This paper reports an analysis of the software (SW) safety testing techniques, as well as the models and methods for identifying vulnerabilities. An issue has been revealed related to the reasoned selection of modeling approaches at different stages of the software safety testing process and the identification of its vulnerabilities, which reduces the accuracy of the modeling results obtained. Two steps in the process of identifying software vulnerabilities have been identified. A mathematical model has been built for the process of preparing security testing, which differs from the known ones by a theoretically sound choice of the moment-generating functions when describing transitions from state to state. In addition, the mathematical model takes into consideration the capabilities and risks of the source code verification phase for cryptographic and other ways to protect data. These features generally improve the accuracy of modeling results and reduce input uncertainty in the second phase of software safety testing. An advanced security compliance algorithm has been developed, with a distinctive feature of the selection of laws and distribution parameters that describe individual state-to-state transitions for individual branches of Graphical Evaluation and Review Technique networks (GERT-networks). A GERT-network has been developed to prepare for security testing. A GERT-network for the process of checking the source code for cryptographic and other data protection methods has been developed. A graphic-analytical GERT model for the first phase of software safety testing has been developed. The expressions reported in this paper could be used to devise preliminary recommendations and possible ways to improve the effectiveness of software safety testing algorithms.

Keywords: software, security testing, graphic-analytical model, cyber threats, software safety, data protection.

1. Introduction

Ensuring the security of computer systems, in the face of increased cyber-attacks, is associated with the need to conduct prompt and accurate control over the level of safety of their software (SW). The relevance of this issue is predetermined by the shift of the vector of interests of cybercriminals in the direction of the information and software component of computer systems (CSs), as well as a significant increase in possible losses in the case of cyber threat implementation involving the software and information support of CSs.

Studies have shown that one of the immediate mechanisms for controlling software safety is the methods and tools to identify vulnerabilities. At the same time, one should note that the process of identifying software threats has a series of drawbacks. This is limited application scope, low speed, and incomplete control over the actual state of software, low reliability of the results of vulnerability detection, etc. In many ways, these negative factors are caused by the lack of attention of developers to the issues related to the reasoned selection of testing procedures, as well as the models and methods of identifying vulnerabilities.

Various models, methods, and procedures of security testing and certification are used to identify software vulnerabilities. Most of them are based on software safety requirements set by international industry standards, as well as software testing models (including penetration testing).

At the same time, most of the software safety testing models in the IT-service market have drawbacks. Most of them are related to the lack of attention to the fuzzy factor of input, neglect of the capabilities of attackers in the process of cryptographic change of code, etc.

Therefore, it is a relevant task to improve software safety testing models.
steps in modeling and task setting. At the same time, a
generalized view of the testing process, without taking into
consideration the specificity of software testing, does not
allow for the full consideration of most of the security factors
required in the models.

Work [2] gives a classification of model-oriented soft-
ware safety testing methods. A series of recommendations
for the use of automated tests and risk-oriented testing
approaches are illustrated. That could give researchers a
conceptual apparatus of the expediency of model-oriented
software testing methods. However, the cited work does not
consider studying the mathematical formalization methods
for the implementation of the proposed models.

The drawback mentioned in work [2] could be eliminated
using paper [3]. In it, the authors, along with the generalized
classification of model-oriented software testing methods, con-
sider a series of approaches of mathematical modeling. Those
include finite state machines, state diagrams, unified modeling
language (UML), as well as Markov chains. In addition, the cit-
ed paper provides an example of the implementation of a secur-
ity testing model based on the method of finite state machines.
However, the emergence of new testing factors (the fuzzy input,
the possibility of cryptographic change, the increased tech-
nological capabilities of software development participants) requires
that other approaches of mathematical formalization and improvement of existing models should be considered.

safety testing models. A criterion for assessing the effective-
ness of testing has been proposed. However, the authors take
more into consideration the human factor of cyber threats,
while neglecting the technical components of the assess-
ment. That reduces the adequacy of the models.

Work [5] proposes a generalized model of security testing
in the form of a systematic process map. The possibilities
to categorize software safety risks have been described, as
well as building the tables to prioritize the tasks on minimiz-
ing these risks. Despite the wide range of potential coverage
of software safety threats and systemic nature in meeting
the targets, the cited work does not take into consideration
individual factors. For example, the fuzziness of software
data, ambiguity in the initial knowledge about the methods
and ways of its development.

Paper [6] proposes a vulnerability detection model based
on firmware logic machines. In this case, the authors reduced
the task of mathematical modeling to the synthesis of a con-
trol firmware machine as part of an adaptive-control module
of the system of identifying vulnerabilities in an unstable
network environment.

The model given in [6], along with the merits described
by its authors (operationality, completeness, accuracy), has
obvious flaws caused by the choice of the basic technology
of problem-solving:
- low adaptability of models to actual changes in the behavior of the system;
- significant complications of implementation algorithms
in the event of a possible slight change in the behavior of at
least one site (agent);
- neglecting the issues of possible cryptographic soft-
ware protection.

