fault tolerance of software systems and a method for predicting failures.
- Developing a method and algorithm for evaluating the reliability of software systems based on certain indicators.

Research methods. The research methods employed to address the issues in the dissertation included mathematical statistics, artificial neural networks, machine learning, and deep learning.

The main provisions of the defense:

- A new hybrid method for calculating the complexity of program code;
- A machine learning-based method for detecting faulty modules in software systems;

- A method for eliminating subjective factors that affect the reliability of a software system;
- A conceptual model for ensuring the fault tolerance of software systems and a method for predicting failures;
- A method and algorithm for evaluating the reliability of software systems based on certain indicators.

The scientific novelty of the work. The scientific novelty of the dissertation is as follows:

- A new hybrid method for calculating the complexity of program code was developed;
- An ensemble method based on machine learning for detecting faulty modules in software systems was developed;
- A method was developed to eliminate the subjective factors that affect the reliability of a software system;
- A conceptual model for ensuring the fault tolerance of software systems and a method for predicting failures based on an LSTM recurrent neural network were developed;
- A method and algorithm based on a hybrid LSTM-GRU artificial neural network were developed to evaluate the reliability of software systems based on certain indicators. These indicators include: evaluating the development team, the level of documentation, the complexity of the program code, and data on errors and failures detected during the testing and operational phases.

The theoretical significance of the work. The research contributes to the development of the theoretical foundations for ensuring the reliability of software systems and plays a significant role in developing new models for improving their reliability. By using artificial intelligence-based forecasting algorithms, it's possible to more accurately evaluate the results of development and testing activities, directly influencing the optimization of resources and risk reduction in software projects. This research also provides a deeper understanding of the processes behind error generation in systems and the mechanisms for reliable operation of program code.

The practical significance of the work. The theoretical and practical results obtained can be used to improve the quality of program code, predict faulty modules and potential failures, determine the end time of the testing process, evaluate the reliability of software systems before they are released, and properly select and evaluate the work of the software development team.

Realization and application of results. The core scientific and theoretical findings of the dissertation were recognized as a significant scientific result and were incorporated into the 2024 annual report of the Department of Physical, Mathematical, and Technical Sciences of the Azerbaijan National Academy of Sciences, presented on the topic of "Software Engineering and the Analysis and Synthesis of Software Systems."

Approbation of the work and application of the results. The main scientific and practical results of the dissertation were presented and discussed at several international and national conferences, as listed below:

1. International Conference on Network Security and Communication Engineering (2015, United Kingdom)
2. The 2nd Republican Scientific and Practical Conference on "Multidisciplinary Problems of Information Security", dedicated to the 150th Anniversary of the International Telecommunication Union (2015, Baku, Azerbaijan)
3. The 1st Republican Scientific and Practical Conference on "Current Scientific and Practical Problems of Software Engineering" (2017, Baku, Azerbaijan)
4. The 3rd Republican Scientific and Practical Seminar on "Current Problems of Information Security" (2017, Baku, Azerbaijan)
5. The International Scientific Conference on "Information Systems and Technologies: Achievements and Prospects" (2018, Sumgayit, Azerbaijan)
6. The 4th Republican Conference on "Current Multidisciplinary Scientific and Practical Problems of Information Security" (2018, Baku, Azerbaijan)
7. Cybersecurity for Critical Infrastructure Protection via

Reflection of Industrial Control Systems, NATO Science for Peace and Security Series D: Information and Communication Security, (2022, Amsterdam))

8. The 3rd International Scientific Conference on "Information Systems and Technologies: Achievements and Prospects" (2022, Sumgayit, Azerbaijan)

Scientific publications. Based on the results of the dissertation, 18 scientific articles were published. Of these, 10 articles were published in peer-reviewed journals recommended by the Supreme Attestation Commission under the President of the Republic of Azerbaijan, and 8 theses were published in the proceedings of international and national conferences. Among these works, 2 articles were published in journals indexed in the Web of Science, and 4 articles and 1 thesis were published in Scopus-indexed journals.

The institution where the dissertation work is performed. The Institute of Information Technology of the Ministry of Science and Education of the Republic of Azerbaijan.

Structure and scope of the work:

The dissertation consists of 166 pages, including an introduction, 4 chapters, a conclusion, and a list of cited literature with 199 names, 25 tables and 35 figures.

GENERAL CHARACTERISTICS OF THE STUDY

In the introduction, the relevance of the dissertation is substantiated, and the purpose, tasks, research object, problems, and research methods are defined. The scientific novelty, theoretical, and practical significance of the results obtained in the dissertation are also demonstrated.

The first chapter is dedicated to "**The study of scientific and theoretical foundations and problems of ensuring the reliability of software systems**". **In the first section** of this chapter, the fundamental aspects of the software system (SS) reliability concept are explored, the SS reliability criteria are defined, and the scientific and theoretical foundations for ensuring reliability are extensively analyzed. **In the second section**, scientific research conducted in the field of enhancing SS reliability is systematically investigated, and the technologies used in this field along with the key factors affecting reliability are identified. **In the third section**, international practices for ensuring SS reliability are examined; the international standards applied in this area, assessment parameters, key research directions actively applied in global practice, and the scientific and theoretical problems arising in this context are identified [1, 7, 4, 8].

The second chapter is dedicated to **"The development of a method and model for the assessment of software system reliability". In the first section** of the chapter, the historical development of existing methods and models for evaluating and predicting software system (SS) reliability and their role in improving reliability are analyzed. A classification of these models based on various parameters is provided, and the intelligent methods and models applied to ensure the reliability of SS are also presented [3, 5].

According to the ISO/IEC 25010 standard, reliability is one of the 8 quality characteristics of a Software System and combines the criteria of maturity, recoverability, availability, and fault tolerance[2]. **In the second section of the second chapter**, the architecture of a

[2] ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model. https://www.standards.ru/document/3617603.aspx 19

conceptual model has been developed to enhance the reliability of SS and ensure reliability criteria (Figure 1). The conceptual model proposes a comprehensive approach to proactive measures for error prevention and reliability improvement, error detection and elimination, ensuring fault tolerance, failure prediction, and reliability assessment [16]. In the dissertation, methods and algorithms have been developed to solve the problems posed in the conceptual model.

The complexity of modern software systems is constantly increasing, which reduces reliability by increasing the number of potential errors. **In the third section of the second chapter**, a **hybrid method** is proposed for assessing the complexity of program code, taking into account the limitations of the LOC (Lines of Code), McCabe[3], and Halstead[4] metrics [12]. The proposed method combines three key aspects of complexity for evaluation:

- The structural complexity of the code;
- The number of linear operators;
- The number of interconnections between modules and components.

The complexity of the program code is calculated using the following formula:

$$C = E + \log_2(N - n) + r. \quad (1)$$

- N - The total number of operators in the program module;
- n - the total number of conditional and loop operators;
- r - the number of inter-module connections.

