Journal homepage: https://itce.vn.ua/en

Information Technologies and Computer Engineering **Vol. 22, No. 3. 2025**

Article's History: Received: 22.08.2025 Revised: 17.11.2025 Accepted: 23.12.2025

UDC 004.415.5 DOI: 10.31649/vitce/3.2025.136

Fuzzy-algorithmic analysis of software reliability

Hanna Rakytyanska

PhD in Technical Sciences, Associate Professor
Vinnytsia National Technical University
21021, 95 Khmelnytske Shose Str., Vinnytsia, Ukraine
https://orcid.org/0000-0001-5863-3730

Bohdan Prus*

Postgraduate Student
Vinnytsia National Technical University
21021, 95 Khmelnytske Shose Str., Vinnytsia, Ukraine
https://orcid.org/0009-0008-7214-0949

Abstract. The relevance of the study was due to the need to develop interpretable process-oriented models that allow assessing the growth of the reliability function depending on the distribution of efforts. The aim of the work was to model the processes associated with introducing, detecting, and correcting errors using algorithmic algebra and fuzzy logic. The proposed methodology for software reliability analysis was based on the theory of reliability of algorithmic processes. The logical-algorithmic model of the development process was built on the basis of linear, alternative, and iterative operator structures. The sequence of works without feedback is described by the linear structure. The verification and validation stages were described using alternative and iterative algorithmic structures. The process of checking and correction, when detected errors were immediately removed, and new errors were not introduced, was described by the alternative structure. The debugging process, during which new errors might be introduced, was described by the iterative structure. The logical-algorithmic model in the form of the fuzzy knowledge base made it possible to design software with the required levels of reliability and cost using improving transformations. The system of fuzzy logical equations connected the correctness levels of the working, checking, and correction operations with the possibility of correct execution of the development process. The allocation of efforts was formalised by the improving substitutions introduced in the logical-algorithmic model. The controllable variables associated with improving substitutions were interpreted as the quality of execution of the working, checking, and correction operations. The proposed fuzzy model of software reliability allows to assess the risks of the development process based on expert and experimental information about the reliability and time characteristics of the life cycle stages. The model was constructed by transferring reliable parts of the development process obtained from the histories of errors and defects of previous projects into a process-oriented reliability model of the current project. The example of reliability analysis of the process of developing a mobile application for image aggregation was considered, where the influencing factors are the error-free execution of working, checking, and correction operations. The practical significance of the study lies in the development of a toolset that makes it possible to predict software reliability at different stages of the life cycle, to optimise the allocation of resources between error detection and correction, and to reduce the risks of unsuccessful decisions in design and debugging

Keywords: software reliability; risk assessment of the development process; effort allocation; logical-algorithmic model; fuzzy reliability model

Introduction

The main goal of software development is to provide the best quality product within a limited time (cost). Functional reliability is the main criterion that requires modelling and analysis at various stages of development. As shown in the study D. Hanagal & N. Bhalerao (2021), traditional reliability growth models predict software failures, but

Suggested Citation:
Rakytyanska, H., & Prus, B. (2025). Fuzzy-algorithmic analysis of software reliability. *Information Technologies and Computer Engineering*, 22(3), 136-147. doi: 10.31649/vitce/3.2025.136

*Corresponding author

do not allocate efforts to prevent them. Modelling events that lead to software failures requires the description of the software operation process and the definition of clusters of influential factors (Sokol, 2025). Software reliability analysis requires identifying the structure and parameters of a reliability model obtained through process validation to ensure compliance with requirements. At the same time, as V. Pradhan *et al*. (2023) note, developers need interpretable reliability models that give them insight into where to focus their efforts to reduce the risk of defects. Understanding the interdependence of events and processes is critical to software reliability. Real-world scenarios involve the risk of missing errors in the system because the processes of error detection and removal are imperfect. In the article M. Macak *et al*. (2022), process mining is defined as a set of methods aimed at extracting basic knowledge from processes to interpret data and understand the process in dynamics. The authors of research U. Samal *et al*. (2025) argued that the description of risks and uncertainties inherent in development processes determines the use of fuzzy models of software reliability.

As noted in the study C. Thieme *et al*. (2020), risk assessment is performed by embedding software functional failures and their propagation effects into traditional risk analysis models such as fault trees and event trees. Risk prediction is performed using defect density analysis based on the coordinate model combined with fault information. In practice, the use of combinatorial models such as fault trees or event trees is limited by the complexity of system behaviour. To manage the number of states for software components, Markov chains that model software behaviour using object-oriented modelling templates are proposed by R. Calinescu *et al*. (2021). The transition probabilities between different system states can be obtained from the operating environment. In the study V. Yakovyna & I. Symets (2021), a Markov model of software reliability is proposed that allows predicting the maximum number of operational states using the representation of a higher-order Markov process by an equivalent first-order process with additional virtual states. In a dynamic and uncertain environment, the entropy characterises the probability of transition to the next state of the system. U. Samal & A. Kumar (2024) considered software reliability models with detection and elimination of multiple failures that occur stochastically during testing stages. Reliability models of debugging stages are focused on evaluating the intensity of failures depending on the frequency of errors. In the study X. Chen & Y. Deng (2024), a software risk assessment model is proposed that can measure the uncertainty associated with stochastic testing and debugging processes by introducing belief entropy. Statistical modelling is aimed at identifying and ranking factors associated with the occurrence, detection, and elimination of errors. To consider risk factors, a conceptual model based on expert recommendations on project management is developed by B. Duarte *et al*. (2021). The conceptual model connects influencing factors and their interaction with on-time delivery, where deviations from the schedule are due to the errors in resource allocation. The work S. Butt *et al*. (2023) proposed an ontological model of the software testing process in the form of a knowledge graph, which establishes internal relationships between test cases and defects. Ontological analysis of software systems includes the construction of an ecosystem of software artifacts at different levels of abstraction. In the study H. Ferreira *et al*. (2023), a conceptual reliability models of software-intensive systems using machine learning and cloud technologies is proposed. The model consisting of dozens of concepts and connections between them is aimed at understanding processes and events at the stages of software verification and validation.

Constructing process-based software reliability models is computationally complex. The use of state-based models such as Markov chains is limited by the growth of the state space. The complexity of system behaviour limits the use of conceptual models based on the knowledge graph. At the same time, an approach aimed at transferring reliable components of the previous system to the new one from the failure history of existing systems can be a solution when constructing process-oriented reliability models. Reliable components are formalised as development stages related to error detection and correction. In this case, the model parameters represent a distribution of efforts that is easily interpreted by developers and can be changed according to requirements. The aim of the research was to develop the interpretable process-oriented models of software reliability based on the algebra of algorithms and fuzzy logic.

Materials and Methods

The proposed methodology for software reliability analysis is based on the theory of reliability of algorithmic processes (Rotshtein *et al*., 2007). The uncertainty associated with development stages is described using fuzzy logic (Rotshtein & Rakytyanska, 2012). The principles formulated in the aforementioned studies were used to model the reliability of the software development process. The description of events related to the introduction, detection, and removal of errors is carried out using V.M. Hlushkov's algebra of algorithms (Doroshenko *et al*., 2004). The logical-algorithmic model consists of operators and logical conditions as well as operator and logical structures that describe discrete software development processes. In the theory of reliability of algorithmic processes, a distinction between working operators and correction operators is made. Logical conditions formalise the operations of checking the correctness of the execution of working and modifying operators. Workflows are described using the linear algorithmic structures associated with the introduction of errors. The processes of error detection and elimination are described by the alternative and iterative algorithmic structures. The logical-algorithmic description of the development process is considered as an analogue of the fuzzy knowledge base. Then, the reliability model in the form of the system of fuzzy logical equations connects the possibilities of correct (incorrect) execution of the process and its elements

included in the logical-algorithmic description. Improving transformations embedded in the logical-algorithmic model ensured the increase in the reliability and cost of the development process. Controllable variables associated with improving transformations allowed generating variants of the development process with the required level of quality under time constraints. Software for software reliability analysis was developed in Python.

