Chapter 7

Models for Testing Modifiable Systems

Alexey Markov, Alexander Barabanov and Valentin Tsirlov

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.75126

Abstract

The work describes reliability and security growth models for modifiable software systems as a result of revisions and tests performed for specified input data areas. The work shows that the known reliability growth models are of monotonically increasing type, which is not in line with current multi-version team technologies of software development that are primarily based on the open-source code. The authors suggest new non-monotonically increasing models of software reliability evaluation and planning that allow taking into account the effect of decreased reliability resulting from updates or wavefront errors. The work describes the elaborated bigeminal and generic reliability evaluation model as well as the models and test planning procedures. The work includes calculated expressions for the evaluation of the model accuracy and shows that the developed models are adequate to real data. An example is given of transition from probability models to fuzzy models in case of incomplete basic data. The work provides general recommendations for selection of software tool testing models.

Keywords: modifiable systems, program tests, software reliability, software security, test planning, reliability growth models, debugging models, nonmonotone models, open-source reliability

1. Introduction

According to the ISO/IEC 17000 standards, the main procedures of software compliance evaluation include acceptance tests, certifications tests, and follow-up inspection control.

For the purpose of certification tests, the software to be assessed for compliance is submitted in a complete form, usually upon the final completion of acceptance testing. At the same time, during preliminary and acceptance tests, the assessed software is revised in order to correct

detected errors of different types. Considering all this, at the stage of certification, the information systems and software products can be regarded as non-modifiable, while at the stage of acceptance tests, they are defined as modifiable systems. This defines the difference in approaches to developing the mathematical test models.

2. Non-monotonic models of software reliability and security evaluation

In the course of preliminary acceptance testing and trial operation of information systems, it is important to define the moment when the testing can be considered complete and the system can undergo commissioning procedures. As for high-security software (including software intended for processing of confidential information or software used in critical system applications), current regulatory documents require that the test results be formalized[1]. In these cases, the test completion criteria (documented in test certificates), besides the very fact that the specified requirements are met, also include the values of test confidence parameters and parameters of the achieved level of reliability or correctness considering the specified evaluation accuracy. For these purposes it is reasonable to use mathematical models [1, 2] that are classified in this work in the following way (**Figure 1**):

- Debugging models that allow assessing the software reliability parameters depending on the results of program runs on specified data areas and subsequent program modifications
- Time reliability growth models that allow assessing the software reliability parameters depending on the time of test considering the corrected program errors

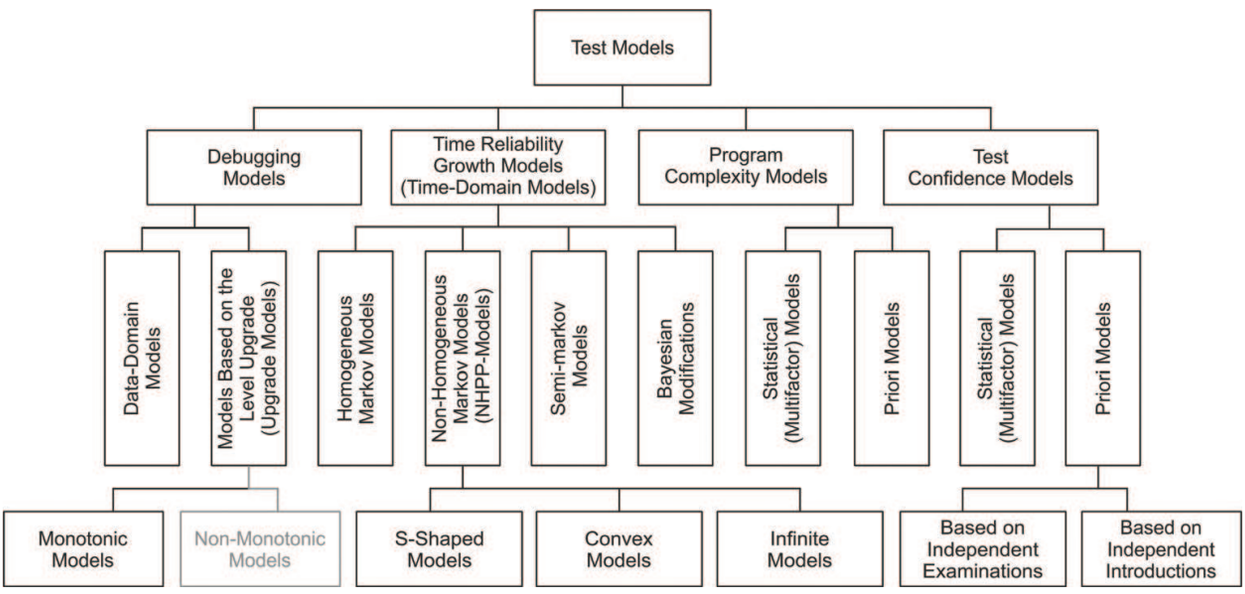

Figure 1. The classification of mathematical models of tests.

[1]ISO/IEC 15408–3:2008. IT—Security techniques—Evaluation criteria for IT security—Part 3.

- Test confidence models that allow assessing confidence parameters of the test procedure
- Program complexity models based on the relationship between the software complexity metrics and program quality, reliability, and safety parameters

It should be noted that the latter three classes of test models are rather well developed[2] [3–14]. For example, today, about 200 time models are known, mainly, NHPP models (e.g., [15–21]). At the same time, debugging models (also known as reliability growth models based on input data areas and revisions) are usually related only to Nelson's model and its modifications [22] developed at the dawn of the programming theory and do not reflect peculiarities of the modern team software development methods.

The early stage of testing is the typical scope of application for the debugging models. This is due to the fact that this period of a software system lifecycle is characterized by active modification of the programs aimed at correcting the detected errors. The models described in the literature reflect monotonic (typically, exponential, or logistic) growth of software operation reliability, which is not always true, as, for instance, in the case of implementation of the open-source software, multi-version or multiple replica software developed at different times by absolutely different teams of developers with diverse qualification, different styles, using various technologies and development systems, etc. This chapter is devoted to justification of new non-monotonic models and calculation of expressions of their parameters. We shall assume that the software reliability is a set of properties that characterize the ability of the program to maintain the specified level of availability in specified conditions during the specified period of time.[3] It is important to note that if the level of availability is restricted by security and vulnerability defects, the term **reliability** shall be equal to the term **information security**.

Definition of the software reliability is fundamentally different from that of the hardware, mainly, due to the fact that the software is not prone to aging in time. Two characteristics of the software reliability can be mentioned:

1. As a characteristic, reliability can alter only as the result of the software modification (i.e., when the tested object is changed), and the level of reliability can either increase or decrease.
2. Values of the software reliability parameters are valid for those input data classes that were used for their calculation.

A number of debugging models were described in the literature, namely, Nelson's model, matrix model, LaPadula model, and other models [2, 5, 12, 13, 22], that reflect the stepwise monotonic growth of reliability and thus do not take into account the possibility of obvious reliability decrease, for example, due to introduction of global wavefront errors or addition of new functionality. Experience gathered by the test laboratory shows that application of such mathematical models either gives unreliable results or significantly increases the time required

[2] IEEE Std. 1633–2008 (R2016). Recommended Practice on Software Reliability.
[3] GOST 28806–90. Software quality. Terms and definitions.

to assess the software reliability [23]. That is why it is necessary to substantiate a non-monotonic software reliability model and obtain calculated values of its parameters which are also required to assess its reliability.