It should be noted that the elimination of these short-
comings is associated with the use of intelligent modeling
methods.

For example, the results of studying a model of the neu-
ral network of CS safety testing are reported in work [7].
Choosing a modern implementation technology in the
form of a multi-cloud system with a virtual data center
combined with the capabilities of intelligent data process-
ing has made it possible, according to the authors, to im-
prove the level of security. However, a lack of attention to
input formation in the early preparatory stages of testing,
eglect of the problems of possible hiding or obfuscation
of software at this stage, significantly reduce the practical
value of the results.

The authors of [8] justify the effectiveness of the devel-
oped model with a reasoned choice of input characteristics
and the results of comparative experiments conducted with a
wide range of methods of intelligent mathematical modeling.
At the same time, the authors could not eliminate the short-
comings associated with the low speed of neural network
learning processes at this stage of the development of intelli-
gent methods of mathematical formalization.

Work [9] attempts to theoretically justify the choice of
the mathematical apparatus of network modeling. It re-
ports GERT-network research using a unified description
approach based on Erlang’s distribution. At the same time,
those models were built without taking into consideration
the specificity of the full software safety testing cycle, the
possibilities of pre-training software safety testing, in gen-
eral, are neglected, and the existing risks of cryptographic
obfuscation of code, in particular, are neglected.

The authors of [10] tried to solve the problem of increas-
ing the adaptability of the resulting mathematical model.
The cited paper reports a model for identifying software
vulnerabilities using network methods of mathematical
formalization. However, the lack of a practical application
and adaptation of the proposed model to specific types of
cyberattacks reduces the value of development and calls into
question the accuracy of the simulation results.

The authors of [11] tried to eliminate this flaw. Howev-
er, the lack of theoretically sound proposals for the use of
methods of selecting the basic mathematical characteristics
of the description of probabilistic processes (mathematical
expectation, variance, etc.) of individual transitions from
state to state in the system reduce the accuracy of modeling.

Work [12] reports a study into and taxonomy of auto-
matic software safety testing tools. The cited work pro-
vides up-to-date practical recommendations on the use of
these means and tools. However, the clear practical aspect
of the work did not allow the authors to raise the issues of
theoretical justification and mathematical description of
the methods and means of automatic testing of software
safety. Thus, it appears appropriate to conduct a study
to improve the accuracy of the results of mathematical
formalization of the vulnerability detection process, by
taking into consideration the possibility of different stag-
es of software testing for penetration in the mathematical
model built.

3. The aim and objectives of the study

The aim of this study is to build an alternative mathe-
atical model of the first stage of software safety testing
that meets the adequacy requirements and provides a
solution to the task of evaluating the effectiveness of the
algorithm.

To accomplish the aim, the following tasks have been set:
- to develop an improved security compliance algorithm;
– to design the GERT-networks for the security testing process;
– to design the GERT-networks for the process of source code verification for cryptographic and other ways to protect data;
– to develop the GERT-models for the first phase of software safety testing.

4. The study materials and methods

Graphic-analytical methods of mathematical formalization based on the postulates of GERT-network modeling were used to solve the set tasks. At the same time, taking into consideration the features of individual stages of the software safety testing process was based on the provisions and methods of probability theory and mathematical statistics.

5. The results of building a mathematical model

5.1. Software vulnerability identification process model

The process of identifying software vulnerabilities can be conditionally broken down into two phases: preparation for research; conducting the research.

Fig. 1 shows a scheme of the preparation for testing research. The goal of the first phase is to analyze, prepare test documentation, conduct a preliminary analysis, and assess the level of software safety.

A distinctive feature of this phase of research is the introduction of additional source code verification procedures for cryptographic and other ways to protect the code. That would make it possible to highlight the data format, make changes to the relevant test documentation, and prepare additional means of analyzing the encryption (obfuscation) of the data.

Fig. 1 demonstrates that, unlike many processes related to computer system status control, it is not possible to set many controlled parametric data and reference values in detecting software vulnerabilities. The inputs for the examination in the presented model are elements of the software itself, the software environment, as well as technical documentation and standards.

These data are subjected to preliminary analysis; the result of such analysis is a research plan, test documentation sets, and non-parametric data for the implementation of the test control bench. An exception to this rule, in the first stage of threat detection, is an analysis of the structure of the assessed object, which results not only in non-parametric data for the emulation of components but also data of the preliminary control of the assessed object.

The result of this assessment may be the object-oriented properties of a software product. At the same time, experts could use a variety of controlled data (control results) set by the following indicators as a characteristic of structural security:

– size;
– the hierarchical structure;
– the connectivity of the relevant parts;
– polymerization;
– the complexity of sharing information.