E - indicates the depth of nested loops and multiple branchings in the program code and is calculated using the following formula:

$$E = \sum_{i=0}^{n} (i+1)m_i, \quad i = 0,1,...,n \quad (2)$$

[3]McCabe, T.J., A Complexity Measure // IEEE Transactions on Software Engineering, – 1976, vol. SE-2, no. 4, – p. 308-320.

[4] Hariprasad, T. Software complexity analysis using halstead metrics / T. Hariprasad, G. Vidhyagaran, K. Seenu [et al] // *International Conference on Trends in Electronics and Informatics (ICEI)*, – Tirunelveli, – 2017, – pp. 1109-1113.

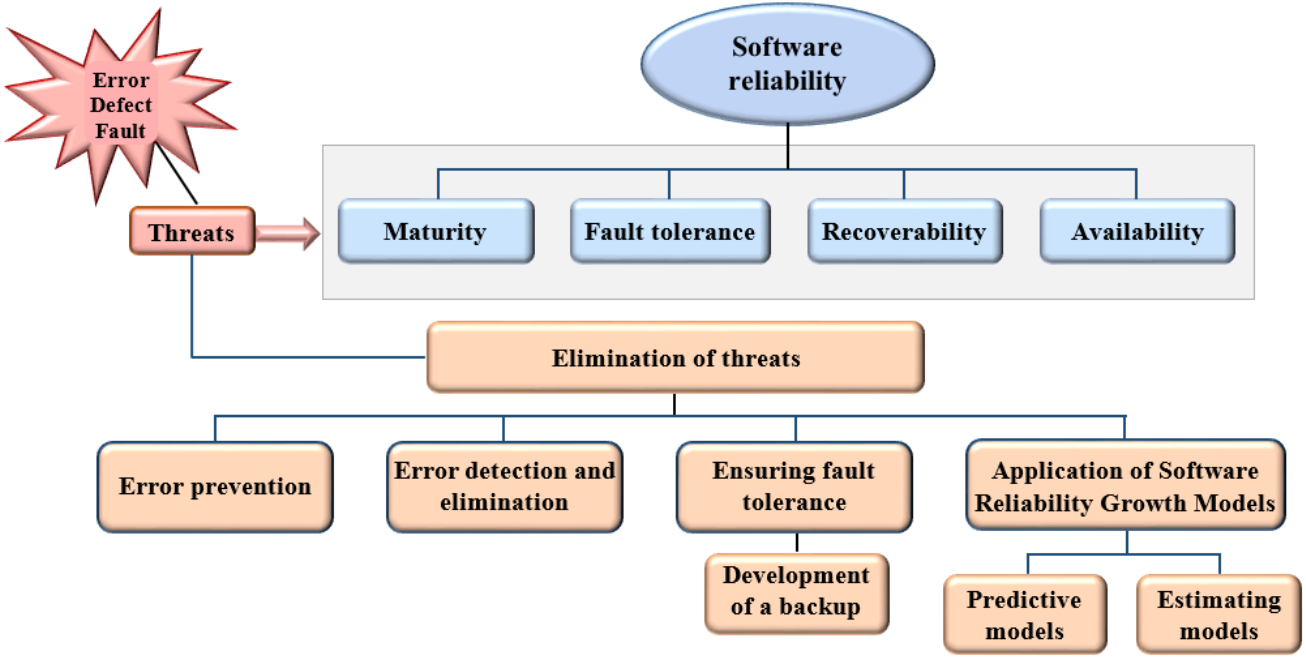

Figure 1. Architectural-technological principles for enhancing the reliability of software

– m_i – is the number of conditional and looping operators containing i conditional and loop operators.

– (N - n) the number of linear operators.

Table 1 presents a comparison of the proposed method with existing metrics.

Table 1.
Comparison of the hybrid method with existing metrics

Metric	Advantages	Disadvantages
LOC	İt calculates the number of lines in the program code, which has a limited impact on complexity.	The structural complexity of the program code and inter-module connections are not considered.
Halstead metric	The simplicity of the calculation allows for estimating the information volume and the number of potential errors in the program code.	Structural complexity and Lines of Code (LOC) are not considered. All operators and operands have the same weight in complexity calculation.
Cyclomatic Complexity (McCabe Metric)	It calculates the number of execution paths based on the count of loop and conditional operators.	Loop depth, inter-module connections, and LOC are not considered.
Proposed Hybrid Method	It maintains a balance between structure, volume, and interconnections.	The programming language dictionary is not considered

In the experiment, the complexity of program code written for the same function in C++ and Java (Appendix 3 and Appendix 4) was compared (Table 2). As the table shows, the value calculated by the hybrid method aligns with the expert assessment. In the McCabe metric, both programs were found to have the same complexity because all conditional and loop operators are treated equally, and the number of linear operators is not considered. The complexity of the program code can vary depending on the programming language selected. Calculating complexity helps in choosing the simpler of two programs and, when necessary, reducing complexity by refactoring the code.

Table 2.
Complexity of Program Code in C++ and Java

Metric	Calculation	C++		Java	
LOC	Number of Lines of Code in the Program		48		52
Mac Cabe	$C = e - n + 2p$	e=29, n=25	6	e=29, n=25	6
Expert assessment			3		4
Halstead	$D = (h_1/2) * (N_2/h_2)$	N_1=142 N_2=57 h_1=25 h_2=21	33.92	N_1=133 N_2=53 h_1=33 h_2=22	39.75
Hibryd	$E = \sum_{i=0}^{n} (i+1)m_i$ $C = E + \log_2(N-n) + r$	N=142 n=3, r=0, m_1=1 (i=1), m_0=2 (i=0)	9.11	N=133 n=5, r=0, m_0=3 (i=0), m_1=1 (i=1), m_2=1 (i=2)	10

The third chapter is dedicated to "**The Development of a Method and Algorithm for Predicting and Preventing Errors in Software Systems**". Errors and faults are an inevitable part of developing complex software systems and can lead to significant financial losses, reputational risks, and even security threats. Error management is a key aspect of ensuring the reliability of software systems.

In the first section of this chapter, the concepts of fault, error, and failure in software systems are explained, along with a classification of errors and methods for their detection [10, 11]. Errors discovered after a software system has been tested and implemented are classified based on their criticality, repair priority, phase of occurrence, and scope of impact. Classifying the discovered errors by determining

their cause and type facilitates their management, reduces testing costs, and prevents the recurrence of failures [9].

In the second section of the third chapter, an Ensemble Method (EM) based on machine learning is proposed for detecting faulty modules in software systems [13]. The architecture of the decision-making system for this method is shown in Figure 2. As the figure illustrates, the decision-making system consists of three classifiers. The combination of a linear model (Logistic regression - LR), a tree-based non-linear model (Random Forest - RF), and rule-based classifier model (Projective Adaptive Resonance Theory - PART) allows for the capture of various dependencies within the data.

Assume that the 21 features (table 3) used in the prediction are given by the vector $X = [x_1, x_2, ..., x_{21}]^T$.