According to the abovementioned first property of the software reliability, the process of software modification can be represented in the form of random transitions from one reliability state to another. The moments of transition are modifications of the tested object, which can be described as any changes of the program aimed at correcting the detected errors or developing the program.

We shall define the main software reliability indicator as the level of the program reliability, which represents the probability of its error-free starting for a set of basic data from the specified range. Considering the above said, we have the following software reliability change model:

$$P_u = P_0 + \sum_{j=1}^{u} \Delta P_j, \tag{1}$$

where P_0 is the initial level of reliability ($0 \le P_0 < 1$), u is the number of completed revisions of the software, and ΔP_j is increment of reliability after j revision.

The process of software reliability change can be graphically presented as a stepwise reliability growth function (**Figure 2**).

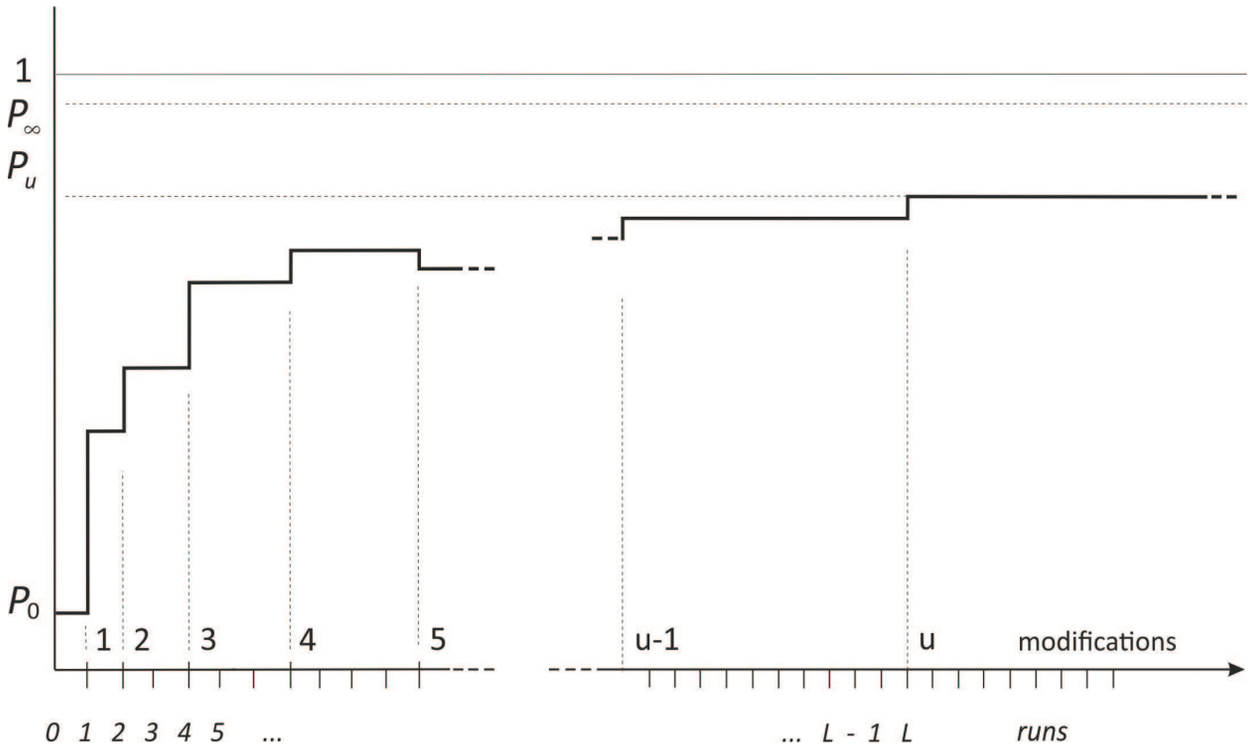

Figure 2. Change of the reliability level as a result of revisions.

If we view software as a modifiable system, the change of the software reliability level after j number of revisions can be represented using the following linear operator:

$$\Delta P_j = A_j\left(1 - P_{j-1}\right) - B_j P_{j-1}, \tag{2}$$

where P_{j-1} is the probability of error-free operation of the software after (*j-1*) revision, $\left(1 - P_{j-1}\right)$ is the probability of detection of software errors after (*j-1*) revision, A_j is the revision efficiency factor that characterizes the decreased probability of error as the result of j revision, and B_j is the revision negativeness factor that characterizes reliability decreases due to j revision.

Proceeding to the recurrent expression and considering the maximum level of reliability to be equal to $P_\infty = \frac{A_j}{A_j + B_j}$, we can obtain the software reliability evaluation model:

$$P_u = P_\infty - (P_\infty - P_0)\prod_{j=1}^{u}\left(1 - A_j/P_\infty\right), \tag{3}$$

where P_0 is the initial level of reliability, P_∞ is the maximum level of reliability ($0 \le P_0 < P_\infty \le 1$), and u is the number of completed revisions.

The obtained expression (Eq. (3)) takes into account the possibility of uneven reliability growth of the tested object and the general trend of ΔP_j growth decrease when the level of reliability P_j increases. However, when the model is presented in this way, it is generally monotonic since it does not take into account the different effects produced by fundamentally different types of modifications, for instance, changes of the software in order to correct errors or introduce new functional elements. Besides, the model does not reflect the degree of modification complexity and, consequently, probability of wavefront errors. Obviously, the model represented in this form can be regarded as a monotonic reliability growth model [23].

2.1. Bigeminal model of software reliability and security evaluation

In order to overcome the drawback described in the previous section, we offer a bigeminal reliability evaluation model based on metrics of the source code modification k_{ij}, for example, for error correction and software updates. This metric has no limits (i.e., the complexity metric that is most suitable for the software system and development system can be used[4]), which ensures comprehensive description of the considered process. Thus, if the revision efficiency factor $A_j = \sum_{i=0}^{2} a_i k_{ij}$, we can obtain the main calculated expression of the bigeminal reliability evaluation model:

$$P_u = P_\infty - (P_\infty - P_0)\prod_{j=1}^{u}\left(1 - \sum_{i=1}^{2} a_i k_{ij}/P_\infty\right), \tag{4}$$

[4]IEEE Std. 1061–1998 (R2009). Standard for a Software Quality Metrics Methodology.

where u is the number of completed revisions of the software, a_1 is the efficiency factor of the software revisions aimed at error correction, a_2 is the efficiency factor of the software revisions aimed at introduction of new functions, and k_{ij} is the scope of j revision with the purpose of correction or update.

The bigeminal model (Eq. (4)) depends on four parameters $(P_0, P_\infty, a_1, a_2)$ that can be easily calculated with the use, for instance, of the maximum likelihood method.

2.2. Generic model of software reliability and security evaluation model

Though the bigeminal model has the advantage of being mathematically simple, it does not take into account peculiarities of various types of software modifications relating to new functionality, correction of global and local errors, elimination of vulnerabilities, issues of integration and upgrade or degradation of the operating system, optimization, etc.

In order to address these issues and increase the model accuracy, we should introduce classification of modifications (including corrected errors) taking the following calculated expression for the revision efficiency factor:

$$A_j = \sum_{i=0}^{e} a_i k_{ij}, \tag{5}$$

where e is the number of software modification classes.

Considering all this, we can obtain a generic non-monotonic reliability evaluation model:

$$P_u = P_\infty - (P_\infty - P_0) \prod_{j=1}^{u} \left(1 - \frac{\sum_{i=1}^{e} a_i k_{ij}}{P_\infty}\right), \tag{6}$$

where e is the number of software modification classes.