Fig. 1. Security testing study preparation flow chart
Combined with confirmation of encryption (obfuscation) of the software, this set forms a separate class of input parametric data for the second stage – conducting research on software safety testing.

This class of input data when assessing software safety is a fuzzy set. This fact imposes some limitations in the second phase of software safety testing (direct research).

In order to improve the effectiveness of software vulnerability detection processes, as well as to improve the accuracy of decision-making, there is a need for theoretically reasoned formalization and mathematical formalization of most of the components of the above stage.

5.2. Improved security compliance algorithm

Consider a problem on identifying software vulnerabilities in terms of matching security probabilities to regulatory requirements. In this case, we introduce the assumption that some generalized safety indicator $X_0$ must meet the requirement:

$$X_0 \leq X_{\text{add}} \tag{1}$$

where $X_{\text{add}}$ is a regulatory measure of safety.

It is not difficult to notice that (1) formalizes a deterministic example of a study where a safety indicator is described by a clear value. However, we can assume that there is uncertainty in the value of $X_0$. Therefore, $X_0$ can be considered a mathematical expectation $X_0 = \mathbb{E}[X]$ of the random value $X$, distributed under one of the known distribution laws with probability density $f(X)$ and distribution function $F(X)$.

Paper [13] illustrates a generalized case of uncertainty about the value of the safety indicator in the form shown in Fig. 2. At the same time, as the figure demonstrates, the general mathematical expression describing the situation when the random value does not exceed the maximum allowable value of $X_{\text{add}}$:

$$P(X \leq X_{\text{add}}) = 1 - P(X > X_{\text{add}}) = 1 - F(X_{\text{add}}) \tag{2}$$

At the same time, the probability of exceeding the allowable value of the $X_{\text{add}}$ is defined as:

$$P_{\text{add}} = P(X > X_{\text{add}}) = 1 - P(X \leq X_{\text{add}}) = 1 - F(X_{\text{add}}) \tag{3}$$

In addition, work [12] gives the algorithms to check compliance with the security criteria for different distribution laws.

However, one of the main drawbacks of this formalization approach is the need to initially define mathematical expectation and the variance of random $X$:

$$M[X] = X_0, \quad D[X] = S_0^2 \tag{4}$$

This can only be used for small examples of mathematical formalization and not for the entire spectrum of technical systems, taking into consideration the accuracy of the result.

Therefore, there is a need to improve this algorithm, taking into consideration the processes shown in Fig. 1.

It is known from work [14] that this type of modeling has a number of advantages:

- it is an effective way to determine previously unknown laws and functions of random distribution under a known algorithm of functioning (process);
- it is easy to implement;
- the results of mathematical modeling are adequate, etc.

The formalization model can be represented as follows:

Stage 1. Select laws and distribution parameters that describe individual state-to-state transitions for the individual branches of GERT-networks. At this stage:

1. Calculate the $X_0$ safety score.
2. Register the $X_{\text{add}}$ value.
3. Check condition (1). If condition (1) is met, proceed to p. 4.
4. Register uncertainty parameters; considered $X_0$ as an estimate of the average random $X$ value, distributed by some law with a probability density of $f(X)$.
5. Calculate $P_{\text{add}}$ using formula (3).
6. Decide whether the received $P_{\text{add}}$ value is acceptable or unacceptable for the probability of exceeding an $X_{\text{add}}$ value.

Stage 2. Develop a GERT-network scheme to prepare for security testing research based on the data shown in Fig. 1 and their actualization by a reasoned choice of the moment functions of each branch of transitions from state to state.

Stage 3. Find the equivalent functions of the distribution of the processes described and study them.

We shall develop and investigate a GERT-model of the process of preparing for safety testing research.

5.3. GERT-model of the first phase of identifying software vulnerabilities

5.3.1. Studying the probability density function for different distribution laws

Using the “R Project for Statistical Computing” mathematical software package, we constructed a dependence of both probability density functions and distribution functions (of the five considered) for which $P_{\text{add}} = f(x_0, s_0, x_{\text{add}})$. As an example, the probability density functions at $x_0=3, s_0=1, x_{\text{add}}=4$ are shown in Fig. 3.

In accordance with (1) to (3), $P_{\text{add}}$ values for different distribution laws (Table 1) have been obtained.

As Table 1 demonstrates, under experimental conditions, the lowest $P_{\text{add}}$ value was obtained when using, for the mathematical formalization, the gamma-distribution, log-normal, and normal distribution laws.

We shall use these test results in the mathematical formalization of the software safety testing process. We adopt a normal distribution law as the basic law of distribution when describing individual steps and transitions.
5.3.2. Developing a GERT-network scheme of the process to prepare for security testing

The process of identifying software vulnerabilities can be considered as a network GERT-structure, the input of which is the flow of tasks that need to be solved, and the ultimate goal is the flow of tasks accomplished. In this case, one should consider that any initial task can be decomposed into smaller sub-tasks, which generally speeds up the process of simplifying GERT-network transformations. In the end, this decomposition would describe a set of several single tasks, such as implementing individual methods (the single task is considered to be the one that a specialist could perform in one working day). Next, a queue of single tasks is formed. This line, according to different software development methodologies, is broken down into either several iterations (according to Agile procedures) or more complicated complex iterative structures (in accordance with the “spiral” methodology).