Logical-algorithmic models of the development process

The software development process was described in the system of V.M. Hlushkov's algorithmic algebras using the typical operator and logical structures (Doroshenko *et al.*, 2004). The linear structure described a sequence of works without checking operations and feedback. Checking operations corresponded to the stages of verification and validation aimed at detecting errors. Correction operations corresponded to the debugging stages aimed at removing errors. The alternative structure described the branched process of "work – checking (testing) – correction without feedback", when errors were detected and immediately removed from the system; the iterative structure described the cyclical process of "work – checking (testing) – correction with feedback", when new errors can be introduced during debugging. The process of developing a null version of software with sequential implementation of stages was described by a linear algorithmic structure of the form:

$$L_0 = A_1 A_2 \ldots A_n, \tag{1}$$

where A_i is the working operator corresponding to the i-th development stage; $i = 1,\ldots, n$; L_0 is the equivalent operator corresponding to the linear structure of the null version.

The software development process is associated with the introduction of errors of the j-type, $j = 1,\ldots, m$, which are subject to detection and removal at the stages of verification and validation. It was assumed that $e_i = \{e_{i1},\ldots, e_{ik_i}\}$ be the set of errors introduced during the execution of the working operator A_i, $k_1 + \cdots + k_n = m$. The processes of verification and validation of the working operator A_i with the detection and removal of errors e_i were described by the following algorithmic structures (Rotshtein *et al.*, 2007; Rotshtein & Rakytyanska, 2012):

- alternative algorithmic structure "work – checking – correction without feedback"

$$B_i = A_i[e_i] \underset{\omega_i}{(} D \vee U_i); \tag{2}$$

- iterative algorithmic structure "work – checking – correction with feedback"

$$C_i = A_i[e_i] \underset{\omega_i}{\{} R_i\}, \tag{3}$$

where ω_i is the logical condition for checking the correctness of the execution of the working operator A_i, $\omega_i = 1(0)$ if the operator A_i is performed correctly (incorrectly); D is the identical operator which is interpreted as fixing the results of checking; U_i is the correction operator during the execution of which detected errors are immediately removed and new errors are not introduced; R_i is the correction operator during the execution of which new errors may be introduced; B_i C_i are the equivalent operators corresponding to the alternative and iterative algorithmic structures.

Software quality management was carried out with the help of improving substitutions introduced into the logical-algorithmic model. Improving substitutions aimed at increasing software reliability are interpreted as the quality of performing working, checking, and correction operations (Rotshtein & Rakytyanska, 2012). Improving substitutions model the allocation of efforts to reduce the risk of introducing errors during the development stage; reducing the risk of missing errors at the testing stage; reducing the risk of leaving errors in the system or introducing new errors at the debugging stage. The distribution of efforts was formalised by adding the following improving substitutions into the logical-algorithmic models (1)-(3):

$$L = (A_1)^{\boldsymbol{X}_1}(A_2)^{\boldsymbol{X}_2}\ldots(A_n)^{\boldsymbol{X}_n}; \tag{4}$$

$$P_i = (A_i)^{\boldsymbol{X}_i} \underset{(\omega_i)^{\boldsymbol{Y}_i}}{(} D \vee (U_i)^{\boldsymbol{Z}_i}); \tag{5}$$

$$Q_i = (A_i)^{\boldsymbol{X}_i} \underset{(\omega_i)^{\boldsymbol{Y}_i}}{\{} (R_i)^{\boldsymbol{Z}_i^*}\}, \tag{6}$$

where $X_i = (x_{i1}, x_{i2},\ldots, x_{ik_i})$ is the vector of parameters that determine the quality of execution of the working operator (A_i) to reduce the risk of introducing the error e_{ij}; $Y_i = (y_{i1}, y_{i2},\ldots, y_{ik_i})$ is the vector of parameters that determine the quality of execution of the checking operator (ω_i) to reduce the risk of missing the error e_{ij} at the stages of verification and validation; $Z_i = (z_{i1}, z_{i2},\ldots, z_{ik_i})$ and $Z_i^* = (z_{i1}^*, z_{i2}^*,\ldots, z_{ik_i}^*)$ are the vectors of parameters that determine the quality of execution of the correction operators (U_i and R_i) to reduce the risk that the error e_{ij} will not be corrected and will remain in the system and new errors will be introduced. The parameters x_{ij}, y_{ij}, z_{ij}, $z_{ij}^* \in \{1, 3, 5, 7, 9\}$, which are chosen in accordance with Saaty's scale (Saaty, 1994; Rotshtein & Rakytyanska, 2012), are interpreted as severity ranks of the errors e_{ij} at the working stages and priority ranks at the stages of checking and correction.

The process of software development, testing, and debugging was described by an equivalent algorithmic structure obtained by replacing sections of the initial algorithm (1) with the improving substitutions (4)-(6). The rules of improving transformations (4)-(6) made it possible to represent the generation of variants of the logical-algorithmic model as inference in a formal grammar (Rotshtein *et al.*, 2007):

$$G = \langle V_t, V_n, L_0, I\rangle, \tag{7}$$

where V_t is the set of operator and logical functional units (terminals), $\{A_i, U_i, R_i\}$ and $\{\omega_i\}$, $i = 1,\ldots, n$; V_n is the set of operator functional structures (nonterminals), $\{B_i, C_i\}$; L_0 is the null version of the logical-algorithmic model; I is the

set of improving substitutions: $\{(A_i)^{X_i}, (U_i)^{Z_i}, (R_i)^{Z_i}\}$ for the working and correction operators; $\{(\omega_i)^{Y_i}\}$ for the logical conditions; $\{L, P_i, Q_i\}$ for the operator structures.

Thus, by selecting improving substitutions, the formal description of the development process was synthesised that ensures acceptable levels of software faultlessness and development time. The functional network corresponding to the null version of the development process is transformed until a logical-algorithmic description is found that satisfies the reliability and cost requirements. The sequential algorithmic structure was chosen as a null option. Each improving substitution meant replacing some operator or logical fragment of the functional network by another fragment with an increased level of reliability at the expense of additional costs.

Fuzzy model of software reliability

The level of correctness of the software development process was assessed using the system of fuzzy rules. An analogue of the fuzzy knowledge base and a carrier of the reliability model is the logical-algorithmic description of events associated with the occurrence, detection, and elimination of the causes of incorrect operation of the software system. The inputs of the process-oriented model are the reliability-time estimates of working, checking, and correction operators. At the output of the process, two classes of situations corresponding to correct (μ^1) and incorrect ($\mu^0 = 1 - \mu^1$) execution of the task are identified. Then, for the sequential algorithmic structure (4) with improving transformations (5), (6), the correctness of completion of the stage A_i, $i = 1,\ldots, n$, is described by a system of fuzzy rules:

IF the working operator A_i is executed correctly

OR errors are correctly detected and removed during checking ω_i and U_i correction ,

AND no new errors have been made,

OR errors are correctly detected and resolved during correction R_i.