This model depends on $(e + 2)$ parameters. The following section includes an example of the model parameter calculation using the maximum likelihood method.

2.3. Calculated expressions of reliability and security evaluation model parameters

The maximum likelihood method can be used to calculate parameters of the bigeminal (Eq. (4)) and generic (Eq. (6)) models. The following data obtained during the software tests can be used as the initial statistics: the set of tests $\{n_j\}$, the set of failed tests (failures) $\{\widehat{m}_j\}$ between revisions, and the set of revision complexity metrics $\{k_{ij}\}$. In this case, if the software runs are considered independent, the function of maximum likelihood represents the probability of obtaining the total sample $(n_i, \widehat{m}_j, j = \overline{1, u})$ of the number of failures in the performed series of software runs:

$$L_u = \prod_{j=1}^{u} C_{m_j}^{n_j} P_j^{n_i - \widehat{m_j}} \left(1 - P_j\right)^{\widehat{m_j}}, \tag{7}$$

where $C_{m_j}^{n_j} = \frac{n_j!}{\widehat{m_j}!\left(n_j - \widehat{m_j}\right)}$, u is the number of the last software revision, P_j is the probability of success of each of the n_j runs of j series, and $\widehat{m_j}$ is the number of failures in n_j runs.

For the sake of convenience, we can take the logarithm of the function L_u and modify the function in the following way:

$$\begin{aligned} \ln\left(L_u\right) = \sum_{J=1}^{u} \Bigg(& \widehat{m_j} \ln\left(1 - P_\infty + \left(P_\infty - P_0\right) \prod_{l=1}^{j} \left(1 - \sum\nolimits_{i=1}^{e} \frac{a_i k_{ij}}{P_\infty}\right)\right) \\ & + \left(n_j - \widehat{m_j}\right) \ln\left(P_\infty + \left(P_\infty - P_0\right) \prod_{l=1}^{j} \left(1 - \sum\nolimits_{i=1}^{e} \frac{a_i k_{ij}}{P_\infty}\right)\right)\Bigg). \end{aligned} \tag{8}$$

The obtained reduced function is convex and is defined for a convex set; that is why in order to find the maximum of the likelihood function, we can use, for example, the modified steepest descent method with the variable increment parameter h^r:

$$\begin{cases} P_0^{r+1} = P_0^r + h^r \left(\dfrac{\partial \ln L\left(P_0^r, P_\infty^r, a_1^r, \ldots, a_e^r\right)}{\partial P_0}\right); \\ P_\infty^{r+1} = P_\infty^r + h^r \left(\dfrac{\partial \ln L\left(P_0^{r+1}, P_\infty^r, a_1^r, \ldots, a_e^r\right)}{\partial P_\infty}\right); \\ a_1^{r+1} = a_1^r + h^r \left(\dfrac{\partial \ln L\left(P_0^{r+1}, P_\infty^{r+1}, a_1^r, \ldots, a_e^r\right)}{\partial a_1}\right); \\ \ldots \\ a_e^{r+1} = a_e^r + h^r \left(\dfrac{\partial \ln L\left(P_0^{r+1}, P_\infty^{r+1}, a_1^{r+1}, \ldots, a_e^r\right)}{\partial a_e}\right), \end{cases} \tag{9}$$

where r is the iteration number.

The following new calculated expressions of partial derivatives of the reduced maximum likelihood function were obtained during this study:

$$\begin{cases} \dfrac{dlnL_j}{dP_0} = \sum\limits_{l=0}^{j} w_l a_l; \\ \dfrac{dlnL_j}{dP_\infty} = \sum\limits_{l=0}^{j} \left(w_l \left(\left(\dfrac{P_0 - P_\infty}{P_\infty} \alpha_l \beta_l - \alpha_l\right) + 1\right)\right); \\ \dfrac{dlnL_j}{da_i} = \sum\limits_{l=0}^{j} \left(w_j \left(\dfrac{P_0 - P_\infty}{P_\infty} \alpha_l \, \gamma_{li}\right)\right), \end{cases} \tag{10}$$

where $w_j = \frac{n_j - m_j}{P_j} - \frac{m_j}{1-P_j}$; $\alpha_j = \prod_{l=1}^{j}\left(1 - \frac{\sum_{i=1}^{e} a_i k_{li}}{P_\infty}\right)$; $\beta_j = \sum_{l=1}^{j} \frac{\sum_{i=1}^{e} a_i k_{li}}{1-\sum_{i=1}^{e} a_i k_{li}/P_\infty}$, $\gamma_{ji} = \sum_{l=1}^{j} \frac{-k_{li}}{1-\sum_{i=1}^{e} a_i k_{li}/P_\infty}$.

Judging from the practical experience, the following accuracy is sufficient in order to define evaluations P_0,P_∞,a_1, …, a_e:

$$\begin{cases} P_0^{r+1} - P_0^r \le 0.001; \\ P_\infty^{r+1} - P_\infty^r \le 0.001; \\ a_i^{r+1} - a_i^r \le 0.0001. \end{cases}$$

Improving accuracy of parameters, a_i ($i = \overline{1, e}$) definition is related to their strong effect on the function P_j of reliability evaluation. Zero-order approximations can be found using the statistical modeling method for logical intervals:

$$\begin{cases} 0 \le P_0 \le 1 - \left(\frac{M_0}{N_0}\right); \\ 1 - \left(\frac{M_\infty}{N_\infty}\right) \le P_\infty \le 1; \\ \frac{1}{K_i e}\left(1 - \sqrt{\frac{M_\infty N_0}{N_\infty M_0}}\right) \le a_i \le \frac{1}{K_i^{max} e}, \end{cases} \tag{11}$$

where M_0 is the number of failures in the first N_0 runs, M_∞ is the number of failures in the last N_∞ runs, and K_i^{max} is the maximum value of k_{ij} when $\overline{j = 1, u}$ and $K_i = \sum_{i=1}^{e} k_{ji}$.

Thus, if we assume that $\widehat{P_0}$, $\widehat{P_\infty}$, $\widehat{a_1}$, …, $\widehat{a_e}$ are random values distributed evenly on previously specified intervals, we should perform a certain number of samples and select a set of parameters corresponding to the maximum likelihood function. This set shall be considered to be the desired initial values. As the experience shows, during the initial stages of tests, the general trend of software reliability increase due to modifications may not be present. This can lead to unreliable results obtained with the use of the maximum likelihood method (an infinite number of iterations will be required to calculate the function maximum).

In order to overcome this drawback, the method of relative entropy minimization can be used:

$$I_u = \sum_{j=1}^{u}\left(\frac{m_j}{n_j}\ln\frac{m_j}{n_j P_j} + \frac{n_j - m_j}{n_j}\ln\frac{n_j - m_j}{n_j(1 - P_j)}\right), \tag{12}$$

where m_j is the number of failed runs of the total number n_j of runs of j series and u is the number of completed software revisions.