The task software team identified typical service characteristics based on the properties of the input flow of data, the parameters and structure of the system, and the disciplines of query service. The main characteristic of the system is the probability that the development team will successfully complete all the tasks set in the iteration.

We present a generalized mathematical model of identifying vulnerabilities in the form of GERT-networks. The first phase is formalized as a GERT-network in the security testing process (Fig. 4).

This model can be interpreted as follows. Node 1 corresponds to the initial state of “the required documentation package, source and executable codes have been collected”.

Node 2 interprets the state of “the source code check for cryptographic and other ways to protect data completed”. Node 3 is a state of “the readiness of test documentation and test control bench”. Node 4 corresponds to the “ready for pre-assessment of a tested object” status. Node 5 is the state of “the preliminary results of object structure analysis are ready”.

The corresponding branches of the model are interpreted by the mathematical formalization of transitions from state to state. In particular, the transition (1–2) formalizes the process of checking source code for cryptographic and other ways to protect data. Transitions (1–3) and (2–3) correspond to the process of preparing test documentation and test control bench. Transition (3–4) formalizes the process of pre-evaluation of the tested object (architecture, completeness of information, etc.), as well as planning of the study. Transition (4–5) characterizes the process of analyzing the structure of the object. Transition (4–2) describes procedures for returning to the state of assessment of possible cryptographic or other ways of encoding software. Transition (4–1) formalizes the necessary process of additional input collection if necessary (insufficient).

The equivalent W-function of the security testing prepara

\[
W_f(s) = W_{11}W_{12}W_{13}W_{14}W_{15} + W_{21}W_{22}W_{23}W_{24}W_{25}W_{26} - W_{31}W_{32}W_{33}W_{34}W_{35} - W_{41}W_{42}W_{43}W_{44}W_{45} - W_{51}W_{52}W_{53}W_{54}W_{55}
\]  

(5)

A distinctive feature of the model being built is to take into consideration the source code verification for cryptographic and other ways to protect the data. This procedure in Fig. 4 is represented by transition (1–2). We shall describe the procedure in more detail.

5.3.3. GERT-network of source code verification process for cryptographic and other ways to protect data

Our study has shown that the process of cryptographic conversion or obfuscation of the source code of software can be represented as a combination of algebraic operations of weighted addition and multiplication, performed in accordance with the following expressions:
where $S$ is the result of a weighted addition operation; $R$ is the set of reversible transformations; $p_i$ is the probability of selecting such $R_i$ systems in which $K_i$ – the set of initial messages and $L_i$ – the set of converted messages are equal; $C$ is the result of a multiplication operation, for which the equality of the $R_i$ system values and the set of determining the $R_{i-1}$ system is a prerequisite.

These algebraic ratios can be formalized as equivalent transformations and GERT-network transitions.

In practice, in the SW encryption or obfuscation processes, the selection of successive operations is done using a random number sensor. The GERT-model of these processes makes it possible to analyze the probabilistic behavior of the software hiding (transformation) system and could be used to estimate the number of options that need to be sorted out when testing software safety for cryptographic transformation.

It should be noted that the most important characteristics of software safety include the average and variance in the number of conversions performed, as well as the average time and variance of operations. We shall consider methods to find these characteristics using an example of the software encryption scheme shown in Fig. 5.

![Fig. 5. Software encryption scheme for a generalized GERT-network](image)

Let us find an average of the number of conversions that are being performed. As a basis, we shall take the exponential law of distribution of the random value of the time of the transformation and, accordingly, the moment-generating functions of branches are equal to $e^{p_i}$. Then the equivalent $W$-function of the GERT-encryption network of software is equal to

$$W_k(s) = \frac{q_1 q_2 e^{k s}}{1 - (p_1 + p_2)e^s + p_1 p_2 e^{2s} - q_1 q_2 p_1 e^{2s}},$$

where $q_1 = 1 - p_1$, $q_2 = 1 - p_2$, $q_3 = 1 - p_3$ are the probabilities of branch selection $(1, 2), (2, 3), (3, 4)$ in the scheme shown in Fig. 5, respectively.