Step 1. Prediction using LR:

The linear combination of features is calculated:

$$z_{LR} = w_1 x_1 + w_2 x_2 + ... + w_{21} x_{21} + b_{LR} = W_{LR}^T X + b_{LR}$$

where $W_{LR} = [w_1, w_2, ..., w_{21}]^T$ - is the weight vector, b_{LR} – is the bias.

The probability of fault:

$$P_{LR}(y = 1|x) = \sigma(z_{LR}) = \frac{1}{1 + e^{-z_{LR}}}$$

Step 1. Prediction using RF:

- Number of decision trees: N=100
- A bootstrap sample, D_i, is created from the initial data.
- At each branching step, a random subset of m features is selected ($m \le 21$, $m = \sqrt{x}$). The next split is chosen only from this subset.
- The RF model "learns" to identify the relationships between the features and the target variable using the initial data.
- Each tree, $T_i(X)$, produces a prediction: $\hat{p}_i = P(y = 1|x, T_i)$

The final prediction is calculated as the average of the probabilities:

$$P_{RF}(y = 1|x) = \frac{1}{N} \sum_{i=1}^{N} \hat{p}_i$$

Table 3.
Software metrics

	Metric	Description	
1.	LOC	Number of code lines	McCabe
2.	v(g)	Cyclomatic complexity	
3.	ev(g)	Basic complexity	
4.	iv(g)	Design complexity	
5.	n	Total number of operators and operands	Halstead
6.	v	Volume	
7.	l	Software length	
8.	d	Difficulty	
9.	i	Intelligence	
10.	e	Effort	
11.	b	Error	
12.	t	Time	
13.	lOCode	Number of code lines	
14.	lOComment	Number of comment lines	
15.	lOBlank	Blank lines	
16.	lOCodeAnd Comment	Number of comment and code lines	
17.	uniq_Op	Number of unique operators	
18.	uniq_Opnd	Number of unique operands	
19.	total_Op	Number of total operators	
20.	total_Opnd	Number of total operands	
21.	branchCount	Number of total graph branches	

Step 3. Bagging with PART (B-PART):

- Number of PART algorithms: N=100
- The 21-dimensional input vector, $x^{(l)}$, is projected onto a smaller m'-dimensional space.
- $x'^{(l)} = P_j x^{(l)}$, the first training sample is accepted as the prototype (w'_{ji}) (class 1 - faulty).

– The similarity between the projection of the next training sample $x'^{(l)}$ and the w'_{ji} prototype is calculated:

$$similarity\ (x'^{(l)}, w_{ji}) = \frac{\left|x'^{(l)}\right\| w'_{ji}\right|}{x'^{(l)} \cdot w'_{ji}}$$

– If the similarity with the prototype is greater than or equal to ρ^j (the vigilance parameter), resonance occurs, and the prototype's weights w_{ji} are updated:

$$w_{ji}^{new} = (1-\beta_j) w_{ji}^{old} + \beta_j x^{(l)}$$

β_j - is the learning rate.

Aggregation of PART predictions:

$$P_{B-PART}(y=1|x) = \frac{1}{K}\sum_{i=1}^{K} P_{PART\_J}(y=1|x)$$

Step 4. Prediction of the ensemble model is calculated as the average of the predicted probabilities from LR, RF, and B-PART:

$$P_{ensemble}(y=1|x) = \frac{P_{LR}(y=1|x) + P_{RF}(y=1|x) + P_{B-PART}(y=1|x)}{3}$$

Then, a threshold value is applied to determine the faulty or non-faulty class. The class prediction is based on a predefined threshold value h (for example, 0.5). If

$$\hat{y}_{ensembl} = \begin{cases} 1, & \text{if } P_{ensembl}\left(y=1|x\right) \geq h \\ 0 & \end{cases}$$, 1- faulty, 0-fault-free.

The data used in this experimental study was obtained from the open-source NASA Promise and GitHub repositories (table 4). Classification was performed on this dataset using 10-fold cross-validation. Several ML algorithms were used for comparison. The experiments were conducted in the Weka software environment.

To evaluate the effectiveness of the proposed model, the *ROC curve* (table 5) and the *F-score* (table 6) were used. The values for both the *ROC curve* and the *F-score* range within the interval (0, 1), with a value of 1 being considered an ideal model.

Table 4.
Used Datasets

	Datasets	Number of Modules	Number of defective modules
1.	CM1	498	49
2.	JM1	10885	2106
3.	KC1	2109	326
4.	KC2	522	107
5.	PC1	1109	77
6.	DATATRIEVE	130	11
7.	KC3	194	36
8.	MC1	1952	46
9.	PC2	722	16
10.	PC3	1053	153

Tables 5 and 6 compare the performance of various machine learning (ML) algorithms and the proposed Ensemble Method (EM) based on their F-score and ROC curve values.

Figures 3 and 4 show a comparison of the effectiveness of the Ensemble Method with the RF algorithm, which performed the best among the ML algorithms.

Table 5.
Comparison of the EM with existing methods by F-measure

	NB	SMO	J48	RF	PART	IBk	MLP	EM
CM1	**0,858**	0,852	0,852	0,854	0,851	0,843	0,845	0,846
Datatrieve	0,840	?	0,867	0,883	0,863	0,883	0,873	**0,883**
JM1	0,770	0,722	0,769	**0,783**	0,763	0,766	0,741	0,767
KC1	0,820	0,786	0,832	**0,843**	0,817	0,836	0,828	0,837
KC2	0,821	0,784	0,810	0,825	0,810	0,802	**0,835**	0,814
KC3	0,785	0,743	0,783	0,769	0,762	0,708	0,762	**0,811**
MC1	0,937	?	0,973	0,976	0,970	0,977	0,974	**0,978**
PC1	0,895	0,897	0,921	0,927	**0,928**	0,921	0,917	0,924
PC2	0,930	?	0,965	?	0,963	0,958	0,965	**0,966**
PC3	0,458	?	0,839	0,840	0,845	0,843	0,842	0,848

Table 6.
Comparison of the proposed method with existing methods by ROC-area value

	NB	SMO	J48	RF	PART	IBk	MLP	EM
CM1	0,658	0,497	0,558	0,750	0,721	0,589	0,734	**0,789**
Datatrieve	0,736	0,500	0,482	0,685	0,550	0,574	0,751	**0,791**
JM1	0,679	0,502	0,653	0,755	0,712	0,640	0,690	**0,763**
KC1	0,790	0,516	0,689	0,823	0,747	0,735	0,771	**0,831**
KC2	0,830	0,597	0,704	0,825	0,704	0,643	0,828	**0,831**
KC3	0,662	0,514	0,653	0,736	0,601	0,539	0,639	**0,759**
MC1	0,747	0,500	0,566	0,850	0,580	0,665	0,728	**0,886**
PC1	0,650	0,500	0,668	0,875	0,814	0,740	0,723	**0,884**
PC2	0,717	0,500	0,463	0,780	0,605	0,551	0,770	**0,804**
PC3	0,749	0,500	0,591	0,832	0,767	0,603	0,783	**0,843**