In order to check the necessary and sufficient condition for acceptability of the maximum likelihood method, the following ratio can be used:

$$\frac{\sum_{j=1}^{u}(j-1)(n_j - m_j)}{\sum_{j=1}^{u}(j-1)} > \frac{\sum_{j=1}^{u}(n_j - m_j)}{u}. \tag{13}$$

2.4. Estimation of accuracy of software reliability and security evaluation model

Authors of the absolute majority of reliability growth models do not provide any analytical assessment of their accuracy, which makes it difficult to select a specific model. This works allows excluding this drawback. The accuracy of the software reliability estimation can be characterized by the root-mean-square deviation. In order to obtain an accuracy estimation model, it is convenient to use the linearization method [24]. In this case, the root-mean-square deviation shall be defined according to the following equation:

$$\sigma_j = \left(\left(\partial P_j/\partial P_0\right)^2 \delta_{P_0}^2 + \ldots + \left(\partial P_j/\partial a_e\right)^2 \delta_{a_e}^2 + 2\left(\frac{\partial P_j}{\partial P_0}\right)\left(\frac{\partial P_j}{\partial P_\infty}\right)\delta_{P_0}\,\delta_{P_\infty}\,\rho_{P_0P_\infty} + \ldots + , \right. \tag{14}$$

where ρ_{xy} is correlation factor of parameters x and y.

The following original calculated expressions were obtained in this work in order to get the values of partial derivatives of the reliability growth function:

$$\begin{cases} \dfrac{dP_j}{dP_0} = \alpha_j; \\ \dfrac{dP_v}{dP_\infty} = \left(\dfrac{P_0 - P_\infty}{P_\infty^2}\alpha_j\beta_j - \alpha_j + 1\right); \\ \dfrac{dP_j}{da_i} = \dfrac{P_0 - P_\infty}{P_\infty}\alpha_j\gamma_{ji}, \end{cases} \tag{15}$$

where $\alpha_j = \prod_{l=1}^{j}\left(1 - \frac{\sum_{i=1}^{e} a_i k_{li}}{P_\infty}\right)$, $\beta_j = \sum_{l=1}^{j}\frac{\sum_{i=1}^{e} a_i k_{li}}{1-\sum_{i=1}^{e} a_i k_{li}/P_\infty}$, and $\gamma_{ji} = \sum_{l=1}^{j}\frac{-k_{li}}{1-\sum_{i=1}^{e} a_i k_{li}/P_\infty}$.

Other parameters of the formula can be defined from the covariance matrix that includes dispersions and correlation moments of the desired values:

$$\mathcal{K} = \begin{bmatrix} \delta_{P_0}^2 & \delta_{P_0}\delta_{P_\infty}\rho_{P_0P_\infty} & \cdots & \delta_{P_0}\,\delta_{a_e}\,\rho_{P_0a_e} \\ \delta_{P_0}\,\delta_{P_\infty}\,\rho_{P_0P_\infty} & \delta_{P_\infty}^2 & \cdots & \delta_{P_\infty}\,\delta_{a_e}\,\rho_{P_\infty a_e} \\ \cdots & \cdots & \cdots & \cdots \\ \delta_{P_0}\delta_{a_e}\,\rho_{P_0a_e} & \delta_{P_\infty}\delta_{a_e}\,\rho_{P_\infty a_e} & \cdots & \delta_{a_e}^2 \end{bmatrix}. \tag{16}$$

The following equation can be used for its formulation:

$$\mathcal{K} = -\mathcal{M}^{-1}, \tag{17}$$

where $\mathcal{M}$ is matrix of the second partial derivatives of the likelihood function:

$$\mathcal{M} = \begin{bmatrix} \dfrac{\partial^2 \ln L_u}{\partial P_0^2} & \dfrac{\partial^2 \ln L_u}{\partial P_0 P_\infty} & \cdots & \dfrac{\partial^2 \ln L_u}{\partial P_0 a_e} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial^2 \ln L_u}{\partial P_0 a_e} & \dfrac{\partial^2 \ln L_u}{\partial P_\infty a_e} & \cdots & \dfrac{\partial^2 \ln L_u}{\partial a_e^2} \end{bmatrix}. \tag{18}$$

The following original calculated expressions were obtained in this work in order to get second partial derivatives:

$$\begin{cases} \dfrac{dlnL_u}{dP_0} = \sum\limits_{j=0}^{u} w_j a_j; \\ \dfrac{dlnL_u}{dP_\infty} = \sum\limits_{j=1}^{u} \left(w_j \left(\left(\dfrac{P_0 - P_\infty}{P_\infty} \alpha_j \beta_j - \alpha_j \right) + 1 \right) \right); \\ \dfrac{dlnL_u}{da_i} = \sum\limits_{j=0}^{u} \left(w_j \left(\dfrac{P_0 - P_\infty}{P_\infty} \alpha_j \, \gamma_{ji} \right) \right), \end{cases} \tag{19}$$

where $w_j = \frac{n_j - m_j}{P_j} - \frac{m_j}{1 - P_j}$.

2.5. Software reliability and security evaluation algorithm

Figure 3 shows the algorithm of software reliability and security evaluation

2.6. Input data normalization of the developed models

Nonstandard situations occurring in the course of the information system operation may lead to the disruption of specified input data, which, according to the second property of the software reliability, results in the inadequacy of obtained values. This situation occurs when invalid input data classes are used and the frequency of utilization of the input data classes does not correspond to the frequency that was used during testing or specified in the technical requirements. This may happen during trial operation aimed at performing accelerated tests of the software, due to the change of environment and in other cases. This situation can be taken into account by correcting the calculated reliability values. The correction can be done using the method of multiple factor analysis. In this case, the program input classes are broken into n equivalence classes. The function of reliability value dependence on frequency n_j of application of equivalency classes is calculated:

$$P^u = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \sum_{i \neq j}^{n} \beta_{ij} x_i x_j + \sum_{i=0}^{n} \beta_{ii} x_j^2 \ldots, \tag{20}$$

where x_i is the frequency of application of *i*-class of input data and β_i is the significance ratio of *i*-class of input data.

The study has shown that first-order polynomial is sufficient for correction:

$$P^u = \beta_0 + \sum_{i \neq j}^{n} \beta_{ij} x_i x_j, \tag{21}$$

where x_i is the frequency of application of *fig*-class of input data and β_i is the significance ratio of *i*-class of input data.