(with the possibility of introducing new errors)

THEN the stage is completed correctly.

The following system of fuzzy logical equations that connects the levels of fuzzy correctness of operators and logical conditions with the classes of decisions corresponding to correct (μ^1) and incorrect ($\mu^0 = 1 - \mu^1$) execution of the task is derived from the fuzzy knowledge base:
for the alternative structure

$$\mu_{P_i}^1 = \mu_{A_i}^1(X_i) \cdot \mu_{\omega_i}^1(Y_i) + \mu_{A_i}^0(X_i) \cdot \mu_{U_i}^1(Z_i); \quad (8)$$

for the iterative structure

$$\begin{aligned}\mu_{Q_i}^1 = {} & \mu_{A_i}^1(X_i) \cdot \mu_{\omega_i}^1(Y_i) + \mu_{A_i}^0(X_i) \cdot \mu_{R_i}^1(Z_i^*) + \\ & + \mu_{A_i}^{00} \cdot [\mu_{R_i}^{00} + \mu_{R_i}^{10}] \cdot \mu_{R_i}^1(Z_i^*) + \cdots \\ & + [\mu_{A_i}^{00}]^q \cdot [\mu_{R_i}^{00} + \mu_{R_i}^{10}]^q \cdot \mu_{R_i}^1(Z_i^*), \end{aligned} \quad (9)$$

where the risk of repeated correction is

$$\mu_{A_i}^{00} = 1 - [\mu_{A_i}^1(X_i) \cdot \mu_{\omega_i}^1(Y_i) + \mu_{A_i}^0(X_i) \cdot \mu_{R_i}^1(Z_i^*)], \quad (10)$$

where $\mu_{A_i}^1$ ($\mu_{A_i}^0$), $\mu_{\omega_i}^1$ ($\mu_{\omega_i}^0$), and $\mu_{U_i}^1$ ($\mu_{U_i}^0$) are the possibilities of correct (incorrect) execution of the working (A_i), checking (ω_i), and correction (U_i) operators; $\mu_{R_i}^1(\mu_{R_i}^{00}, \mu_{R_i}^{10})$ is the distribution of fuzzy correctness of the cyclic correction operator R_i; q is the number of verification and correction cycles for the iterative structure; $\mu_{P_i}^1$ ($\mu_{P_i}^0$) and $\mu_{Q_i}^1$ ($\mu_{Q_i}^0$) is the possibility of correct (incorrect) execution of the equivalent operators P_i and Q_i.

The reliability of working, checking, and correcting operators was modelled on the basis of expert and experimental data on the distribution of errors of various types (interface errors, errors in the logic of the program, errors in the processing of parallel data flows, etc.). Following the m-ary concept of errors (Rotshtein *et al.*, 2007), the possibility of identifying and correcting errors of the j-th type, $j = 1,\ldots, k_i$, was considered at the i-th stage. Managing the risks of introducing and omitting errors when executing working, checking, and correction operators was carried out using controllable variables as follows:

$$\begin{aligned}\mu_{A_i}^1(X_i) &= \prod_{j=1}^{k_i}\left(1 - [\mu_{A_{ij}}^0]^{x_{ij}}\right), \\ \mu_{\omega_i}^1(Y_i) &= \prod_{j=1}^{k_i}\left(1 - [\mu_{\omega_{ij}}^0]^{y_{ij}}\right), \\ \mu_{U_i}^1(Z_i) &= \prod_{j=1}^{k_i}\left(1 - [\mu_{U_{ij}}^0]^{z_{ij}}\right),\end{aligned} \quad (11)$$

$$\mu_{R_i}^1(Z_i^*) = \prod_{j=1}^{k_i}\left(1 - [\mu_{R_{ij}}^{00} + \mu_{R_{ij}}^{10}]^{z_{ij}^*}\right), \quad (12)$$

where $\mu_{A_i}^1$ ($\mu_{A_i}^{0L}$) is the possibility of avoiding (introducing) the error e_{ij} when executing the working operator A_i; $\mu_{\omega ij}^1(\mu_{\omega ij}^0)$ is the possibility of detecting (missing) the error e_{ij} when checking the truth of the logical condition ω_i; $\mu_{Uij}^1(\mu_{Uij}^0)$ is the possibility of removing (leaving in the system) the error e_{ij} when executing the correction operator U_i; $\mu_{Rij}^1(\mu_{Rij}^{00}, \mu_{Rij}^{10})$ is the possibility of removing (leaving in the system) the error e_{ij} and introducing new errors when executing the correction operator R_i.

Fuzzy correctness of the multistage development process described by a sequence of the alternative and iterative algorithmic structures was defines as follows:

$$\mu^1 = \prod_{i=1}^{n}(\mu_{S_i}^1(X_i, Y_i, Z_i, Z_i^*)), S_i \in \{P_i, Q_i\}. \quad (13)$$

The execution time of the working (t_{A_i}), checking (t_{ω_i}), and correction (t_{U_i}, t_{R_i}) operations was calculated in proportion to the development time of the null version of the working operator t_{Ai}^0 as follows:

$$\begin{aligned}t_{A_i}(X_i) &= (1 + \frac{\max_j(x_{ij})}{x_{max}})t_{A_i}^0, t_{\omega_i}(Y_i) = \frac{\max_j(y_{ij})}{y_{max}}t_{A_i}^0, \\ t_{U_i(R_i)}(Z_i) &= \frac{\max_j(z_{ij})}{z_{max}}t_{A_i}^0,\end{aligned} \quad (14)$$

where the maximum quality ratings are $x_{max}, y_{max}, z_{max} = 9$.

The execution time of the algorithms (4)-(6) was estimated as follows:
for the alternative structure

$$t_P^* = \sum_{i=1}^{n}(t_{A_i}(X_i) + t_{\omega_i}(Y_i) + t_{U_i}(Z_i)); \quad (15)$$

for the iterative structure

$$t_Q^* = \sum_{i=1}^{n}(t_{A_i}(X_i) + q(t_{\omega_i}(Y_i) + t_{R_i}(Z_i^*))), \quad (16)$$

where t_P^*, t_Q^* is the execution time of the equivalent operators corresponding to the alternative and iterative structures.

Thus, correlations (8)-(13) define the fuzzy model of software reliability growth. The increase in software reliability was achieved by managing risks of incorrect performance of the task. Parameters of working, checking, and correction operators determine the efforts aimed at reducing the risks of errors at all stages of development. The growth of the reliability function is ensured by introducing the quality indicators of working, checking, and correction operations in accordance with the Saaty's scale. Fuzzy logical equations model the increase in the level of fuzzy correctness depending on the distributed efforts. The development time determined by correlations (15), (16) is calculated in accordance with the priority ranks of working, checking, and correction operators.

Results and Discussion

Example: Risk assessment of the mobile application development.

The problem of reliability analysis of a mobile application for image aggregation is considered (Rakytyanska & Prus 2024). It is assumed that a logical-algorithmic model of the development process is given, where risk assessments of the working, checking, and correction operations can be obtained on the basis of already completed projects. The problem of reliability analysis was formulated as follows. For the given logical-algorithmic model, it is necessary to minimise risks of introducing, missing, and leaving errors in the system by distributing efforts to perform working, checking, and correction operations. The development time should not exceed 16 weeks.