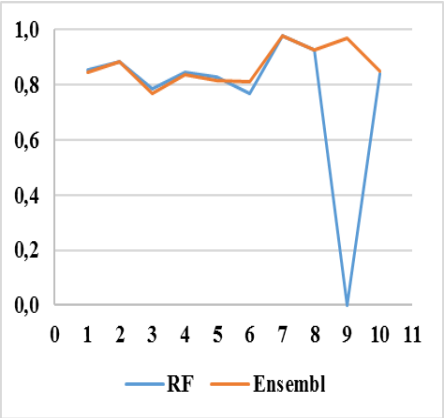

Figure 3. Comparison of RF and EM efficiency by ROC-area value

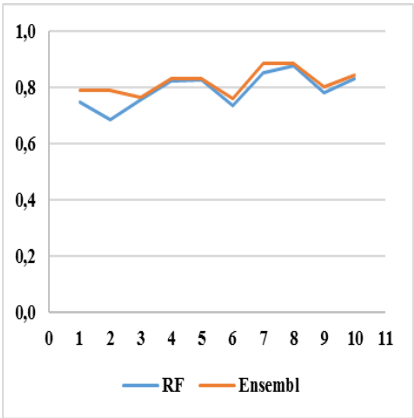

Figure 4. Comparison of RF and EM efficiency by F-measure value

Unlike the RF method, the proposed ensemble method demonstrated stable results on both small- and large-scale software systems. It is clear from the figure that the ensemble model is superior to the RF model on both metrics and possesses a more stable predictive

capability. This research provides a basis for concluding that the application of ensemble methods for predicting software system errors yields more accurate results compared to individual machine learning methods.

The third section of the third chapter is dedicated to the analysis of subjective factors affecting the reliability of the software system and the development of a method for their elimination [6].

The calculation used the following statistical indicators for each member of the team:

N_1 – number of tasks completed on time;

N_2 – number of tasks completed ahead of schedule;

N_3 – number of tasks completed late;

N_4 – number of uncompleted tasks;

T_1 – time to complete successfully finished tasks;

T_{1a} – time to complete successfully finished analogous tasks;

T_2 – total break time during the completion of successfully finished tasks;

T_{2a} – total break time during the completion of successfully finished analogous tasks;

N_v – number of successfully finished tasks;

N_{va} – number of successfully finished analogous tasks;

H_s – number of errors made on all previously completed tasks;

H_u – total number of errors made by all programmers on all completed tasks;

$V_1 = \frac{T_1 - T_2}{N_v}$ – average time for the programmer to complete successfully finished tasks;

$V_2 = \frac{T_{1a} - T_{2a}}{N_{va}}$ – average time for the programmer to complete successfully finished analogous tasks.

Based on these indicators, the programmer's work efficiency is calculated according to four parameters:

1. Efficiency based on time spent on successfully completed tasks:

$$C_v = 1 - V_2 / V_1 \tag{3}$$

2. Efficiency based on the number of errors:

$$C_s = 1 - \frac{H_s}{H_u} \tag{4}$$

3. Efficiency based on the number of successfully completed tasks:

$$C_f = \frac{a_1 N_1 + a_2 N_2 + a_3 N_3 + a_4 N_4}{N} \tag{5}$$

a_1, a_2, a_3, a_4 - are the weighting coefficients.

4. Expert Evaluation:

$$C_e = \frac{1}{\delta} \sqrt[3]{\prod_{j=1}^{3} \sum_{i=1}^{\delta} b_{ij}} \tag{6}$$

δ - number of experts;

b_{i1} - *i*-th expert's rating of the programmer's task completion speed;

b_{i2} - *i*-th expert's rating of the programmer's task completion quality;

b_{i3} - *i*-th expert's rating of the programmer's knowledge and skills;

Overall programmer work efficiency based on all parameters:

$$E = \frac{\omega_1 C_v + \omega_2 C_f + \omega_3 C_s + \omega_4 C_e}{4} \tag{7}$$

$\omega_1, \omega_2, \omega_3, \omega_4$ - are the weighting coefficients that depend on the type of project.

To demonstrate the effectiveness of the proposed method, the following experimental values were calculated by evaluating the performance of three programmers.

In Tables 7 and 8, their evaluation by two experts based on three parameters and other statistical indicators are provided. Based on these, the efficiency values of C_e, C_v, C_s, C_f , and E were calculated for each programmer.

To demonstrate the significance of the calculated metrics, Table 9 presents the total number of tasks solved by the programmers and the number of successfully solved tasks.

Table 7.
Calculation of C_e and C_f values

	b_{11}	b_{12}	b_{13}	b_{21}	b_{22}	b_{23}	C_e	N_1	N_2	N_3	N_4	N	C_f
$P1$	5	4	5	4	4	4	**4,3**	18	7	3	0	28	**2.20**
$P2$	4	5	4	3	4	5	**4,1**	26	9	4	1	40	**3.08**
$P3$	5	5	5	5	5	5	**5,0**	23	8	1	0	32	**2.58**

Table 8.
Calculation of C_v and E values

	T_1	T_2	T_{1a}	T_{2a}	N_v	N_{va}	C_v	H_s	H_u	C_s	E
$P1$	345	24	194	11	25	16	**0,12**	12	35	**0,66**	**1,83**
$P2$	372	20	305	13	35	30	**0,03**	8	35	**0,77**	**2,00**
$P3$	150	8	250	14	6	15	**0,34**	4	35	**0,89**	**2,20**

Table 9.
Number of Projects and Successful Tasks

	Total number of projects (N)	**Number of Successful Tasks** ($N_1 + N_2$)
$P1$	28	25
$P2$	40	34
$P3$	32	31

These values are shown graphically in Figure 5. Based on these graphs, it can be concluded that programmer $P2$ worked more efficiently than programmer $P1$ and $P3$.

The values calculated using the proposed method and shown in Figure 6 indicate the opposite. This is because, when calculating programmer productivity with our proposed metrics, we considered important factors beyond just the total number of tasks completed.