Expression (6) can determine the average number of $N$ conversions performed and its variance $D_N$. If one indicates the probability of passing loops of the first kind $(1, 2, 3, 4, 1); (1, 2, 3, 1)$ via $\gamma_1 = q_1 q_2 q_3$ and $\gamma_2 = q_1 q_3 q_2$, respectively, and the product of probabilities of the loop of the second kind $(1, 1), (2, 2)$ – via $\gamma_3 = p_1 p_2$, one can determine

$$N = \frac{3\gamma_1 + 2\gamma_2 - \gamma_3 + 1}{\gamma_1},$$

$$D_N = \frac{(2\gamma_2 - \gamma_3 + 1)^2 - \gamma_1 (1 - 4\gamma_2 + \gamma_3)}{\gamma_1^2}.$$  \hspace{1cm} (8)

Using the Mathcad specialized mathematical package, we shall calculate some combinations of $q_1$ – $q_3$ probabilities and the corresponding $N$ and $D_N$ values. The results are given in Table 2.

Table 2

<table>
<thead>
<tr>
<th>No. of entry</th>
<th>$q_1$</th>
<th>$q_2$</th>
<th>$q_3$</th>
<th>$N$</th>
<th>$D_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>211</td>
<td>41.49</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>56</td>
<td>20.96</td>
</tr>
<tr>
<td>3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>26.536</td>
<td>13.743</td>
</tr>
<tr>
<td>4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>16</td>
<td>8.94</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>11</td>
<td>7.25</td>
</tr>
<tr>
<td>6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>8.222</td>
<td>5.307</td>
</tr>
<tr>
<td>7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>6.51</td>
<td>3.724</td>
</tr>
<tr>
<td>8</td>
<td>0.8</td>
<td>0.8</td>
<td>0.8</td>
<td>5.375</td>
<td>2.36</td>
</tr>
<tr>
<td>9</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
<td>4.58</td>
<td>1.134</td>
</tr>
<tr>
<td>10</td>
<td>0.999</td>
<td>0.999</td>
<td>0.999</td>
<td>4.005</td>
<td>0.011</td>
</tr>
<tr>
<td>11</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>43</td>
<td>6.21</td>
</tr>
<tr>
<td>12</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
<td>29.667</td>
<td>9.097</td>
</tr>
<tr>
<td>13</td>
<td>0.5</td>
<td>0.1</td>
<td>0.1</td>
<td>27</td>
<td>13.05</td>
</tr>
<tr>
<td>14</td>
<td>0.7</td>
<td>0.1</td>
<td>0.1</td>
<td>25.837</td>
<td>17.156</td>
</tr>
<tr>
<td>15</td>
<td>0.9</td>
<td>0.1</td>
<td>0.1</td>
<td>25.222</td>
<td>21.312</td>
</tr>
<tr>
<td>16</td>
<td>0.1</td>
<td>0.9</td>
<td>0.1</td>
<td>25.222</td>
<td>21.312</td>
</tr>
<tr>
<td>17</td>
<td>0.3</td>
<td>0.9</td>
<td>0.1</td>
<td>11.889</td>
<td>10.137</td>
</tr>
<tr>
<td>18</td>
<td>0.5</td>
<td>0.9</td>
<td>0.1</td>
<td>9.222</td>
<td>8.561</td>
</tr>
<tr>
<td>19</td>
<td>0.7</td>
<td>0.9</td>
<td>0.1</td>
<td>8.079</td>
<td>8.357</td>
</tr>
<tr>
<td>20</td>
<td>0.9</td>
<td>0.9</td>
<td>0.5</td>
<td>7.444</td>
<td>8.61</td>
</tr>
</tbody>
</table>

We shall use (9) to find the equivalent $W$-function of the source code verification process for cryptographic and other ways to protect the data.
For the cases set in Table 2 (case 1 – \(q_1=0.1\), \(q_2=0.4\), \(q_3=0.4\), case 2 – \(q_1=0.3\), \(q_2=0.1\), \(q_3=0.5\). For both cases, \(D_\text{N}=9\), \(k_1=k_2=k_3=2\)), we shall obtain the \(W\)-function of \(R\) conversion time:

\[
W_k(s) = e^{(1266s+165.6s^2)}.
\]  

Then the equivalent \(W\)-function of the time of the process of testing a cryptographically converted software product

\[
W_k(s) = 1 - (p_1 + p_2)e^{1.338s+165.8s^2} + p_1p_2e^{2.76s+33.8s^2} + q_1q_2p_1e^{4.14s+56.7s^2}.
\]  

The density of the probability distribution of the time of the process of testing the cryptographically converted software product at different values of \(q_1\), \(q_2\), \(q_3\), and at values \(t_{\text{dol}}=0.6\ s\), \(t_{\text{mal}}=3\ s\), \(t_{\text{sum}}=5\ s\), obtained using the software platform Mathcad is shown in Fig. 6.

Important indicators that characterize the complexity of software testing for cryptographic transformations (obfuscation) are the average time and variance of cryptographic or obfuscation operations.

Fig. 6 demonstrates that the maximum distribution density of the random time of the test time for cryptographic transformations (obfuscation) is reached within 0.13–0.15 s. The estimated variance values showed the following results for case 1 – 0.009, for case 2 – 0.002.

\[\text{Fig. 6. The density of probability distribution of the time to execute a process of testing a cryptographically converted software product}\]

It should be noted that the analytical expressions derived from the simulation, as well as the values obtained experimentally, could be used in the examined GERT-network of the security testing process preparation. This is an important component step in modeling.