http://dx.doi.org/10.5772/intechopen.75126

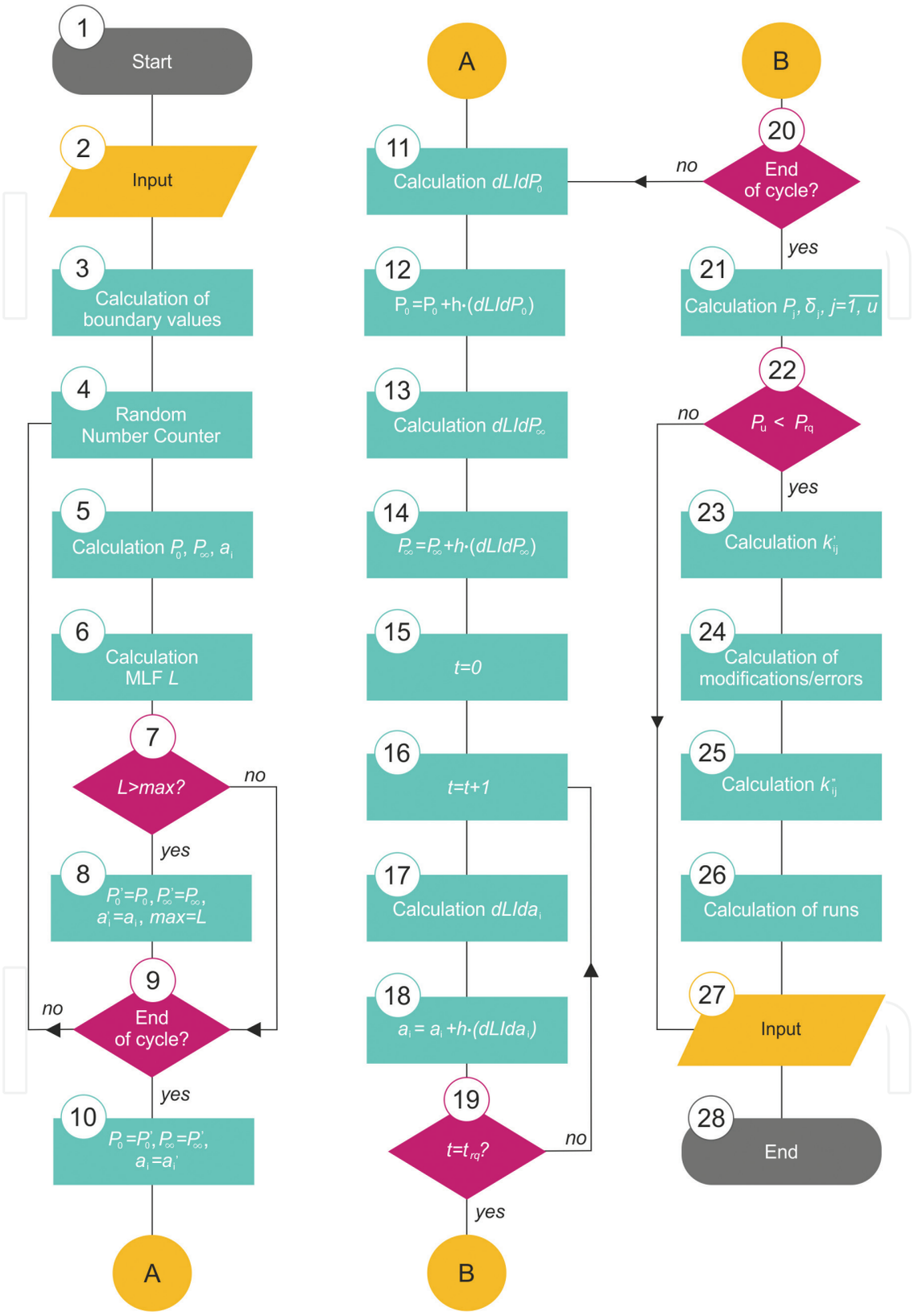

Figure 3. Software reliability and security evaluation algorithm.

This model has two unknown parameters that can be easily found with the help of the least squares methods.

2.7. Approbation of the non-monotonic software reliability and security evaluation model

The study has shown that the suggested non-monotonic models (Eqs. (4) and (6)) provide high accuracy ($\sigma_j < 0.001$) when the number of revisions exceeds 10 and the number of runs exceeds 50. In order to control the model consistency with the basic data, the Mises criterion was used (at threshold value of 0.01) [25]:

$$\begin{gathered} \omega_n^2 n \in [0.26; 1.9] \\ \widehat{u}(0.01) = 2.1, \end{gathered} \tag{22}$$

where ω_n^2 is the Mises criterion and $\widehat{u}$ is the threshold value.

Analysis of the effect of the software revision efficiency factor on the model (Eq. (6)) accuracy has shown that the accuracy can increase by an order of magnitude on the condition that revision classes are taken into account. Comparison of the suggested models with the well-known debugging models has demonstrated a number of their advantages, namely:

- Taking into account the possible steep decrease of reliability due to upgrades
- Possibility of taking into account the revision complexity
- Absence of restrictions for tests and information acquisition
- Possibility of taking into account the software reliability values obtained during the previous stages of development and implementation
- Absence of subjective factors, such as programmer's qualification and the level of development technology
- Ease of application since there is no need to calculate probability of all program paths as, for example, in Nelson's model and its modifications [22]

Thus, the study actually substantiates the method of test planning based on utilization of the non-monotonic software reliability evaluation model using the results of runs and revisions. Within the scope of the suggested method, we obtained calculated expressions of parameters of the software reliability evaluation model and estimated accuracy and test planning. The suggested generic non-monotonic model (Eq. (6)) allows considering probable moments of the software reliability decrease typical, for instance, for open-source software development, multiple version software, etc. Accuracy of the generic model depends on how the task of software revision classification is solved. The model can be integrated with software reliability values obtained during the early stages of the software development. Simplification of the model allows reducing it to exponential NHPP models of reliability growth used at the stages of information system operation and upgrade [23].

The main advantage of the suggested non-monotonic models is the possibility to increase accuracy by more than 10% (as the results of introducing revision categories), which is equal

to 5–15% reduction of the required number of software runs during test procedures. It should be noted that debugging models provide low accuracy at low statistics; however, this drawback can be avoided by using appropriate accuracy increase techniques, including Wald's method.

The suggested method and models can be also recommended to estimate the parameters of various modifiable and learning systems.

3. Test planning and software revision models

In the course of the software reliability management, it is necessary to plan the cost of testing in order to achieve the required level of the software reliability. Thus, it is useful to evaluate the trends relevant to the software development and implementation and predict the number of remaining errors and complexity of their correction.

The models (Eqs. (3), (4), (6)) described above can be used to calculate a number of planning indicators. Unfortunately, statistical models of reliability evaluation do not allow predicting the frequency of corrections of a specific type but only use this information. Specific revisions that depend on operating conditions, the achieved level of reliability, requirements for the software reliability, developers' qualification and experience and, consequently, their content may differ. In order to consider the revision types, it is reasonable to use the theory of multiple factor analysis. Since the change of the number of specific corrections is considered within the scope of revisions, the software modification complexity function can be approximated using, for example, a quadratic polynomial in one variable:

$$k_j = \kappa_0 + \kappa_1 j + \kappa_2 j^2, \tag{23}$$

where κ_0, κ_1, and κ_2 are the polynomial parameters ($j = \overline{1, u}$).

It is easy to demonstrate that the polynomial parameters have the following form:

$$\begin{cases} \kappa_0 = \dfrac{30\left(\sum_{j=1}^{u} \widehat{k}_j - \dfrac{2}{u(u-1)} \sum_{j=1}^{u} \widehat{k}_j j^2 - \beta_2\left(2+3u-3u^2-2u^3\right)\right)}{10(u-1)}; \\ \kappa_1 = \dfrac{6\left(\sum_{j=1}^{u} \widehat{k}_j - \dfrac{2}{u+1} \sum_{j=1}^{u} \widehat{k}_j j - \beta_2\left(1-u^2\right)u\right)}{u(1-u)}; \\ \kappa_2 = \dfrac{\sum_{j=1}^{u} \widehat{k}_j \dfrac{u^2+3u-2}{2} - u\sum_{j=1}^{u} \widehat{k}_j j - \dfrac{2}{u-1} \sum_{j=1}^{u} \widehat{k}_j j^2}{u-(4-u^2)}. \end{cases} \tag{24}$$

Then, assuming that the estimation P_u of the model parameters and the achieved software reliability level was obtained based on the available test data, we have the following calculated expression of the reliability-level prediction model:

$$P_{rq} = P_\infty - (P_\infty - P_u) \prod_{i=u+1}^{u+j} \left(1 - \frac{\sum_{i=1}^{e} a_i k_{ij}}{P_\infty}\right), \tag{25}$$

where P_{rq} is the required level of the software reliability, u is the number of the last revision, and j is the quantity of planned revisions.

The quantity of revisions required to achieve the desired level of reliability can be calculated using the cyclic recalculation of the expression (Eq. (25)). To this end P_u is calculated using the formula (Eq. (25)); further, in the cycle the value P_{u+j} is defined by increasing j. When the condition $P_{u+j} \geq P_{rg}$ is met, the cycle stops.