The logical-algorithmic model of the development process looks as follows:

$$A^* = A_0\,[e_0]_{\omega_0}(D \vee U_0)\ldots A_2\,[e_2]_{\omega_2}(D \vee U_2)\,A_3\,[e_3]_{\omega_3}\{R_3\}\,A_4\,[e_4]_{\omega_4}(D \vee U_4)$$
$$A_5\,[e_5]_{\omega_5}(D \vee U_5)\,A_6\,[e_6]_{\omega_6}\{R_6\}\ldots A_{15}\,[e_{15}]_{\omega_{15}}\{R_{15}\}$$
$$A_{16}\,[e_{16}]_{\omega_{16}}(D \vee U_{16})\ldots A_{19}\,[e_{19}]_{\omega_{19}}(D \vee U_{19}). \quad (17)$$

Here A_0 – requirements analysis; $e_0 = \{e_{0.1}\}$, where $e_{0.1}$ – requirements incorrectly defined by the customer;

A_1 – development of technical specifications for software design; $e_1 = \{e_{1.1},\ldots, e_{1.5}\}$, where $e_{1.1}$ – incorrect formulation of technical requirements; $e_{1.2}$ – lack of clear software goals; $e_{1.3}$ – improper distribution of roles and responsibilities; $e_{1.4}$ – changing requirements; $e_{1.5}$ – incomplete detailing of the project;

A_2 – development of technical specifications for programming; $e_2 = \{e_{2.1},\ldots, e_{2.4}\}$, where $e_{2.1}$ – misunderstanding of technical aspects; $e_{2.2}$ – inconsistency of resources and deadlines; $e_{2.3}$ – lack of quality control; $e_{2.4}$ – failure to consider risks;

A_3 – development of UI/UX design; $e_3 = \{e_{3.1},\ldots, e_{3.7}\}$, where $e_{3.1}$ – inconsistency with user needs; $e_{3.2}$ – complex and confusing interface; $e_{3.3}$ – poor readability and contrast; $e_{3.4}$ – non-optimised images; $e_{3.5}$ – lack of processing of user paths; $e_{3.6}$ – ignoring updates and trends; $e_{3.7}$ – non-adaptive design;

A_4 – analysis of implementation methods; $e_4 = \{e_{4.1}, e_{4.2}\}$, where $e_{4.1}$ – neglect of resources; $e_{4.2}$ – neglecting expansion needs;

A_5 – analysis of architectural solutions; $e_5 = \{e_{5.1}, e_{5.2}\}$, where $_{5.1}$ – disregarding software functionality; $e_{5.2}$ – neglecting expansion needs;

A_6 – software architecture development; $e_6 = \{e_{6.1},\ldots, e_{6.3}\}$, where $e_{6.1}$ – errors in the analysis of architectural solutions; $e_{6.2}$ – suboptimal performance; $e_{6.3}$ – dissatisfaction with security needs;

A_7 – interface development; $e_7 = \{e_{7.1},\ldots, e_{7.5}\}$, where $e_{7.1}$ – design pattern mismatch; $e_{7.2}$ – using the wrong style elements; $e_{7.3}$ – inappropriate animations; $e_{7.4}$ – non-adaptive interface; $e_{7.5}$ – errors in the text;

A_8 – development of software modules; $e_8 = \{e_{8.1},\ldots, e_{8.4}\}$, where $e_{8.1}$ – errors in the program code; $e_{8.2}$ – third-party library errors; $e_{8.3}$ – errors in the work algorithms of software modules; $e_{8.4}$ – unoptimised code;

A_9 – development of image processing module; $e_9 = \{e_{9.1},\ldots, e_{9.4}\}$, where $e_{9.1}$ – processing algorithm errors; $e_{9.2}$ – internal library error; $e_{9.3}$ – hardware processing algorithms are not supported; $e_{9.4}$ – overloading of the computing resources of the device;

A_{10} – database development; $e_{10} = \{e_{10.1},\ldots, e_{10.3}\}$, where $e_{10.1}$ – database normalisation is broken; $e_{10.2}$ – wrong data types; $e_{10.3}$ – incorrect types of relationships between tables;

A_{11} – connecting logic to the interface; $e_{11} = \{e_{11.1}, e_{11.2}\}$, where $e_{11.1}$ – inconsistency of the interface with the program logic; $e_{11.2}$ – interface design errors;

A_{12} – software testing; $e_{12} = \{e_{12.1},\ldots, e_{12.5}\}$, where $e_{12.1}$ – using the wrong user flow during testing; $e_{12.2}$ – there is no verification of software operation on different devices; $e_{12.3}$ – no smoke testing; $e_{12.4}$ – there is no verification of the problem in several approaches; $e_{12.5}$ – no testing with poor or no internet connection;

A_{13} – error correction; $e_{13} = \{e_{13.1},\ldots, e_{13.4}\}$, where $e_{13.1}$ – incorrectly formulated ways of reproducing the error; $e_{13.2}$ – not following all paths to reproduce the error; $e_{13.3}$ – no playback error with correct playback paths; $e_{13.4}$ – there is no verification of the entire software module when the software code changes in it;

A_{14} – checking the requirements of the technical task; $e_{14} = \{e_{14.1}, e_{14.2}\}$, where $e_{14.1}$ – inconsistency of the current design with the requirements; $e_{14.2}$ – non-compliance of the software functionality with the requirements of the technical task;

A_{15} – verification of the software product by the customer; $e_{15} = \{e_{15.1}, e_{15.2}\}$, where $e_{15.1}$ – non-compliance of the software with the primary requirements; $e_{15.2}$ – non-compliance of the interface with the primary requirements;

A_{16} – preparation for publication; $e_{16} = \{e_{16.1},\ldots, e_{16.4}\}$, where $e_{16.1}$ – there is no full testing of the software before publication; $e_{16.2}$ – uncorrected critical software errors;

$e_{16.3}$ – errors in the description of the user terms of service provision; $e_{16.4}$ – there is no testing of alpha and beta versions of the software;

A_{17} – checking requirements before publishing to AppStore, Google Play; $e_{17} = \{e_{17.1},..., e_{17.4}\}$, where $e_{17.1}$ – non-compliance with the requirements of application stores; $e_{17.2}$ – misrepresentation of user data; $e_{17.3}$ – non-compliance with the requirements to be effective after publication; $e_{17.4}$ – non-compliance with country-specific publication requirements;

A_{18} – publication; $e_{18} = \{e_{18.1}, e_{18.2}\}$, where $e_{18.1}$ – error when uploading the application assembly to the application store; $e_{18.2}$ – publication of the application without taking into account analytical data;

A_{19} – support; $e_{19} = \{e_{19.1},..., e_{19.3}\}$, where $e_{19.1}$ – no resources for support; $e_{19.2}$ – critical bug fixes are missing; $e_{19.3}$ – no support for new platform requirements.