5.4. GERT-model of the security testing process preparation

Taking into consideration the results obtained in subchapter 5.1–5.3, as well as expression (5), we shall find an analytical ratio for calculating and researching the equivalent \(W\)-function of the software safety testing process.

\[\text{Table 3}
\]

<table>
<thead>
<tr>
<th>No. of entry</th>
<th>Branch</th>
<th>(W)-function</th>
<th>Probability</th>
<th>Moment-generating function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1, 2)</td>
<td>(W_{12}) ((\text{expression } 6))</td>
<td>(p_1)</td>
<td>(\lambda_1/\lambda_1-s)</td>
</tr>
<tr>
<td>2</td>
<td>(1, 3)</td>
<td>(W_{13})</td>
<td>(p_1)</td>
<td>(\lambda_2/\lambda_2-s)</td>
</tr>
<tr>
<td>3</td>
<td>(2, 3)</td>
<td>(W_{23})</td>
<td>(p_1)</td>
<td>(\lambda_2/\lambda_2-s)</td>
</tr>
<tr>
<td>4</td>
<td>(3, 4)</td>
<td>(W_{34})</td>
<td>(p_2)</td>
<td>(\lambda_2/\lambda_2-s)</td>
</tr>
<tr>
<td>5</td>
<td>(4, 5)</td>
<td>(W_{45})</td>
<td>(p_3)</td>
<td>(\lambda_3/\lambda_3-s)</td>
</tr>
<tr>
<td>6</td>
<td>(4, 2)</td>
<td>(W_{42})</td>
<td>(p_4)</td>
<td>(\lambda_4/\lambda_4-s)</td>
</tr>
<tr>
<td>7</td>
<td>(4, 1)</td>
<td>(W_{41})</td>
<td>(p_5)</td>
<td>(\lambda_5/\lambda_5-s)</td>
</tr>
</tbody>
</table>

Then

\[
W_k(s) = \frac{(W_k(s)+1)(\lambda_3-s)}{(\lambda_3-s)(\lambda_2-s)(\lambda_1-s)}.
\]  

The following expression was obtained to calculate the equivalent \(W\)-function of the software safety testing process:

\[
W_k(s) = \frac{(W_k(s)+1)(gs^2-ks+r)}{(\lambda_3-s)(\lambda_2-s)(\lambda_1-s)}.
\]  

where

\[
g = p_1p_2p_3p_4p_5p_6 \lambda_7 \lambda_8; \]

\[
k = p_1p_2p_3p_4p_5p_6 \lambda_9 \lambda_2; \]

\[
v = p_1p_2p_3p_4p_5p_6 \lambda_9 \lambda_1; \]

\[
r = (W_k(s)+1)(p_1p_2p_3p_4p_5p_6\lambda_8)+(2p_1p_2p_3p_4p_5p_6\lambda_8) \lambda_9 \lambda_{12}; \]

\[
y = (W_k(s)+1)(p_1p_2p_3p_4p_5p_6\lambda_8)+2p_1p_2p_3p_4p_5p_6 \lambda_8 \lambda_9; \]

\[
d = (W_k(s)+1)(p_1p_2p_3p_4p_5p_6\lambda_8)+(2p_1p_2p_3p_4p_5p_6\lambda_8) \times \lambda_9 \lambda_2 \lambda_3 + 2\lambda_9 \lambda_2 \lambda_4 \lambda_6 + \lambda_9 \lambda_2 \lambda_5 \lambda_7 - \lambda_2 - \lambda_9 \lambda_4 \lambda_6 \lambda_7 \lambda_9 \lambda_2 \lambda_5 \lambda_7 - \lambda_9 \lambda_2 \lambda_3 \lambda_4 \lambda_5 \lambda_6 \lambda_7 - (W_k(s)+1)(p_1p_2p_3p_4p_6\lambda_8) \lambda_9 \lambda_{12}; \]

\[
x = (-\lambda_2-\lambda_3) \lambda_5 \lambda_9 \lambda_2 \lambda_3 \lambda_4 \lambda_5 \lambda_7 \lambda_9 \lambda_2 \lambda_4 \lambda_6 \lambda_7 \lambda_9 \lambda_2 \lambda_5 \lambda_7 \lambda_9 \lambda_2 \lambda_3 \lambda_4 \lambda_5 \lambda_6 \lambda_7 \lambda_9 \lambda_2 \lambda_5 \lambda_7 \lambda_9 \lambda_2 \lambda_3 \lambda_4 \lambda_5 \lambda_6 \lambda_7 \lambda_9 \lambda_2 \lambda_5 \lambda_7 \lambda_9 \lambda_2 \lambda_3 \lambda_4 \lambda_5 \lambda_6 \lambda_7 \lambda_9 \lambda_2 \lambda_5 \lambda_7.
The density of the distribution of software safety testing probability time takes the following form:

$$\Phi(x) = \frac{1}{2\pi i} \int_{-\infty}^{\infty} \frac{e^{\alpha\Phi(z)}}{(z^2 - yz^2 + dz^2 - Hz^2 - xz^2 + cz + b)} dz,$$

where integration is performed along the Bromwich contour [15].