To simplify application of the predictive model, let us assume that $A_j = a$, which corresponds to the transition from the model (Eq. (6)) to (Eq. (3)). Then, after we reduce the expression (Eq. (25)) and take its logarithm, we will obtain the following expression required to evaluate the number J of software revisions that are necessary to achieve the desired level of reliability:

$$J = \left\| \left(\frac{\ln\left(\frac{P_\infty - P_{rg}}{P_\infty - P_u}\right)}{\ln\left(1 - a/P_\infty\right)} \right) \right\|, \tag{26}$$

where $\|א\|$ is the operation of obtaining of the nearest biggest integer א and a is the averaged software revision efficiency factor.

Assuming that revisions do not introduce additional errors (i.e., $P_\infty = 1$), we can obtain the formula for the number of remaining errors after u revision:

$$N_u = \left\| \left(\frac{\ln\left(\frac{1 - P_{rq}}{1 - P_u}\right)}{\ln\left(1 - a\right)} \right) \right\|. \tag{27}$$

4. Fuzzy model of software reliability and security evaluation-based on test results

Testing of software complexes for compliance with requirements for reliability and security is one of the most time-consuming and difficult stages of implementation of automation system. This is primarily due to the extreme structural complexity of modern software and its heterogeneity. Incomplete information on the software structure, principles and functioning, heterogeneity of its composition, presence of imported elements, and insufficient specifications make it difficult to evaluate and predict the software reliability. In these cases, traditional approaches to acquisition and forecasting of reliable values are associated with significant costs; that is why models based on the fuzzy sets of theory that allow estimating the software reliability with practically acceptable accuracy are of immediate interest [26–28].

At the present time, the literature describes fuzzy models of software reliability evaluation. These models are peculiar for their focus on static and dynamic analysis of the software graph, which is practically difficult due to the extreme structural complexity of the modern software systems and environments. We suggest describing the software testing and debugging process

by a non-monotonic software reliability growth function utilizing the fuzzy sets of theory in order to take into account the incompleteness of input data.

It is possible to demonstrate that the non-monotonic software reliability growth function looks as follows:

$$P_n = P_\infty - (P_\infty - P_0)\left(1 - \frac{a}{P_\infty}\right)^n, \tag{28}$$

where P_n is the probability of successful software run after n revision, a is the revision efficiency factor, P_0 is the initial level of reliability, and P_∞ is the maximum level of reliability.

This model depends on three parameters that can be conveniently calculated with the help of the maximum likelihood method. To create the likelihood function, it is reasonable to use the data recorded during the software tests, namely, the order of revisions, results of the software runs (whether any vulnerabilities were detected or not), and number of runs between the revisions.

It is easy to show that the maximum likelihood function logarithm will look as follows:

$$\ln(L_n) = \sum_{J=1}^{n} \left(\widehat{m_j} \ln\left(1 - P_\infty + (P_\infty - P_0)(1 - a/P_\infty)^i\right) + (n_j - \widehat{m_j}) \ln\left(P_\infty + (P_\infty - P_0)(1 - a/P_\infty)^i\right)\right).$$

where $\widehat{m_j}$ is the number of failures in n_j tests and n is the number of revisions.

The function $\ln(L_n)$ is convex and is defined for a convex set; that is why in order to effectively find the maximum of the likelihood function we can use, for example, the modified steepest descent method with the variable increment parameter, which allows obtaining the desired parameters of the model (Eq. (28)). The greatest difficulty of modeling the automation system operational readiness is determined by the fact that the software reliability level has to be evaluated in conditions of considerable uncertainty, namely:

1. Fuzziness of cause-and-effect relationship of the automation system as an ergatic system does not allow clear distinction between successful and unsuccessful revisions.
2. Definition of the amount of revisions as a function of the software metric characteristics does not always line up with reality. Knowledge of the software developers is required.
3. A number of errors appear as the result of shortcomings of the debugging and update procedures. Some errors are automatically eliminated at the final stages of the software development and do not require correction.

These uncertainties introduce a significant portion of subjectivity to the software reliability evaluation. The fuzzy set of theory allows taking them into account without substantial alteration of the model (Eq. (3)). This work is primarily aimed at solving this task.

4.1. Development of a fuzzy software reliability and security model

Let us present the information on the debugging process in the form of the set $\boldsymbol{X} = \{x_i\}$, where x_i is the software revision ($i = \overline{1, n}$). The number of relevant revisions is defined as

$m = \sum_{i=1}^{n} \chi_i$, where χ_i={0,1} is the characteristic function defining the presence of revision x_i. Let us formalize the probable fuzziness of the software revision by transition from the characteristic function {0,1} to continuum [0,1]. Then, we have:

1. Fuzzy set $\boldsymbol{A} = \{(x_i, \mu_A(x_i))\}$ representing a set of ordered couples of revisions x_i of the universal set X и membership functions that characterize availability of revisions.
2. Set of relevant revisions $\boldsymbol{R} = \{m\}$, $m = \overline{0, n}$.

In this case, the fuzzy set of relevant revisions will look as follows:

$$\boldsymbol{M} = \{(m, \mu_M(m))\}, \tag{29}$$

where $\mu_M(m)$ is the membership function defining the level of confidence in the fact that the number of relevant revisions is equal to m.

In general, the membership function can be found using the following expression:

$$\mu_M = \text{maxmin}\left\{ \overline{\mu_{i_1}}, \ldots \overline{\mu_{i_m}}, \mu_{j_1}, \ldots, \mu_{j_{(n-m)}} \right\}. \tag{30}$$

For the purpose of practical calculation, it is convenient to expand the revision membership function in ascending and descending order:

$$\begin{cases} \mu_0 \geq \mu_1 \geq \ldots \geq \mu_m \geq \mu_{m+1} \geq \ldots \geq \mu_n; \\ \\ \overline{\mu_0} \leq \overline{\mu_1} \leq \ldots \leq \overline{\mu_m} \leq \overline{\mu_{m+1}} \leq \ldots \leq \overline{\mu_n}. \end{cases} \tag{31}$$

This provides the main calculated ratio: $\mu_M(m) = \min(\overline{\mu_{m+1}}, \mu_m)$. The number of relevant revisions corresponding to the maximum level of confidence (i.e., to the maximum membership function) is equal to:

$$m = \sum_{i=0}^{n} m_i, \tag{32}$$

where $m_j = \begin{cases} 0, \text{если } \mu_i < \overline{\mu_i}; \\ 1, \text{если } \mu_i \geq \overline{\mu_i}. \end{cases}$

The maximum membership function can be calculated in the following way:

$$\mu_{max} = \min \max_{1<i<m} (\mu_i, \overline{\mu_i}). \tag{33}$$

By applying the generalization principle, we can move from the fuzzy set of relevant revisions (Eq. (29)) to the desired fuzzy set of the software reliability levels:

$$\boldsymbol{P} = \{(P_m, \mu_P(P_m))\}, \tag{34}$$

where $\mu_P(P_m) = \min(\overline{\mu_{i+1}}, \mu_i)$, $m = \overline{0, n;}$ and P_m —reliability level defined according to the formula (Eq. (3)).