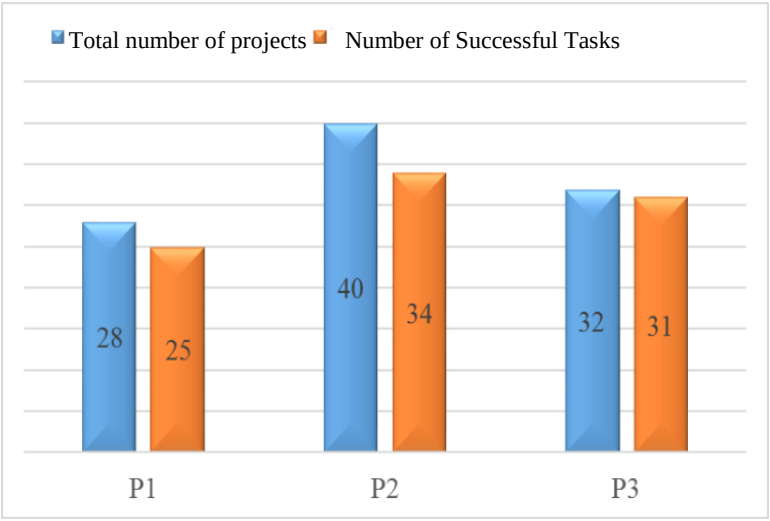

Figure 5. Number of Completed Tasks

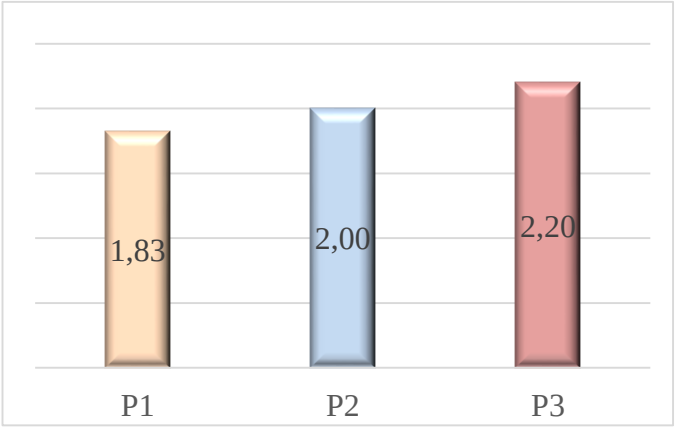

Figure 6. Evaluation based on the proposed method

These factors include work breaks, experience in the relevant field, the number of unsuccessful projects, and the number of errors.

Since the parameter for relevant work experience is used in these calculations, the method can also be effectively applied for the proper distribution of tasks.

The fourth chapter is dedicated to "**The Development of a Method and Algorithm for Enhancing the Reliability of Software Systems**". In the first section of this chapter, a conceptual model for a system that ensures the fault tolerance of software systems was developed [2, 18].

The proposed system consists of 6 subsystems, and its operating principle is outlined below.

Step 1. The monitoring system continuously oversees the state of the SS, collecting various data about its current condition:

- Information about resources (CPU load, memory capacity, disk space, speed, etc.);
- Data on hardware status (temperature, voltage, and other parameters indicating potential issues);
- Queries processed by the system;
- Data processing speed;
- Error-related information from log files;
- System error alerts;
- System availability (uptime and downtime);
- Unauthorized access attempts;
- Detection of malicious activities, etc.

This data is gathered from sensors, agent programs installed on servers, log file analysis tools, and other specialized software add-ons. The monitoring system operates continuously in real-time. Data can be presented in the form of graphs, tables, or alert notifications.

Step 2. Anomaly Detection System analyzes the data received from the monitoring system to detect deviations based on predefined rules and norms. It compares current data with historical data or established threshold values. When an anomaly is detected, it sends an alert to other system components and personnel.

To detect anomalies, the system uses statistical or intelligent methods, such as machine learning and deep learning, which are trained to identify deviations from normal behavior. It can use simple rules and thresholds for basic anomalies, or more complex algorithms for sophisticated issues. By analyzing data and making decisions, the anomaly detection system enables the timely discovery of potential problems, prevents failures, and ensures stable operation of the software system.

Step 3. Diagnostic System analyzes the data collected by the monitoring and anomaly detection systems. It can be thought of as an "expert system" that uses knowledge and experience to determine the root cause of anomalies. This system can include:

- Knowledge bases about the possible causes of anomalies.
- Analyzers for logs and other data.
- Automated diagnostic algorithms to identify the reasons behind anomalies.

The knowledge bases may contain rules, scenarios, typical problems, and their solutions. Automated diagnostic algorithms are capable of analyzing large volumes of data to uncover complex patterns. The system identifies the factors causing an anomaly, which could be errors in the program code, hardware problems, network failures, or other issues. It then provides recommendations for resolving the root cause.

Step 4. Automatic Recovery System automatically responds to anomalies identified by the diagnostic system and takes action to restore normal operation. Its main goal is to minimize downtime and prevent serious failures.

This system includes a recovery block (with tools for managing scenarios and resources), tools for failover to backup resources, and mechanisms for system restarts and scaling. In response to a detected problem, the system automatically performs predefined actions, such as restarting components, switching to backup resources, or scaling resources.