In (16), the stages $A_0 - A_2, A_4, A_5, A_{16} - A_{19}$ are described by the alternative algorithmic structures without feedback, and the stages A_3, $A_{16} - A_{19}$ are described by the iterative algorithmic structures with feedback. Risk assessments for the working, checking, and correction operators are given in Table 1. Error distribution ranges were obtained on the basis of expert and experimental data. To estimate the fuzzy correctness of the operators and operator structures in (16), the histories of errors and defects of 56 already completed projects in the field of mobile application development were considered. The reliability of working operators ($\mu_{A_i}^1$) is determined based on fuzzy estimates of the frequency of introducing errors of various types. The reliability of checking ($\mu_{\omega i}^1$) and correction (μ_{Ui}^1, μ_{Ri}^1) operators is determined based on fuzzy estimates of the frequency of detection and correction of errors of various types. Given the fuzzy correctness (μ^1), the risk ($\mu^0 = 1 - \mu^1$) is calculated.

Table 1. Risk assessments for the working, checking, and correction operators

Stage	Working operator	Checking operator	Correction without feedback	Correction with feedback	
A_i	μ_{Aij}^0	$\mu_{\omega ij}^0$	μ_{Uij}^0	μ_{Rij}^{00}	μ_{Rij}^{10}
A_0	0.392	0.464	0.214	-	-
A_1	0.115-0.428	0.185-0.396	0.120-0.275	-	-
A_2	0.196-0.410	0.142-0.339	0.107-0.250	-	-
A_3	0.250-0.375	0.160-0.285	-	0.071-0.160	0.053-0.107
A_4	0.160-0.267	0.178-0.250	0.089-0.178	-	-
A_5	0.214-0.357	0.196-0.304	0.185-0.214	-	-
A_6	0.304-0.410	0.214-0.392	-	0.089-0.178	0.071-0.142
A_7	0.232-0.446	0.285-0.464	-	0.107-0.267	0.053-0.125
A_8	0.125-0.392	0.196-0.428	-	0.160-0.250	0.125-0.178
A_9	0.214-0.464	0.125-0.375	-	0.142-0.232	0.107-0.160
A_{10}	0.107-0.250	0.196-0.267	-	0.089-0.214	0.071-0.125
A_{11}	0.285-0.392	0.178-0.304	-	0.178-0.232	0.089-0.160
A_{12}	0.196-0.428	0.214-0.339	-	0.160-0.304	0.107-0.142
A_{13}	0.304-0.446	0.267-0.410	-	0.125-0.160	0.071-0.125
A_{14}	0.107-0.160	0.089-0.125	-	0.071-0.089	0.053-0.071
A_{15}	0.125-0.178	0.107-0.160	-	0.107-0.214	0.053-0.089
A_{16}	0.089-0.232	0.125-0.178	0.071-0.160	-	-
A_{17}	0.196-0.267	0.160-0.214	0.107-0.178	-	-
A_{18}	0.160-0.214	0.142-0.196	0.053-0.071	-	-
A_{19}	0.107-0.178	0.125-0.214	0.089-0.142	-	-

Source: created by the authors

When calculating the reliability of algorithmic structures using formulas (8), (9), the risk of incorrect completion of development stages for preliminary risk assessments is 10-25%. This means that without proper resource allocation, the possibility of cascading and interactive effects as the consequences of errors propagation steadily increases. As a result, the goals of the project cannot be achieved within the established time frame. Assessments of the correctness of software development stages after efforts distribution are given in Table 2. Risk management is carried out based on formulas (11), (12) by dividing efforts (x_{ij}, y_{ij}, z_{ij}) to reduce the risks of introduction (μ_{Aij}^0), imperfect detection ($\mu_{\omega ij}^0$), and incomplete correction (μ_{Uij}^0, μ_{Rij}^0) of the set of errors $e_i = \{e_{ij}\}$, $j = 1,..., k_i$. The result of the effort distribution is a stable growth of the reliability function due to the correct execution of the working (μ_{Ai}^1), checking ($\mu_{\omega i}^1$), and correction (μ_{Ui}^1, μ_{Ri}^1) operators. Estimates of the correct completion of each stage (μ_{Pi}^1, μ_{Qi}^1) associated with correctness of the execution of the alternative or iterative structure, are obtained using formulas (8), (9).

Table 2. Fuzzy correctness of development stages after distribution of efforts among working, checking, and correction operators

Stage	Working operator		Checking operator		Correction without feedback			Correction with feedback		
A_i	x_{ij}	$\mu^1_{A_i}$	y_{ij}	$\mu^1_{\omega_i}$	z_{ij}	$\mu^1_{U_i}$	$\mu^1_{P_i}$	z^*_{ij}	$\mu^1_{R_i}$	$\mu^1_{Q_i}$
A_0	5	0.940	7	0.995	7	0.998	0.996	-	-	-
A_1	5-7	0.986	7-8	0.998	6-8	0.999	0.998	-	-	-
A_2	4-5	0.972	6-7	0.998	6-7	0.997	0.996	-	-	-
A_3	4-5	0.980	5-6	0.998	-	-	-	5-6	0.997	0.995
A_4	3-4	0.981	4-5	0.997	3-4	0.998	0.996	-	-	-
A_5	4-5	0.984	5-6	0.997	5-6	0.995	0.997	-	-	-
A_6	5-6	0.978	6-7	0.996	-	-	-	6-7	0.997	0.996
A_7	5-6	0.963	5-7	0.996	-	-	-	7-8	0.999	0.996
A_8	4-6	0.974	6-7	0.997	-	-	-	7-8	0.996	0.995
A_9	4-5	0.950	5-6	0.996	-	-	-	6-7	0.995	0.992
A_{10}	3-4	0.984	4-5	0.998	-	-	-	5-6	0.999	0.997
A_{11}	3-4	0.973	4-5	0.995	-	-	-	4-5	0.996	0.995
A_{12}	5-6	0.936	6-8	0.998	-	-	-	7-8	0.996	0.994
A_{13}	5-6	0.990	6-7	0.996	-	-	-	7-8	0.998	0.996
A_{14}	3-4	0.982	4-5	0.998	-	-	-	4-5	0.997	0.996
A_{15}	3-4	0.990	4-5	0.997	-	-	-	4-5	0.998	0.995
A_{16}	2-3	0.947	2-3	0.995	2-3	0.996	0.995	-	-	-
A_{17}	2-3	0.981	3-4	0.998	2-3	0.994	0.997	-	-	-
A_{18}	2-3	0.954	3-4	0.997	3-4	0.998	0.996	-	-	-
A_{19}	2-3	0.968	4-3	0.998	2-3	0.997	0.995	-	-	-

Source: created by the authors

The obtained results demonstrate the correct ranking of risks during the performance of working, verification, and correction operations. The possibility of correct completion of the project is estimated using formula (13). Risk assessments during the analysis of requirements and development of technical specifications at the stages A_0-A_2 amount to approximately 1%. Risk assessments during the analysis of user's design, implementation methods, and architectural solutions at the stages A_3-A_5 are less than 1.5%. Risk assessments during the development and testing at the stages A_6-A_{15} are around 5%. Risk assessments in preparation for publication and support at the stages A_{16}-A_{19} are less than 2%. Reducing the risk of incorrect completion of the previous stage A_{i-1} allows avoiding the effects of cascading and interaction when spreading the consequences of errors $e_i = \{e_{ij}\}, j = 1,\ldots, k_i$, at the stages $A_i, A_{i+1},\ldots, A_n$.