It is important to note that considering the monotonic dependence of the software reliability level from the number of revisions, it is possible to formalize the fuzzy set $\boldsymbol{P}$ (Eq. (34)) with the complex of hierarchically ordered crisp sets. According to the decomposition theorem, we have:

$$\mu_P = \bigcup_{\alpha \in [0,1]} \left(\alpha \mu_{P_\alpha}\right), \tag{35}$$

where $\mu_{P_\alpha} = \begin{cases} 0, \text{если } \mu(x) \geq \alpha; \\ 1, \text{если } \mu(x) < \alpha. \end{cases}$

Then, by defining the value α based on the specific software operating conditions and accuracy of expert estimation, we can obtain the interval (guaranteed) software reliability level:

$$\boldsymbol{P} = \left\{P_m \mid \mu_M(m) \geq \alpha\right\}. \tag{36}$$

4.2. Example of possible application of fuzzy sets

Below is the simplest example of calculation of the software reliability level. During the debugging stage, 48 tests were carried out, 5 groups of defects were detected, and required revisions were performed. After the expert opinions were processed, the information on debugging was obtained in the form of a fuzzy set of revisions:

$$\mathrm{A} = \{(1, 0.0), (2, 0.4), (3, 0.2), (4, 1.0), (5, 0.9)\}. \tag{37}$$

Having arranged the fuzzy set A by the membership function values, we obtained a fuzzy set of relevant revisions:

$$\mathrm{M} = \{(0, 0.0), (1, 0.2), (2, 0.4), (3, 0.6), (4, 0.1), (5, 0.0)\}. \tag{38}$$

After we calculated reliability levels using the formulae (Eq. (3)), we obtained a fuzzy subset of the software reliability levels:

$$\mathrm{P} = \{(0.31, 0.2), (0.69, 0.4), (0.97, 0.6), (0.98, 0.1)\}. \tag{39}$$

According to the accepted assurance level α=0.4, we have.

$$\mathbf{P} = [0.69, 0.97]. \tag{40}$$

Thus, practical solutions suggested in the work take into account the uncertainties of software development and testing conditions. This allows obtaining rather accurate maximum and interval estimates of the software reliability and security. Analytical expressions allow simplifying the software reliability analysis as compared with the methods based on expert judgments. It is reasonable to apply the described results for planning of system and complex tests.

5. Evaluation models and test planning selection criteria

It should be noted that there is no universal model of the software evaluation and test planning. Moreover, beside the described classes of models, studies suggest simulation models [29],

structural models [22], fuzzy models [26, 27], interval models [30], software dynamic models [31–33], software/hardware complex models [34, 35], Bayesian model modifications [19, 30, 36, 37], as well as neural networks applied for certain scientific purposes [38, 39]. In order to select a suitable model, a number of qualitative and quantitative criteria can be suggested [40].

The following qualitative criteria can be used:

1. Ease of application that primarily concerns the degree of the model adequacy to the statistic collection system, i.e., utilized input data can be easily obtained; the data must be representative; and the input and output data must be clear for the experts.
2. Validity: the model must be reasonably (sufficiently) accurate to solve the tasks of analysis or synthesis in the field of software security. The positive property of the model that allows reducing the input sample is the ability to use a priori information and integrate data from other models.
3. Applicability for various tasks. Some models allow estimating a wide range of parameters necessary for experts at different stages of the software lifecycle, for instance, reliability values, expected number of errors of different types, predicted time and financial expenditure, developers' qualification, test quality, software cover parameters, etc.
4. Simplicity of implementation including the possibility of automated estimation based on well-known mathematical packages and libraries, model learning after revisions, taking into account the incomplete or incorrect input statistics, and other restrictions of the models.

The following quantitative criteria can be used:

- Evaluation accuracy parameters.
- Predictive model's quality parameters (convergence, noise tolerance, prediction accuracy, consistency).
- Information criteria of predictive model's quality (dimensionality, BIC/AIC criteria).

Combined and integral parameters, for instance:

$$IC = \max \sum_{i=1}^{K} k_i \chi_i, \tag{41}$$

where k_i is the weighting factor of i property of the considered model selected by the expert and χ_i. is the characteristic function of the i property.

As the study has shown, there are a lot of mathematical models that allow estimating the software reliability and security at different stages of lifecycle, which is important for budget planning. On a practical level, the described classification of models simplifies selection and integration of the models based on the available statistics.

It is important to bear in mind that due to the dynamic nature, complexity, and heterogeneity of modern software development projects, the described models are not able to meet strict requirements for accuracy and serve for making intuitive decisions relating to the software test planning for all sets of input data. However, the results obtained from the model application are useful both for substantiating the labor content of the tests and for preparation of reports, which can increase the customer's confidence in the work deliverables.

6. Conclusion

1. The chapter presents a new class of probabilistic step models for software reliability (and security) assessment which allows to improve the adequacy and accuracy of evaluation for modern multi-version software systems (e.g., open-source software). One of the main features of the developed models is taking into account the effect of reducing the degree of reliability when updating programs.

   These mathematical models have undergone a detailed study and lead to a method that allows planning and monitoring the level of software reliability at the stages of preliminary testing, trial operation, acceptance testing, inspection, and testing after modifications. Completeness and consistency of the method is ensured by the fact that the developed models do not impose strict limitations on the taxonomy of errors, modifications, tests, and input data.

2. The results of the proposed version of the test process modeling can be used at different stages of the software life cycle and integrated into various systems for modeling the reliability and safety of software. To do this the chapter proposes qualitative and quantitative criteria for selecting software test models.

3. It should be mentioned that in the field of information security the use of mathematical models becomes a mandatory procedure in case of checking the high confidence level of the software. This is determined by the methodology of Common Criteria[5] regulated by ISO/IEC 15408.

In the field of quality and functional safety of software, the application of mathematical models is welcomed to reduce the level of subjectivity in testing using black box method, fuzzing, functional testing, etc. (see the lines of international standards IEC 61508, IEC 61511, and ISO/IEC 33001 and also the Russian new standard GOST R 56939). In this respect, IEC 61508–7:2010[6] is extremely useful because it regulates the relationship between the classes of software testing and the use of formal and semiformal models in detail.

[5] www.commoncriteriaportal.org

[6] IEC 61508–7:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems—Part 7: Overview of techniques and measures.

Author details

Alexey Markov[1]*, Alexander Barabanov[2] and Valentin Tsirlov[2]

*Address all correspondence to: mail@cnpo.ru

1 Bauman Moscow State Technical University, Moscow, Russia

2 NPO Echelon, Moscow, Russia

References

[1] Gokhale SS, Marinos PN, Trivedi KS. Important milestones in software reliability modeling. In: Proceedings of Software Engineering and Knowledge Engineering (SEKE 96); Lake Tahoe; 1996. pp. 345-352

[2] Markov A. Software testing models against information security requirements. Cornell University Library [Internet]. 2013. Available from: http://arxiv.org/ftp/arxiv/papers/1306/1306.1958.pdf [Accessed: February 5, 2018]

[3] Andersson B, Persson M. Software reliability prediction—An evaluation of a novel technique. SEBIT; 2004. p. 32

[4] Bondi AB. Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice. 1st ed. Harlow: Addison-Wesley Professional; 2014. p. 426

[5] Kapur PK, Pham H, Gupta A, Jha PC. Software Reliability Assessment with OR Applications. London: Springer; 2013. p. 548. DOI: 10.1007/978-0-85729-204-9

[6] Karanta I. Methods and problems of software reliability estimation. VTT WP. 2006;**63**:57

[7] Lyu MRT. Software Reliability Theory. John Wiley & Sons Inc.; 2002. p. 43. DOI: 10.1002/0471028959.sof329

[8] Musa JD. More Reliable Software Faster and Cheaper. 2nd ed. New York: McGraw-Hill; 2004. p. 632