The choice of integration technique can be made depending on whether the function $\Phi(z)$ has only simple poles or poles of some order. In the example above, the $e^{\alpha\Phi(z)}$ expression can be represented as:

$$e^{\alpha\Phi(z)} = \frac{e^{\alpha \left(gz^2 - Hz + \alpha\right)}}{\psi(z)},$$

where $\psi(z) = z^2 - yz^2 + dz^2 - Hz^2 - xz^2 + cz + b$.

Then the density of the time of security testing of all types of software (including cryptographically protected) is:

$$\Phi(x) = \frac{\sum_{i=1}^{n} \text{Res} \left(e^{\alpha \Phi(z)} \right)_{z=z_i}}{\psi(z)}$$

$$= \frac{\sum_{i=1}^{n} \text{Res} \left(e^{\alpha \left(gz^2 - Hz + \alpha\right)} \right)_{z=z_i}}{\psi(z)}.$$

The function $\Phi(z)$ can have a pole of the second or third order. Then the density of the distribution of the transfer time $\varphi(z)$ is calculated from the formula of finding $r_1$ deducions from the $z_i$ poles of order $n$:

$$r_1 = \frac{1}{(n-1)!} \lim_{z \to z_i} \frac{d^{n-1}}{dz^{n-1}} \left(\frac{\left(gz^2 - Hz + \alpha\right)}{(z^2 - yz^2 + dz^2 - Hz^2 - xz^2 + cz + b)} \right).$$

Expression (18) is a fractional-rational function relative to $z$ with a denominator power greater than the numerator's one. That is why the conditions of Jordan's lemma [15] are met for it.
The polynomial $r^2g^3dz^4-hz^3-xz^2+cz+b$ generates seven poles. The solution to the following equation
\[
r^2 - gy^3 - dz^4 - hz^3 - xz^2 + cz + b = 0. \tag{19}
\]
can be found by any method, for example, by Viet’s formulas [14]. As a result, special points $z_1, z_2, z_3, z_4, z_5, z_6$ are calculated.

A number of experiments have been conducted to substantiate the validity of the modeling results, in accordance with the following conditions:

- a team of software developers consists of 8 people; one is an automated tester and one – a Person Non Grata tester;
- the basic methodology for managing software development is SCRUM;
- the number of experiments is $N^*=100$.

The test results were used to build a histogram of the time of preparation for the software safety testing, shown in Fig. 8.

Our hypothesis of the normal distribution of this random value has been tested by the Pearson agreement criterion $\chi^2$ [15].

\[
\chi^2 = N \sum_i (P_i - P) \hat{P} / P_i,
\]
where $k$ is the number of bits (intervals) of the statistical series; $P_i$ and $P$ is the "statistical" and theoretical probability of "matching" the preset indicator with the $i$-th bit.

Our test proved the plausibility of the hypothesis that the amount of software safety testing time is distributed according to normal law.

The following estimates of the mathematical expectation $\hat{t}_{test}^{(0)}$ and variance $\hat{D}_{test}^{(0)}$ ( $\hat{\sigma}_{test}^{(0)}$ is the rms deviation) of the random value $t_{test}$ for a software safety testing time have been obtained:

\[
\hat{t}_{test}^{(0)} = \frac{1}{N} \sum t_{test}, \quad \hat{D}_{test}^{(0)} = \frac{1}{N-1} \sum (t_{test} - \hat{t}_{test}^{(0)})^2, \quad \hat{\sigma}_{test}^{(0)} = \sqrt{\hat{D}_{test}^{(0)}},
\]

\[
P \left( \frac{t_{test}^{(0)} - \hat{t}_{test}^{(0)}}{\hat{\sigma}_{test}^{(0)}} < 1 \right) = 2K \left( 0.05 \hat{\sigma}_{test}^{(0)} \right),
\]
where $\Phi$ is the Laplace function in the form
\[
\Phi(x) = \frac{1}{\sqrt{2\pi}} e^{x^2/2} dt \ [15].
\]

The results of our experiments showed that for all the data studied, the confidence probability that the statistical value $\hat{t}_{test}^{(0)}$ “does not deviate” from the mathematical expectation $t_{test}^{(0)}$ by more than 0.05 is $P=0.94$.