The schedule for the execution of the sequential discrete process (16) is shown in Figure 1. The calendar plan was calculated using formulas (14), (15) taking into account the costs of checking and correction in accordance with the distribution of efforts. To comply with time constraints, the start of each stage was chosen as early as possible. The development of technical specifications at the stages A_0-A_2 takes 1-2 weeks. The development of user's design and architectural solutions at the stages A_3-A_5 takes approximately 4 weeks. The duration of the development and testing stages A_6-A_{15} is around 12 weeks. The final stages of preparation for publication A_{16}-A_{19} take no more than 2 weeks. When calculating the calendar plan, contextual histories of already completed projects were used, where the duration of the development and testing stages makes up 80% of the project time. The time spent on detecting and correcting errors was estimated in proportion to the development time of the null version of the stage, depending on the priority of the errors. As a result of proper distribution of efforts, the software development period does not exceed 16 weeks, which guarantees timely delivery.

Within the given schedule, the progress of the project in the form of the cumulative flow diagram can be predicted using the system of fuzzy rules and assessments of correct execution of the operators and logical conditions (Tables 1, 2). The number of tasks in development and testing is known for each week of the calendar plan. The decision to accept the task (or return it for revision) is made based on the possibility of correct execution of the working and checking operators $\mu^1_{A_i}$, $\mu^1_{\omega_i}$, The decision about the successful revision is made based on the possibility of correct execution of the debugging operators $\mu^1_{U_i}$, $\mu^1_{R_i}$. The decision about the successful completion of the stage is made based on the possibility of correct execution of the alternative or iterative structure $\mu^1_{P_i}$, $\mu^1_{Q_i}$ that allows to estimate the number of completed tasks and the number of tasks in the queue. The experimental and model cumulative flow diagrams are shown in Figure 2.

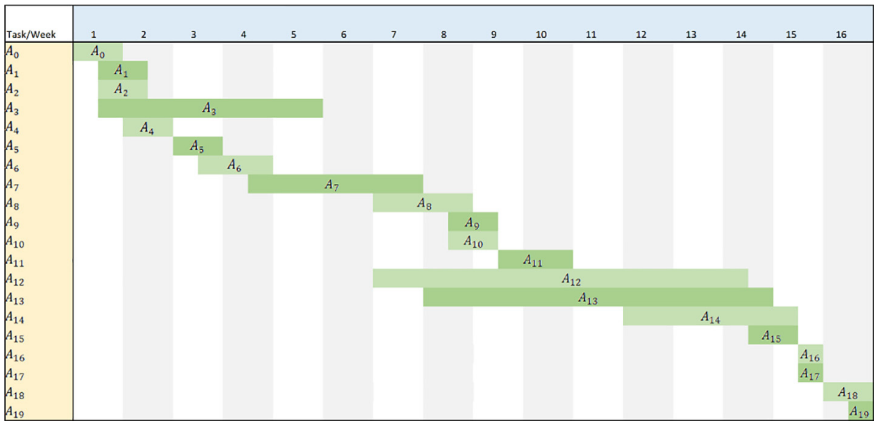

Figure 1. Development process schedule

Source: created by the authors

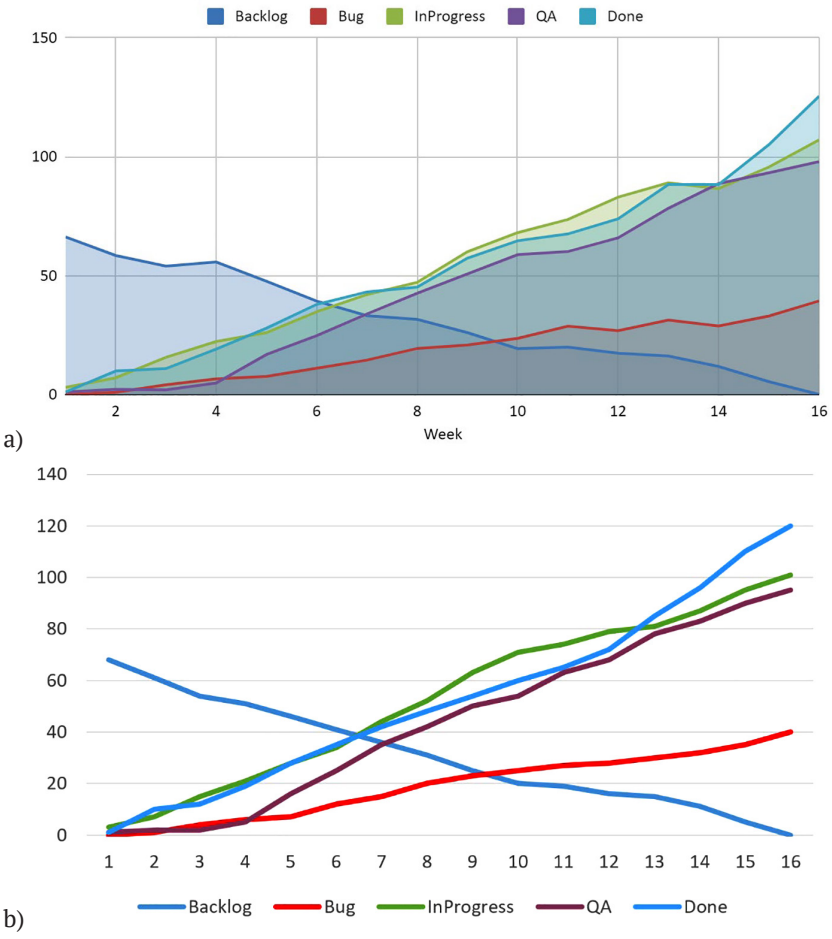

Figure 2. Experimental (a) and model (b) cumulative flow diagram

Source: created by the authors

Comparison of the experimental and model dynamics of the project indicates stable software quality management. As a result of the correct distribution of efforts, the number of completed tasks is steadily increasing; the number of tasks in development and testing remains stable; the number of tasks in the queue is steadily decreasing. In practice, dynamics of a real project allows for short-term horizontal sections indicating the resolution of problems. In case of periods of decline, the model can predict the recovery of the chart. Thus, the proposed model demonstrates the ability to correctly approximate the progress of real projects under conditions of risks and uncertainty.

Software developed in Python for software reliability analysis is an intelligent system that implements a

logical-algorithmic model of the development process. The system is integrated with the analytical module by forming fuzzy rules that take into account the possibility of occurring, detecting, and correcting errors at different stages of the software life cycle. The system allows modelling the processes "work – checking – correction" using improving substitutions and assessing the level of correctness based on a system of fuzzy logical equations. The controllable variables embedded in the logical-algorithmic model formalise the distribution of efforts according to task priority. Visualisation tools make it possible to analyse the dynamics of the project depending on the distributed efforts. For practical application, the model is integrated with repositories of error and defect histories, automatically adapting the parameters of fuzzy rules based on empirical data from previous projects.

Discussion of the results of evaluating the effectiveness of the software reliability model

This paper proposes a software reliability model that allows generating improving transformations of the development process in the form of linguistic rules to prevent the risks of software defects. Such rules are the carrier of the software quality management model, as they give practitioners the opportunity to distribute efforts under limited time conditions. The logical-algorithmic model of software reliability is obtained on the basis of an intelligent analysis of software development processes. The process-oriented model predicts the risks of software defects based on assessments of the correct execution of the process elements, such as the stages of development, testing, and debugging. The principal difference is the integration of the controllable variables into the logical-algorithmic description of the software life circle that allows managing the development risks. Improving substitutions allow to simulate events that ensure software reliability growth at the development stages. As a result, the proposed approach, which is similar to knowledge distillation, allows transferring reliable parts of the previous projects into a process-oriented reliability model of the current project.