[9] Naik S, Tripathy P. Software Testing and Quality Assurance: Theory and Practice. Hoboken: Wiley; 2008. p. 616

[10] Shooman ML. Reliability of Computer Systems and Networks: Fault Tolerance, Analysis and Design. New York: Wiley-Interscience; 2002. p. 560

[11] Subburaj R. Software Reliability Engineering. New York: McGraw Hill Education; 2014. p. 458

[12] Tian J. Software Quality Engineering: Testing, Quality Assurance and Quantifiable Improvement. Hoboken: Wiley-IEEE Computer Society Press; 2005. p. 440

[13] Xie M, Dai Y-S, Poh K-L. Computing Systems Reliability. Models and Analysis. Dordrech: Kluwer Academic Publishers; 2004. 293p. DOI: 10.1007/b100619

[14] Yamada S. Software Reliability Modeling: Fundamentals and Applications. Japan: Springer; 2014. p. 90. DOI: 10.1007/978-4-431-54565-1

[15] Anniprincy B, Sridhar S. Prediction of software reliability using COBB-Douglas model in SRGM. Journal of Theoretical and Applied Information Technology. 2014;**62**(2):355-363

[16] Bubnov VP, Sergeev SA. Non-stationary models of a local server of the automated system for monitoring artificial structures. SPIIRAS Proceedings. 2016;**2**(45):102-115. DOI: 10.15622/sp.45.6

[17] Krymsky VG, Ivanov IV. Application of interval-valued probabilities and unified scheme of non-homogeneous Poisson process models to software failure prognostics. In: Podofillini L, Sudret B, Stojadinovic B, Zio E, Kröger W, editors. Safety and Reliability of Complex Engineered Systems: ESREL 2015. Balkema: CRC Press; 2015. pp. 2403-2411

[18] Tamura Y, Yamada S. Cost optimization based on decision-making and reliability modeling for big data on cloud computing. Communications in Dependability and Quality Management. 2015;**18**(4):5-19

[19] Wang LJ, Hu QP, Xie M. Bayesian analysis for NHPP-based software fault detection and correction processes. In: 2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM); IEEE; 2015. pp. 1046-1050

[20] Zeephongsekul P, Jayasinghe CL, Fiondella L, Nagaraju V. Maximum-likelihood estimation of parameters of NHPP software reliability models using expectation conditional maximization algorithm. IEEE Transactions on Reliability. 2016;**65**(3):1571-1583. DOI: 10.1109/TR.2016.2570557

[21] Zhao C, Qiu J, Liu G, Lv K. Planning, tracking and projecting method for testability growth based on in time correction. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability. 2015;**230**(2):228-236

[22] Teyer TA, Lipow M, Nelson EC. Software Reliability. A Study of Large Project Reality, TRW Systems and Energy. Amsterdam/Lausanne/New York: Elsevier; 1978. p. 326

[23] Markov A. Nonmonotone models of reliability and security of software in the early stages of testing. Voprosy kiberbezopasnosti [Cybersecurity Issues]. 2014;**2**(3):10-17. DOI: 10.21681/2311-3456-2014-2-10-17 (in Russia)

[24] Lloyd DK, Lipow M. Reliability Management, Methods, and Mathematics. 2nd ed. Milwaukee: American Society for Quality; 1984. p. 589

[25] Gnedenko B, Pavlov IV. Ushakov IA. Statistical Reliability Engineering, New York: Wiley-Interscience; 1999. p. 528

[26] Junhong G, Xiaozong Y, Hongwei L. Software reliability nonlinear modeling and its fuzzy evaluation. In: 4th WSEAS International Conferernce on Non-Linear Analysis, Non-Linear Systems and Chaos (NOLASC'05); 27–29 October 2005; Sofia: ACM; 2005. pp. 49-54

[27] Kumar R, Khatter K, Kalia A. Measuring software reliability: A fuzzy model. ACM SIGSOFT Software Engineering Notes. 2011;**36**(6):1-6. DOI: 10.1145/2047414.2047425

[28] Vorobiev EG, Petrenko SA, Kovaleva IV, Abrosimov IK. Organization of the entrusted calculations in crucial objects of informatization under uncertainty. In: Proceedings of 2017 20th IEEE International Conference on Soft Computing and Measurements (SCM 2017); 24–26 May 2017; St. Petersburg: IEEE; 2017. 17039917. DOI: 10.1109/SCM.2017.7970566

[29] Iqbal J, Quadri SMK. Software reliability simulation: Process, approaches and methodology. Global Journal of Computer Science and Technology. 2011;**1**(8):1-8

[30] Utkin LV, Zatenko SI, Coolen FPA. New interval Bayesian models for software reliability based on non-homogeneous Poisson processes. Automation and Remote Control. 2010; **71**(5):935-944. DOI: 10.1134/S0005117910050218

[31] Danilov AI, Khomonenko AD, Danilov AA. Dynamic software testing models. In: Proceedings of International Conference on Soft Computing and Measurements (SCM 2015); 19–21 May 2015; St. Petersburg: IEEE; 2015. pp. 72-74. DOI: 10.1109/SCM.2015.7190414

[32] Ivannikov V, Gaissaryan S, Avetisyan A, Padaryan V, Leontyev H. Dynamic analysis and trace simulation for data parallel programs in the parjava environment. In: Avances en la Ciencia de la Computacion (ENC'04); Colima; 2004. pp. 481-488

[33] Ivutin AN, Larkin EV, Perepelkin DA. Software errors and reliability of embedded software. In: 2016 IEEE Conference on Quality Management, Transport and Information Security, Information Technologies (IT&MQ&IS); 4–11 October 2016; Nalchik: IEEE; 2016. pp. 69-71. DOI: 10.1109/ITMQIS.2016.7751926

[34] Kostogryzov A. Modeling software tools complex for evaluation of information systems operation quality (CEISOQ). Lecture Notes in Computer Science. 2001;**2052**:90-101. DOI: 10.1007/3-540-45116-1_12

[35] Smagin VA, Novikov AN, Smagin SY. A probabilistic model of the control of technical systems. Automatic Control and Computer Sciences. 2010;**44**(6):324-329. DOI: 10.3103/S0146411610060027

[36] Rana R, Staron M, Berger C, Hansson J, Nilsson M, Meding W. Analyzing defect inflow distribution and applying Bayesian inference method for software defect prediction in large software projects. Journal of Systems and Software. 2016;**117**:229-244. DOI: 10.1016/j.jss.2014.08.033

[37] Stieber HA. Estimating the total number of software faults reliability models and mutation testing a Bayesian approach. In: 2015 IEEE 39th Annual Computer Software and Applications Conference; 1–5 July 2015. Taichung: IEEE; 2015. pp. 423-426. DOI: 10.1109/COMPSAC.2015.180

[38] Bisi M, Goyal NK. Artificial Neural Network Applications for Software Reliability Prediction, Performability Engineering Series. Wiley-Scrivener; 2017. p. 303

[39] Kaswan KS, Choudhary S, Sharma K. Software reliability modeling using soft computing techniques: Critical review. Journal of Information Technology and Software Engineering. 2015;**5**:144. DOI: 10.4172/2165-7866.1000144

[40] Maevsky D, Kharchenko V, Kolisnyk M, Maevskaya E. Software reliability models and assessment techniques review: Classification issues. In: 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS); 21–23 Sept. 2017; Bucharest: IEEE; 2017. pp. 894-899. DOI: 10.1109/IDAACS.2017.8095216