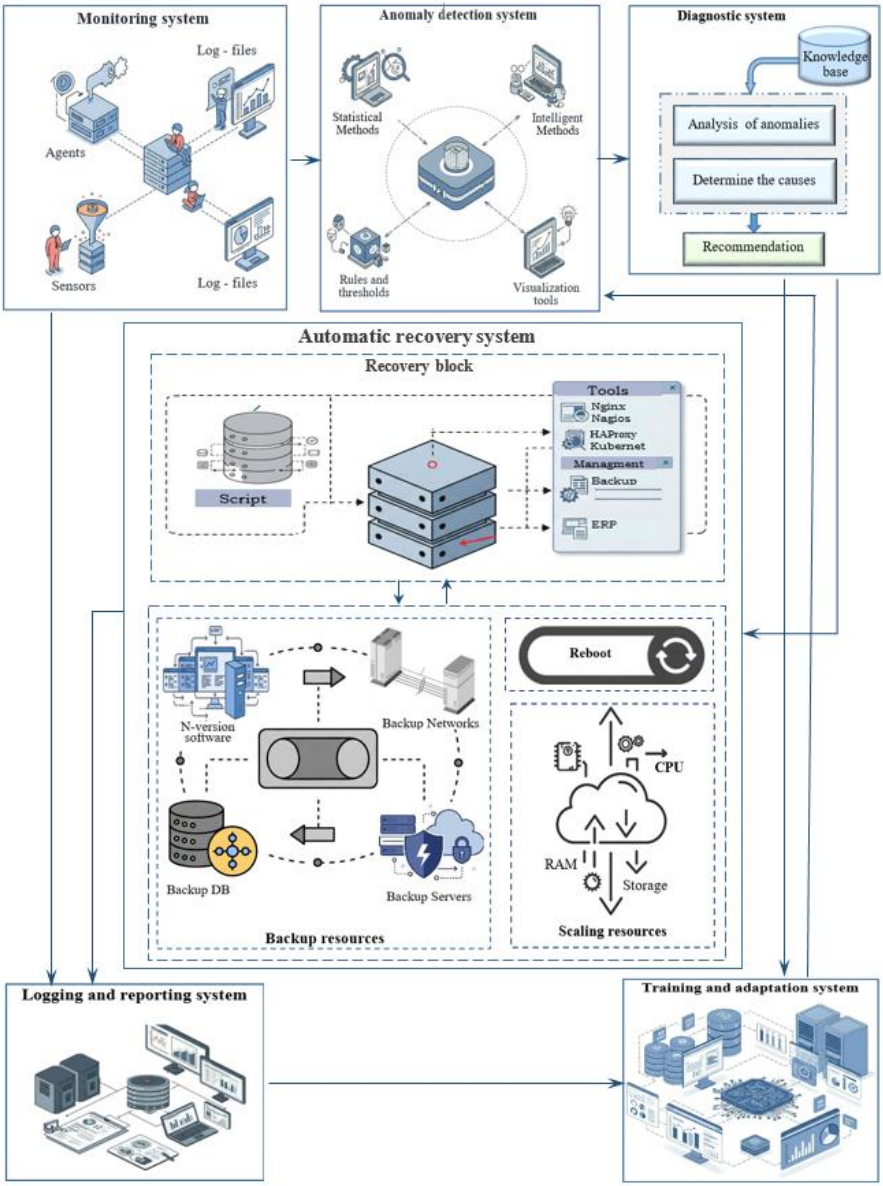

Figure 7. A conceptual model for increasing the software fault tolerance

Step 5. Logging and Reporting System logs all important events that occur within the software system and generates reports for analysis. Its purpose is to track events, identify patterns, and analyze problems. The system records information about various events, including errors, warnings, configuration changes, and user activity. It creates reports that allow for analysis of the collected data over a specific period, which helps to identify historical trends.

Step 6. Learning and Adaptation System analyzes the data collected by the monitoring and logging systems, as well as the outcomes from other components. It identifies patterns and uses this information to improve the operational algorithms of other system components. Its objective is to make the system more effective, reliable, and fault-tolerant.

The application of this proposed approach in the development of fault-tolerant systems enables the creation of more reliable and stable software systems.

In the second section of the fourth chapter, a method for predicting failures in software systems was developed [15].

An LSTM network (Figure 8) yields good results in predicting complex-structured time series. Because data about failures and errors are also complex-structured time series, a method based on an LSTM network is proposed in this work to predict failures in Software Systems. The operating principle of the proposed method consists of the following steps:

Step 1. The Forget Gate determines which information from the cell state C_{t-1} should be discarded or kept. The gate's output, f_t, is evaluated in the range of [0,1] based on the input, x_t, and the previous hidden state, h_{t-1}. The resulting value determines which portion of the information is passed through. A value of 0 means no information is passed, while a value of 1 means all information is passed.

$$f_t = \sigma(W_f \cdot h_{t-1} + U_f \cdot x_t + b_f)$$

Step 2. The Input Gate i_t determines which values will be updated in the cell state:

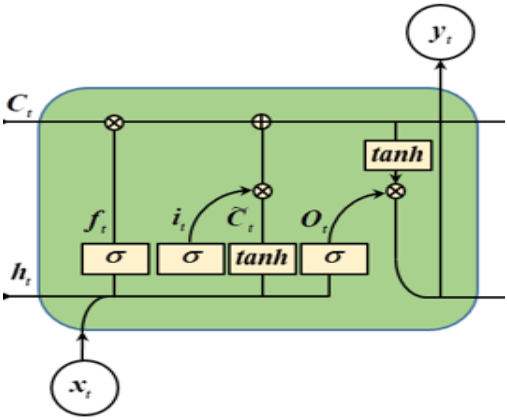

Figure 8. Structure of LSTM Cell

$$i_t = \sigma(W_i \cdot h_{t-1} + U_i \cdot x_t + b_i)$$

Step 3. A new candidate vector of values, $\tilde{C}_t$, is calculated to be added to the cell state:

$$\tilde{C}_t = \phi(W_C h_{t-1} \cdot U_C x_t + b_C)$$

Step 4. The old cell state, C_{t-1}, is updated to the new cell state, C_t:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Step 5. Calculating the Output Value: First, the output gate, o_t, is evaluated in the [0,1] interval based on the current input, x_t, and the previous hidden state, h_{t-1}:

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

Step 6. Calculating the Final Output. The cell state, C_t, is evaluated in the [−1,1] interval by passing through a layer with a tanh activation function. The final output value, h_t, is then calculated:

$$h_t = o_t \cdot \varphi(C_t)$$

$$LSTM_{out} = h_t$$

W, *U* – weights, *b* – bias.

To evaluate the performance of the proposed method, seven real-world datasets on software system failures were used (Table 10).

Table 10.
Used Datasets

Data Set	Number of faults	LOC	Software type
DS1	136	21.700	Real-time command and control system
DS2	46	40.000	On-line Data Entry
DS3	535	870.000	Realtime Control Application
DS4	213	38.500	Flight Dynamic Application
DS5	266	-	Realtime Control Application
DS6	181	-	Realtime Control Application
DS7	81	-	Brazilian Electronic Switching System

The experiments were conducted in a Python environment. For hyperparameter optimization, the AutoML library was utilized, and the effectiveness was assessed using three criteria:

1. Root Mean Square Error (RMSE)
2. Average Error (AE)
3. Coefficient of Determination

The coefficient of determination indicates how well a method fits the dataset, with its value ranging from 0 to 1. A value higher than 0.8 suggests that the proposed method is sufficiently good. To compare the model's effectiveness with existing models based on the Root Mean Square Error (RMSE) and the coefficient of determination, the DS1 dataset was used (Table 11).

Table 12 presents a comparison of the model's Average Error (AE) with the PRNNDWCM, PNNE, and DWCM hybrid neural network models proposed in recent years.

The results of the experiments prove the effectiveness of the proposed model. The main advantage of this model is that its prediction results are positive across datasets of various sizes and from different application domains.

Table 11.
Comparison of the performance of the proposed model with other models in terms of RMSE and R^2

	Neural Net	Saburag-Goel	Goel-Okumoto	LSTM
RMSE	0.0952	2.539	3.926	**0.051**
R^2	0.9958	0.9911	0.9901	**0.9995**

Table 12.
Comparison of LSTM and hybrid models

	PRNNDWCM	PNNE	DWCM	LSTM
AE	0.0112	0.0129	0.0176	0.0039

The third section of the fourth chapter is dedicated to the development of a method and an algorithm for evaluating the reliability of software systems based on certain indicators [14, 17].

This work proposes a reliability evaluation method for software systems that utilizes both expert evaluation and an LSTM-GRU hybrid method. The reliability of the software system is calculated based on the following features:

- Evaluation of the team working on the software system.
- Level of software documentation.
- Complexity of the program code.
- Data on errors and failures discovered during the software testing process.
- Data on errors and failures discovered during the software operational process.

1. *Team Evaluation.* During the team evaluation, each feature may have its own weight, and an integral value can be obtained by using these weights. Let's assume that *N* features are used to evaluate the software development team. These features form the vector $K = (k_1, k_2, ..., k_N)$, and the corresponding weight vector can be

denoted by $W = (w_1, w_2, ..., w_N)$. The values of the weight vector are refined through various experiments. Here, a certain condition $\sum_{i=1}^{N} w_i = 1$ must be met. The team can be evaluated as follows:

$$x_1 = \sum_{i=1}^{N} w_i k_i \quad (8)$$

2. *Evaluation of Software Documentation.* During the development of a software system, various operational documents are created. These documents are analyzed to evaluate the completeness and relevance of the technical documentation, including requirements, architectural diagrams, and installation guides. The ratings for this evaluation can be given as follows:

- Fully and accurately documented: 8-10;
- Good coverage: 5-7;
- Sufficient coverage: 3-4;
- Insufficiently covered: 1-2;
- Poorly covered: 0.