Distillation-based models of software reliability aim to define action plans based on software analytics and practitioner findings. In the study B. Littlewood *et al*. (2020), failure rate data of existing software is used as a prior distribution when assessing the reliability of a new system. This approach ensures transferring reliable components of the previous system to the new one from the history of failures of existing systems. The article A. Filippetto *et al*. (2021) consider a database of failures and defects of already completed projects as a context history. The failure history of existing software is used to assess the reliability of a new system by modelling scenarios using distance-based similarity measures. In the work M. Asif & J. Ahmed (2020), a decision support system is developed that automatically generates rules to reduce software risks based on frequent failure patterns. A rule-based machine learning approach is used to establish relationships between risk factors and software reliability, where the previous failure cases and the corresponding action plan are associated with rules. N. Alnahdi & R. Alnanih (2024) consider an information model that integrates usability testing and reliability analysis at the stage of interface design to enhance the system performance based on user experience. The conceptual model is adjusted to identify risks using expert recommendations for minimising risks observed during the implementation of similar projects. The work D. Rajapaksha *et al*. (2022) proposes a defect prediction model for the automatic development of planning strategies using a qualitative study and empirical assessment of the impact factors determined by the software development methodology. Current planning actions are generated in the form of rules-based explanations and associated risk thresholds. The rule-based model proposed by T. Hovorushchenko (2021) predicts the risks of software defects based on many factors obtained from the analysis of experimental data. The risk management model generates planning strategies directly in the form of expert recommendations. R. Ouriques *et al*. (2023) use the approach based on the grounded theory to generate explanations regarding the compliance of the current state of the project with the requirements. The intelligent process analysis allows reconstructing the sequence of events that can cause software failure. As noted in the study J. Díaz *et al*. (2023), modelling condition-event relationships using the grounded theory requires transparency and replicability, which improve the trustworthiness of the generated recommendations. Qualitative data analysis in empirical software research takes into account the perspectives of a group of experts to structure codified knowledge based on a consistent interpretation of events.

Unlike process-based models that examine clusters of factors influencing condition-event relationships, the proposed model uses a time series approach. Correction of the execution of the multidimensional discrete process with the m-ary concept of errors is carried out by selecting improving substitutions. The number of model parameters is reduced to the number of error types k_i at each stage A_i, $i = 1,\ldots, n$. Then, the following groups of risks are considered: risks of introducing errors during development; risks of missing errors during testing; risks of leaving errors in the system or introducing new errors due to imperfect debugging. In particular, to assess the risks of the mobile application development process, from 2 to 7 risk factors or types of errors were considered at each stage of the life cycle $A_0, \ldots, A_{19}$. Therefore, in order to manage the quality of development, it is sufficient to distribute efforts for the correct completion of the stages, where the processes of introducing, detecting, and removing of errors are described by the algorithmic structures "work – checking – correction". Improving transformations formalised by the controllable variables are associated with the thoroughness of development due to the increase of the working time or the skills of developers. The application of the proposed model is limited to discrete algorithmic processes, where risk management is carried out within the time frame of each stage.

Conclusions

This article proposes an approach to modelling the reliability of the software development process based on the algebra of algorithms and fuzzy logic. The multidimensional discrete process of the software development is described using the modified system of V.M. Hlushkov's algorithmic algebras. The algorithmic description of events related to the introduction, detection, and removal of errors is considered as the fuzzy knowledge base "work – checking – correction". The linear structure describes a sequence of works without feedback. The alternative structure describes the process of testing and correction where errors are detected and immediately removed from the system. The iterative structure describes the debugging process with the possibility of introducing new errors. The logical-algorithmic model makes it possible to develop algorithms with the required levels of correctness and cost based on expert and experimental reliability assessments obtained at the stages of the software life cycle.

A fuzzy model of software reliability growth is proposed. The reliability model in the form of the system of fuzzy logical equations connects the possibility of correct (incorrect) execution of the process and the assessments of correctness of the working, checking, and correction operators. Software reliability analysis is associated with assessing the risks that arise during development, verification, and validation due to non-compliance with design requirements. Risk management is carried out with the help of improving substitutions embedded into the logical-algorithmic model. Improving substitutions allow modelling the distribution of efforts to reduce the risk of introducing errors during the development stage; the risk of missing errors at the testing stage; the risk of leaving errors in the system or introducing new errors at the debugging stage. The growth of the reliability function and the progress of the project are ensured by the introduction of indicators of the quality of execution of operators and logical conditions in accordance with Saaty's scale. The synthesis of the fuzzy knowledge base that ensures acceptable levels of software risks and development time is carried out by selecting controllable variables associated with improving substitutions.

Further research involves training the reliability model on experimental data in the form of histories of errors and defects. This approach consists in building and training membership functions of fuzzy correctness for the operators and logical conditions as well as rule weights for the algorithmic structures. A model trained by transferring reliable elements of the development process from the training dataset to the logic-algorithmic model will make it possible to predict the project dynamics depending on the distributed efforts.

Acknowledgements

None.

Funding

The study was not funded.

Conflict of Interest

None.

References

[1] Alnahdi, N., & Alnanih, R. (2024). A novel information model for software interface reliability in the software development life cycle. *Procedia Computer Science*, 251, 116-123. doi: 10.1016/j.procs.2024.11.091.

[2] Asif, M., & Ahmed, J. (2020). A novel case base reasoning and frequent pattern based decision support system for mitigating software risk factors. *IEEE Access*, 8, 102278-102291. doi: 10.1109/ACCESS.2020.2999036.

[3] Butt, S., Ur Rehman Khan, S., Hussain, S., & Wang, W.-L. (2023). A conceptual model supporting decision-making for test automation in Agile-based Software Development. *Data & Knowledge Engineering*, 144, article number 102111. doi: 10.1016/j.datak.2022.102111.

[4] Calinescu, R., Paterson, C., & Johnson, K. (2021). Efficient parametric model checking using domain knowledge. *IEEE Transactions on Software Engineering*, 47(6), 1114-1133. doi: 10.1109/TSE.2019.2912958.

[5] Chen, X., & Deng, Y. (2024). Evidential software risk assessment model on ordered frame of discernment. *Expert Systems with Applications*, 250, article number 123786. doi: 10.1016/j.eswa.2024.123786.

[6] Díaz, J., Pérez, J., Gallardo, C., & González-Prieto, A. (2023). Applying inter-rater reliability and agreement in collaborative Grounded Theory studies in software engineering. *Journal of Systems and Software,* 195, article number 111520. doi: 10.1016/j.jss.2022.111520.

[7] Doroshenko, A., Finin, G., & Tceitlin, G. (2004). *Algebra-algorithmic basics of programming*. Kyiv: Naukova Dumka.

[8] Duarte, B., de Almeida Falbo, R., Guizzardi, G., Guizzardi, R., & Silva Souza, V.E. (2021). An ontological analysis of software system anomalies and their associated risks. *Data & Knowledge Engineering*, 134, article number 101892. doi: 10.1016/j.datak.2021.101892.

[9] Ferreira, H., Nakagawa, Y., & Santos, P. (2023). Towards an understanding of reliability of software-intensive systems-of-systems. *Information and Software Technology,* 158, article number 107186. doi: 10.1016/j.infsof.2023.107186.

