SOFTWARE CONTRIBUTION TO THE AVAILABILITY OF MICROPROCESSOR-BASED RELAY PROTECTION

M. I. Uspensky
•
Komi SC UB RAS, Syktyvkar, Russian Federation.
uspensky@energy.komisc.ru

Abstract

An important characteristic of relay protection functioning is availability of microprocessor relay protection software. An approach to estimation of such parameter and correlation between it and hardware availability on the example of 110/35/10 kV distribution network microprocessor protection is considered in the paper. The behavioral nature of the availability under research, reasons and a share of various kinds of the error leading to failure of program execution, variants of program volume definition, some solution approaches to the task at hand, including methods of Jelinsky-Moranda, and also examples of assessing the ratio of these availabilities are considered. An algorithm for the software evaluation used is presented. The influence of different conditions on such evaluation is shown. Applications of different approaches to software readiness estimation for the above types of protection based on data during debugging of protection programs are given.

***Key words*:** reliability, availability, software, relay protection module.

I Introduction

The reliability index is an important characteristic of relay protection and automatics (RPA) functioning. Many authors, including us [1, 2], noted that such characteristic of modern digital protections is convenient to divide into components: hardware or technical reliability, connected with failure (destruction) of relay protection device elements; traffic reliability, defined by temporary loss or distortion of data without failure of process bus element; program reliability due to errors in development of execution programs; and resistance to external purposeful influence on transmitted information. In [3], the behavior of the first component on the reliability indicator was given and shown by the example of the 110/35/10 kV distribution network protection system. Here we will consider the approach to software reliability characterization, and on the example of the same system, we will evaluate the contribution of this component to the total availability of the aforementioned protections.

II Specifics of software reliability

It is known that software failure is associated with its inadequacy to the set tasks. There are many definitions of software failure. Most definitions of a software error come down to [4]: *Software reliability is the probability that a program will work without failures for a certain period of time taking into account the degree of their influence on the output results*. The frequency of errors from statistical data, reduced to 100% errors is given in Table 1, and the position "Incomplete or erroneous task" is disclosed in more detail.

On the one hand, software is not subject to wear and tear and its reliability is determined only by development errors. Thus, this indicator should increase with time, if correction of detected errors does not introduce new errors. On the other hand, many programmers' experience shows that in a large software, no matter how much you test it, some errors will remain. Due to the testing that simulates almost all the real modes, the errors of incorrect software operation are corrected, but there always remains a set of data that occurs due to some, usually external conditions, for example, interference or erroneous human actions, which cannot be foreseen and which will lead the software to work incorrectly. The next dilemma to solve here is how to optimize the quality/cost ratio so as not to lose market priority, or customer confidence. It is important to remember that we are here examining the readiness of the software to work.

Table 1. Frequency of occurrence of some error types [4]

Cause of error	**Frequency**, %
Task deviation	12
Ignorance of programming rules	10
Erroneous data sample	10
Erroneous logic or operation sequence	12
Erroneous arithmetic operations	9
Insufficient time to solve	4
Improper interrupt handling	4
Incorrect constants or input data	3
Inaccurate writing	8
Incomplete or erroneous assignment	28
⇓	
Errors in numerical values	12
Insufficient accuracy requirements	4
Erroneous characters or symbols	2
Mistakes in the design	15
Incorrect description of hardware	2
Incomplete or inaccurate design basis	52
Ambiguity of requirements	13

The manifestation of an error in the software system is reflected in a failure situation, which leads the program either to a hang (stopping while waiting for the next command, which does not really exist) or to incorrect calculations, leading to erroneous actions.

The specificity of relay protection programs is that often the application programs are prepared in the languages of programmable logic controllers (PLCs) [5], which reduces the probability of program errors. However, the operating environment is written in more traditional software languages such as C, Java, etc. A system of programs written in different programming languages when estimating its reliability, is reduced to the average assembler equivalent per 1000 lines through "KAELOC - K of Assembler Equivalent Lines of Code", where K is 1000 lines of code [6] (see Table 2).

Table 2. Conversion factors

Programming language	Factor
Assembler, macroassembler	1
C	2.5
C++	11
Fortran	3
Pascal	3.5
LISP	1.5
Ada	4.5
Forth	5
Query languages (like SQL)	25
Object-oriented 4th generation languages	16
PLC languages	10 … 33

Basically, software bugs are tried to remove when writing and debugging, and a lot of programs are created to detect bugs at the debugging stage. But it is expected that some (small) number of errors is present in the program. The detection programs are tuned for specific external conditions (which group of people prepares the program under test, the temperature and electromagnetic environment, etc.) What to do with the remaining errors? 1. The salesperson continues to test and identify errors, which are corrected in customers. 2. Buyers identify bugs and turn them over to the creators for correction. 3. Change the vendor.