3. *Evaluation of Program Code Complexity.* Relevant metrics can be used for this evaluation, such as the method proposed in Chapter II, Halstead metrics, or cyclomatic complexity metrics.

4. *Number of Errors and Failures During Testing.* The number of errors and failures discovered during the software testing process is calculated.

5. *Number of Errors and Failures During Operation.* The number of errors and failures discovered during the software operational process is calculated.

Based on these features, the reliability of the software system is calculated using the proposed expert evaluation. The resulting values will then be used as the output data for the LSTM-GRU neural network.

Expert evaluation method for the reliability of software systems is given as follows:

$$R = \frac{1}{5}\sum_{i=1}^{5} \omega_i f_i , \quad (j = 1,...,5), \tag{9}$$

f_1 - the parameter for evaluating the team;

f_2 - data obtained from the initial testing of the software;

f_3 - documentation level evaluation parameters

f_4 - data obtained during the operation of the software system;

f_5 - is the evaluation parameter for the program code's complexity;

ω_i - are the weights of the parameters f_i (the reliability value coefficient).

f_1 - is calculated according to formula (6).

Suppose we are given the evaluation of $n \in N$ experts on $f_1, f_2,..., f_5$ and $\omega_i (i = 1,2,...,5)$ in an $\lambda \in N$-point system. We can then calculate the weights, ω_i, using the formula given below:

$$\omega_j = \frac{1}{2n\lambda}\sum_{i=1}^{n} P_{ij} + \frac{1}{2\lambda}\left(\prod_{i=1}^{n} P_{ij}\right)^{\frac{1}{n}}, \ (j = 1,...,5), \ \omega_j \in [0,1], \tag{10}$$

P_{ij} is the i-th expert's evaluation of the weight ω_j. Note that the parameters, $f_2,..., f_4$, can be calculated as follows:

$$f_l = \frac{1}{nk\lambda}\sum_{i=1}^{k} \gamma_{jl} a_{lij} , \quad (l = 2,3,4), \ \gamma_{jl} \in [0,1] \tag{11}$$

a_{lij} - is the *i*-th expert's rating for the *j*-th factor of parameter f_l on an λ-point scale.

γ_{jl} - is the weight of the *j*-th factor of parameter f_l.

k - is the number of factors in parameter f_l.

The weights γ_{jl} can be calculated based on formula (10).

To evaluate the reliability of the software system, the five features analyzed previously were selected as the input data for the LSTM-GRU model. (Note that N number of features can be taken, where N is the number of inputs to the neural network.) The output data for the

model is the expert evaluation of the software system's reliability based on these features.

The model is trained using this data. In subsequent evaluations, when the five features are provided as input, the trained neural network can then evaluate the reliability of the software system.

Problem Statement. This work focuses on the development of a hybrid LSTM-GRU based method for evaluating the reliability of software systems, as shown in Figure 9.

Input Data: A vector $X = (x_1, x_2, ..., x_5)$ consisting of 5 features.

Output Data: The calculated reliability value of the software, denoted as *R*.

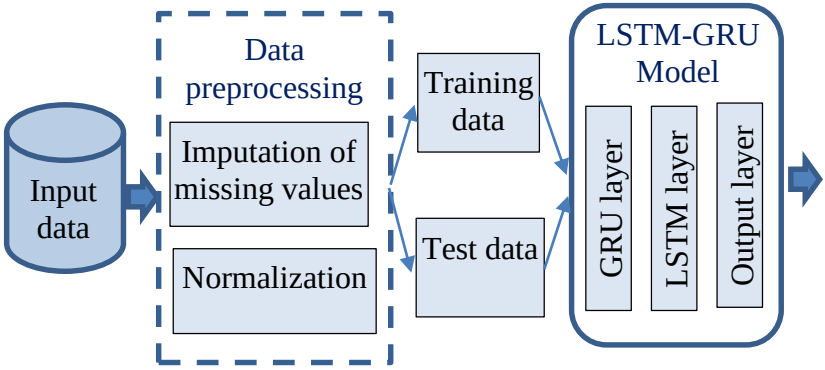

Figure 9. LSTM-GRU Model

Step 1: A synthetic dataset of 1000 data points is created.

Step 2: The value of each feature is scaled to the [0,1] interval using MinMaxScaler.

Step 3: The Input Layer accepts sequences consisting of 5 features.

Step 4: The GRU Layer processes the input sequence and models the dependencies between features. The GRU uses update and reset gate mechanisms to control the flow of information and remember important points in the sequence.

Step 5: The LSTM Layer further processes the output from the GRU layer, enhancing the network's long-term memory and its ability to model complex time series.

Step 6: The Output Layer consists of a Dense layer with a Softmax activation function for evaluating reliability. It calculates output values to assess the reliability level, for example:

– Low reliability: 0 - 0.3
– Medium reliability: 0.3 - 0.7
– High reliability: 0.7 - 1.0

The experiments were conducted in a Python environment.

The scatter plot presented in Figure 10 visually demonstrates the relationship between the actual reliability values of software systems and the values predicted by the hybrid LSTM-GRU model. The closer the points on the graph are clustered to the diagonal line (i.e., the ideal *x=y* line), the more accurate the model's predictions are.

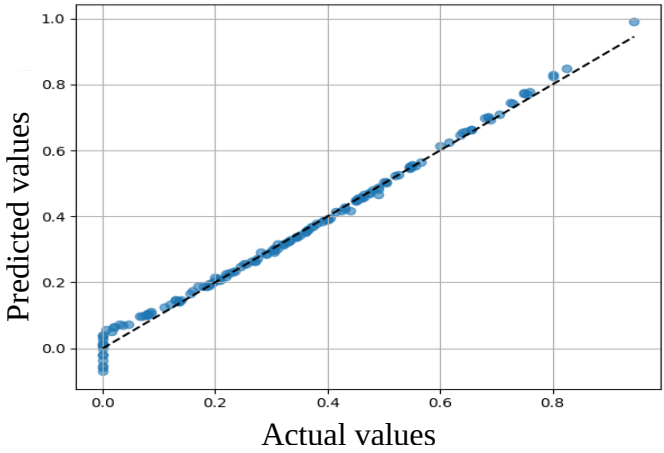

Figure 10. The relationship between the actual and predicted values

Based on the graph, almost all the points are located directly on the diagonal line or are distributed very close to it. This tight clustering indicates that the model has high accuracy in its reliability predictions and that the predictions are very close to the actual values. The minimal spread of points around the diagonal line also confirms the low number of prediction errors.