[10] Filippetto, A., Lima, R., Luis, J., & Barbosa, V. (2021). A risk prediction model for software project management based on similarity analysis of context histories. *Information and Software Technology,* 131, article number 106497. doi: 10.1016/j.infsof.2020.106497.

[11] Hanagal, D., & Bhalerao, N. (2021). *Software reliability growth models*. Singapore: Springer.

[12] Hovorushchenko, T. (2021). Method of the software risks management. In *Proceedings of the 2nd international workshop on computational & information technologies for risk-informed systems* (pp. 26-38). Kherson: CITR.
[13] Littlewood, B., Salako, K., Strigini, L., & Zhao, X. (2020). On reliability assessment when a software-based system is replaced by a thought-to-be-better one. *Reliability Engineering & System Safety*, 197, article number 106752. doi: 10.1016/j.ress.2019.106752.
[14] Macak, M., Daubner, L., Sani, F., & Buhnova, B. (2022). Process mining usage in cybersecurity and software reliability analysis: A systematic literature review. *Array*, 13, article number 100120. doi: 10.1016/j.array.2021.100120.
[15] Ouriques, R., Wnuk, K., Gorschek, T., & Svensson, B. (2023). The role of knowledge-based resources in Agile Software Development contexts. *Journal of Systems and Software*, 197, article number 111572. doi: 10.1016/j.jss.2022.111572.
[16] Pradhan, V., Kumar, A., & Dhar, J. (2023). Emerging trends and future directions in software reliability growth modeling. In H. Garg & M. Ram (Eds.), *Advances in reliability science, engineering reliability and risk assessment* (pp. 131-144). Amsterdam: Elsevier. doi: 10.1016/B978-0-323-91943-2.00011-3.
[17] Rajapaksha, D., Tantithamthavorn, C., Jiarpakdee, J., Bergmeir, C., Grundy, J., & Buntine, W. (2022). SQAPlanner: Generating data-informed software quality improvement plans. *IEEE Transactions on Software Engineering,* 48(8), 2814-2835. doi: 10.1109/TSE.2021.3070559.
[18] Rakytyanska, H., & Prus, B. (2024). Constructing prototype-based granular fuzzy rules for scene classification on mobile devices. In S. Babichev & V. Lytvynenko (Eds.), *Lecture notes in data engineering, computational intelligence, and decision-making* (pp. 194-218). Cham: Springer. doi: 10.1007/978-3-031-70959-3_10.
[19] Rotshtein, A., & Rakytyanska, H. (2012). *Fuzzy evidence in identification, forecasting and diagnosis.* Heidelberg: Springer.
[20] Rotshtein, A., Shtovba, S., & Kozachko, A. (2007). *Modeling and optimization of multivariable algorithmic processes reliability*. Vinnytsia: UNIVERSUM.
[21] Saaty, T.L. (1994). *Fundamentals of decision making and priority theory with the analytic hierarchy process*. Pittsburgh: RWS Publications.
[22] Samal, U., & Kumar, A. (2024). A software reliability model incorporating fault removal efficiency and it's release policy. *Computational Statistics,* 39, 3137-3155. doi: 10.1007/s00180-023-01430-9.
[23] Samal, U., Kushwaha, S., Nain, G., Singh, S., Usmani, S., & Kumar, A. (2025). Chapter 14. Necessity of fuzzy logic: Trends in software reliability assessment. In A. Kumar, A.S. Bhandari & M. Ram (Eds.), *Reliability assessment and optimization of complex systems* (pp. 275-285). Amsterdam: Elsevier. doi: 10.1016/B978-0-443-29112-8.00009-8.
[24] Sokol, O. (2025). Automation of error detection in code using machine learning. *Bulletin of Cherkasy State Technological University*, 30(1), 35-47. doi: 10.62660/bcstu/1.2025.35.
[25] Thieme, C., Mosleh, A., Utne, I., & Hegde, J. (2020). Incorporating software failure in risk analysis. Part 1: Software functional failure mode classification. *Reliability Engineering & System Safety*, 197, article number 106803. doi: 10.1016/j.ress.2020.106803.
[26] Yakovyna, V., & Symets, I. (2021). Reliability assessment of CubeSat nanosatellites flight software by high-order Markov chains. *Procedia Computer Science*, 192(C), 447-456. doi: 10.1016/j.procs.2021.08.046.

Нечіткий алгоритмічний аналіз надійності програмного забезпечення

Ганна Ракитянська

Кандидат технічних наук, доцент
Вінницький національний технічний університет
21021, вул. Хмельницьке шосе, 95, м. Вінниця, Україна
https://orcid.org/0000-0001-5863-3730

Богдан Прус

Аспірант
Вінницький національний технічний університет
21021, вул. Хмельницьке шосе, 95, м. Вінниця, Україна
https://orcid.org/0009-0008-7214-0949

Анотація. Актуальність дослідження зумовлена необхідністю розробки інтерпретабельних процес-орієнтованих моделей, які дозволяють оцінити зростання функції надійності залежно від розподілу зусиль. Мета роботи полягала в моделюванні процесів, пов'язаних із внесенням, виявленням та виправленням помилок засобами алгебри алгоритмів та нечіткої логіки. Запропонована методологія аналізу надійності програмного забезпечення базувалася на теорії надійності алгоритмічних процесів. Логіко-алгоритмічна модель процесу розробки побудована на основі лінійної, альтернативної та ітеративної операторних структур. Послідовність робіт без зворотного зв'язку описана лінійною структурою. Етапи верифікації та валідації описані за допомогою альтернативної та ітеративної алгоритмічних структур. Процес перевірки та виправлення, коли виявлені помилки негайно усувалися, а нові помилки не вносилися, описано альтернативною структурою. Процес налагодження, під час якого можуть вноситись нові помилки, описано ітеративною структурою. Логіко-алгоритмічна модель у вигляді нечіткої бази знань дозволила проектувати програмне забезпечення з необхідними рівнями надійності та витрат, використовуючи покращувальні перетворення. Система нечітких логічних рівнянь пов'язувала рівні правильності робочих, контрольних та доробочних операцій з можливістю правильного виконання процесу розробки. Розподіл зусиль формалізовано за допомогою покращувальних підстановок, введених у логіко-алгоритмічну модель. Керувальні змінні, пов'язані з покращувальними підстановками, інтерпретувалися як якість виконання робочих, контрольних та доробочних операцій. Запропонована нечітка модель надійності програмного забезпечення дозволила оцінити ризики процесу розробки на основі експертної та експериментальної інформації про надійність та часові характеристики етапів життєвого циклу. Нечітка модель була побудована шляхом перенесення надійних частин процесу розробки, отриманих з історій помилок та дефектів попередніх проєктів, у процес-орієнтовану модель надійності поточного проєкту. Розглянуто приклад аналізу надійності процесу розробки мобільного додатку для агрегації зображень, де впливовими факторами є безпомилкове виконання робочих, контрольних і доробочних операцій. Практичне значення дослідження полягає у створенні інструментарію, що дає змогу прогнозувати надійність програмного забезпечення на різних етапах його життєвого циклу, оптимізувати розподіл ресурсів між виявленням і виправленням помилок та зменшувати ризики невдалих рішень у проєктуванні і налагодженні

Ключові слова: надійність програмного забезпечення; оцінка ризиків процесу розробки; розподіл зусиль; логіко-алгоритмічна модель; нечітка модель надійності