The experimental data have made it possible to conduct a comparative study of the results from mathematical modeling. The results of the comparison are shown in Fig. 9 in the form of a chart of the density of the distribution of the time probabilities $t_{test}$ of the software safety test and the corresponding boundaries of the confidence interval:

\[
I_{\beta} = [\hat{f} - \epsilon_{\beta}, \hat{f} + \epsilon_{\beta}],
\]

in which the true value $\hat{f}$ falls with a confidence probability of $\beta=0.94$ and estimates of its mathematical expectation $\hat{t}_{test}^{(0)}$.

Fig. 9 demonstrates that in a key test situation (time $t_{test}=0.15$ hour/person) the “calculated” curve $f$ (solid curve), obtained in accordance with the constructed mathematical model, in most practical cases fall into the "average" confidence interval (shaded area).

This confirms the validity of the built mathematical model of the first phase of software safety testing and the analytical expression, resulting from our mathematical modeling, which formalizes the distribution of security testing time for all types of software.

6. Discussion of results of studying the mathematical model of the first stage of software safety testing

Thus, based on the GERT-network formalizing technology, a mathematical model of the security preparation process has been constructed. The mathematical model differs from those known by the theoretically sound choice of moment-generating functions when describing transitions from state to state, as well as taking into consideration...
the stage of checking the source code for cryptographic methods of protection.

A series of stages in the developed model of the security preparation process relate to solving one of the current problems, improving the accuracy of the results of mathematical modeling. Thus, the improved “security compliance check” algorithm, described in chapter 5.2, has reduced input uncertainty by adding the second and third steps. This was made possible by the reasoned choice of the distribution law with a minimum probability value \( P_{\text{add}} \) for exceeding the allowable value of the regulatory measure of security \( X_{\text{add}} \) at each transition of the developed GERT-network. Unlike prototype models [6, 9, 10], that has made it possible not to introduce unreasonable assumptions about the random distribution law for the testing process in general. In addition, the security compliance algorithm avoids unreasonable assumptions about mathematical expectation and variance.

The GERT-network scheme for security testing, proposed in chapter 5.3.2, was designed to take into consideration possible destabilizing factors in additional software coding. It also improves the accuracy of the results of mathematical formalization. It should be noted that neglecting the threat of obfuscation cryptographic software coding reduces the quality of software safety testing and, in practice, could lead to irreversible consequences for computer systems.

The security compliance algorithm and the GERT-network scheme of the security preparation process reported in this paper are the components of the GERT-model described in chapter 5.4. Developing a GERT-model of the security preparation process provided an analytical expression to calculate the density of the probability of software safety testing time. That, in turn, has made it possible to estimate the time indicators of security testing for variations in the intensity of the tester’s activities at:

- analyzing the documentation and other materials containing information about the assessed object;
- checking a source code for cryptographic and other ways to protect data;
- preparing test documentation and a test control bench;
- pre-assessing the tested object (for architecture, completeness of information, etc.) and planning research;
- analyzing the structure of the object.

Thus, the main advantage of our model is to improve the accuracy of the results by reasonably choosing the GERT-network approach of mathematical formalization, the justified use of the distribution law at each stage of network formalization, as well as taking into consideration the factor of SW coding.

It should be noted that the mathematical model of the security testing process obtained by GERT-network formalization is informative and could provide a clear, accessible for direct analysis dependence of the performance indicators of the testing algorithm on the values of the algorithm’s statistical characteristics. The expressions presented in this paper could be used to make preliminary recommendations and possible ways to improve the effectiveness of software safety testing algorithms.

Restrictions on the use of the devised model are associated with the presence of input information in the form of software code or its emulation. In addition, the mathematical model presented is relevant when examining the initial stage of software safety verification. This imposes a preliminary restriction on their use to implement automated software safety tests.

Possible areas of further research involve the need to develop the second phase of software safety testing, taking into consideration the uncertainty of the initial data that may be described vaguely. The challenges that arise could be solved on the basis of the methods proposed in [9].

7. Conclusions

1. A GERT model for the first phase of software safety testing has been developed. The model differs from those known by the theoretically sound choice of moment-generating functions when describing the transitions from state to state, as well as taking into consideration the initial code verification phase for cryptographic protection methods. That could improve the accuracy of the software safety test results, as well as use the results in the overall software testing process.

2. An advanced security compliance algorithm has been developed. This algorithm differs from those known by considering the uncertainty parameters when selecting the moment-generating functions of each branch of transition from state to state of the GERT-network being developed. This could reduce the uncertainty of inputs during the development phase of the GERT-network preparation process for software safety testing research.

3. A GERT-network has been developed to prepare for the security testing process. Its distinctive feature is the accounting of a source code verification for cryptographic and other ways to protect the data. That could improve the accuracy of the modeling results in the face of this type of cyber abuse.

4. A GERT-network has been developed to check a source code for cryptographic and other ways to protect data. Analytical expressions have been obtained and the data used in the GERT-model of the software safety testing process have been experimentally calculated.

References