The Mean Absolute Error (MAE) value used to quantitatively evaluate the model's performance confirms these visual observations.

The low MAE value obtained (for example, a small value like 0.0039) proves that the model's predictions differ from the actual values by only a slight amount on average. This demonstrates the model's ability to provide sufficiently reliable predictions for practical applications.

In conclusion, the hybrid LSTM-GRU model has been shown to exhibit high efficiency in reliability evaluation and establishes a strong correlation between the actual and predicted values.

RESULTS

In the process of developing this dissertation, the objectives set for the proposed topic were achieved, and the results obtained are outlined below:

1. A new hybrid method for calculating program code complexity was developed, combining the strengths of LOC, Halstead, and McCabe metrics. This method enables the comparison of code written in different programming languages and facilitates decisions regarding refactoring and optimization [4, 8, 12, 16].
2. An ensemble method was developed for detecting faulty modules by combining the machine learning algorithms LR, RF, and PART. This method can be useful for predicting faulty modules and determining the completion time of the testing process [9, 10, 11, 13].
3. An analysis of the subjective factors (human factor) affecting the reliability of software systems was conducted, and a method for their elimination was developed. This can be used for the proper formation of a software development team, the effective distribution of tasks, and objective labor evaluation, among other things [1, 3, 5, 6, 7].
4. A conceptual model for a fault-tolerant system was proposed, and a method based on the LSTM recurrent neural network was developed for predicting failures. Artificial intelligence-based prediction algorithms help to more efficiently use resources and minimize project risks during the development and testing phases [2, 15, 18].

5. A method for evaluating the reliability of software systems was developed, consisting of a sequential combination of LSTM-GRU deep learning algorithms. This method is based on specific indicators such as team evaluation, documentation level, code complexity, and data on errors and failures discovered during both testing and operation. Its application can serve as a valuable tool for early-stage reliability assessment and timely management of potential problems in software development processes [14, 17].

The main content of the dissertation is published in the following scientific works:

1. Bayramova T.A. Program engineering, development stages, problems, and perspectives in Azerbaijan // Network Security and Communication Engineering: Proceedings of the 2014 International Conference on Network Security and Communication Engineering . Hongkong, – 2015, – p. 657-659. (**Scopus**)
2. Kazımov T.H., Bayramova T.A. Proqram sistemlərinin təhlükəsizliyi və etibarlılığının təmin edilməsində özünüidarə mexanizmləri // İnformasiya təhlükəsizliyinin multidissiplinar problemləri üzrə II respublika elmi-praktiki konfransı. – Bakı, –14 may –2015, – s. 242-245.
3. Kazımov T.H., Bayramova T.A. Soft kompütinq metodları əsasında proqram təminatının etibarlılığının təmin edilməsi // İnformasiya təhlükəsizliyinin aktual problemləri III respublika elmi-praktiki seminarı. – Bakı, – 8 dekabr –2017, – s. 73-76.
4. Kazımov T.H., Bayramova T.A. Proqram mühəndisliyinin elmi-nəzəri əsasları haqqında // Proqram mühəndisliyinin aktual elmi-praktiki problemləri I respublika konfransı. – Bakı, – 17 may, 2017, – s. 10-14.
5. Kazımov T.H., Bayramova T.A. Proqram təminatı etibarlılığının qiymətləndirilməsi modelləri // İnformasiya Texnologiyaları Problemləri, – 2017, 8(1) – s. 112-118.
6. Kazımov T.H., Bayramova T.A. Proqram təminatı işləyən komandaların idarə edilməsi üçün metrikalar // "İnformasiya təhlükəsizliyinin aktual multidissiplinar elmi-praktiki problemləri" IV respublika konfransı. – Bakı, – 14 dekabr, – 2018. – s. 142-144.
7. Bayramova T.A. Proqram mühəndisliyinin metodları prosesləri və vasitələri // İnformasiya sistemləri və texnologiyalar nailiyyətlər və perspektivlər beynəlxalq elmi konfrans. – Sumqayıt, – 15 noyabr, – 2018, – s. 69-72.

8. Bayramova T.A. Proqram mühəndisliyi standartların analizi // İnformasiya Cəmiyyəti Problemləri. – 2020, 11(1) – s. 83–95.
9. Kazimov T.H., Bayramova T.A., Malikova N.D. Research of intelligent methods of software testing // System Research & Information Technologies, – 2021. № 4. – p. 42- 52. (**Scopus**)
10. Bayramova T.A. Proqram təminatı xətalarının klassifikasiyası // İnformasiya sistemləri və Texnologiyalar nailiyyətlər və Perspektivlər III beynəlxalq elmi konfransı. – Sumqayit, 8-9 dekabr, – 2022, – s. 256-258.
11. Bayramova T.A. Analysis of Modern Methods for Detecting Vulnerabilities in Software for Industrial Information Systems // Cybersecurity for Critical Infrastructure Protection via Reflection of Industrial Control Systems, NATO Science for Peace and Security Series D: Information and Communication Security. – Amsterdam, – 2022, – p. 160-162.
12. Kazimov T.H., Bayramova, T.A. Development of a hybrid method for calculation of software complexity // System Research & Information Technologies, – 2022. № 2. – p. 32-44. (**Scopus**)
13. Bayramova T.A. Software defect prediction using the machine learning methods // Problems of Information Technology. – 2023, 14 (2), – p. 23-31.
14. Bayramova T. A. Development of a Method for Software Reliability Assessment using Neural Networks //Procedia Computer Science, – 2023, vol. 230, – p. 445-454. (**WoS, Scopus**)
15. Bayramova T. Predicting the reliability of software systems using recurrent neural networks: LSTM model // Problems of Information Technology. – 2024, 15(1) – p. 52-61.
16. Bayramova T. A., Malikova N.C. Developing a conceptual model for improving the software system reliability // Problems of Information Society, – 2024, 15(1) – p. 42-56.
17. Bayramova T. A., Kazimov T. H. Application of LSTM-GRU combined model to calculate the reliability of software systems

// Procedia Computer Science, – 2025, vol. 252, – p. 127-135. (**WoS, Scopus**)

18. Bayramova T.A. Development of a conceptual model for ensuring fault tolerance of software systems // Problems of Information Technology, – 2025, 16(1) – p. 35-46.

The defense of the dissertation will take place in 24.10.2025 at 13.00 at the meeting of the dissertation council ED 1.20, operating under the Institute of Control Systems of the Ministry of Science and Education of the Republic of Azerbaijan.

Address: AZ 1141, Baku, B. Vahabzade street, 68

The dissertation work can be found in the library of the Institute of Control Systems of the Ministry of Science and Education of the Republic of Azerbaijan.

Electronic versions of the dissertation and abstract are posted on the official website of the Institute of Control Systems of the Ministry of Science and Education of the Republic of Azerbaijan.

The abstract was sent to the required addresses in 22.09.2025

Printed: 18.09.2025
Paper format: 60 × 84 1⁄16
Volume: 38 342 characters
Printing: 20 copies