III Evaluating the software's contribution to availability by programming averages

A fairly rough estimate of software availability can be determined as follows [7]. For responsible applications, which include the RPA software, by the time the system is delivered to the client it may contain from 4 to 15 errors per 100 000 lines of program code [8]. For illustration, let us note that the number of code lines of WINDOWS XP is over 45 million, the NASA program is 40 million, the Linux 4.11 kernel is over 18 million. If we estimate the complex of simultaneously working RPA programs at 1 million code lines, the number of errors at the beginning of software operation $E = (V/100\,000) \cdot 15 = 150$ errors. Then, using the formula of average software MTBF, we get

$$\lambda_{SW} = \beta \frac{E}{V} = 0.01 \frac{150}{10^6} = 1.5 \cdot 10^{-6} \text{ or } t_{SW} = \frac{1}{\lambda_{SW}} = \frac{10^6}{1.5 \cdot 8760} \approx 76 \text{ years}, \quad (1)$$

where E is the number of errors per complex of jointly working programs accepted for operation, V is the size of the complex in code lines, β is the program complexity factor, usually in the range of 0.001...0.01, λ_{SW} is the failure rate and t_{SW} is the MTBF of software, 8760 is the number of hours per year. The size of the RPA application programs is most often limited to thousands of assembler lines because of the requirement for their speed. Then, at the value of 15 errors per 100 000 code lines, adopted for the application software after testing with the volume of code lines $E = 4000 \cdot 15/100{,}000 = 0.6$ errors

$$\lambda_{SW} = \beta \frac{E}{V} = 0.01 \frac{0.6}{4000} = 1.5 \cdot 10^{-6} \text{ or } t_{SW} = \frac{1}{\lambda_{SW}} = \frac{6.67 \cdot 10^5}{8760} \approx 76 \text{ years} \quad (2)$$

or about one failure per 76 years. With a recovery time of t_r = 2 h $A_{SW} = \frac{\mu}{\lambda+\mu} = \frac{4380}{1.5 \cdot 10^{-6} + 4380} = 0.9999999997$.

IV Software contribution to availability according to the Jelinsky-Moranda model

There are a number of models of reliability growth concerning the process of failure detection [9, 10]. The classification of such models divides them into two groups: models that consider the number of failures as a Markov process; models that consider the failure rate as a Poisson process. Let us use the model of the second group.

The Jelinsky-Moranda model is based on the following assumptions: 1) the time to the next failure is exponentially distributed; 2) the failure rate of a program is proportional to the number of errors remaining in the program.

This model assumes that the time elapsed between failures follows an exponential distribution with a parameter that is proportional to the number of remaining errors in the software. Figure 1 shows a stepped curve characteristic of program failure rate changes as a function of its model run time. It can be seen that as each error is detected, the degree of risk decreases by proportionality constant. This indicates that the impact of each fault correction is the same.

According to these assumptions, the probability of program failure as a function of time t_i is

$$P(t_i) = e^{-\lambda_i t_i}, \quad (3)$$

where the failure rate is

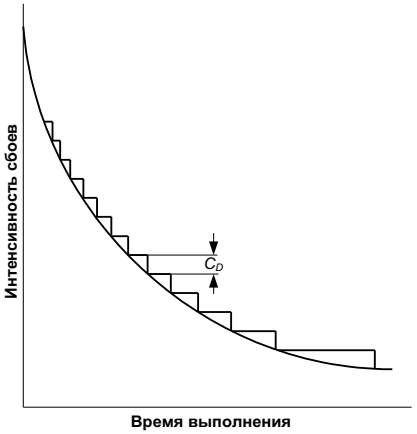


Fig. 1. In the Jelinsky-Moranda model, the failure rate curve decreases from constant CD.

$$\lambda_k = C_D[E_0 - (k-1)]. \quad (4)$$

Here E_0 is the initial number of errors, k is the number of the last observed program failure/fault, C_D is the proportionality factor. The time countdown starts from the penultimate (k – 1) program failure. The disadvantage of the model is that it assumes complete elimination of errors after it detection without introducing new errors.

From model (3) and the maximum likelihood method we can write

$$F = \prod_{i=1}^{k-1} C_D(E_0 - i + 1)\, e^{-C_D(E_0-i+1)t_i}, \quad (5)$$

or logarithmic likelihood function

$$L = \ln F = \sum_{i=1}^{k-1}\{\ln[C_D(E_0 - i + 1)] - C_D(E_0 - i + 1)\}, \quad (6)$$

wherefrom finding the extremum

$$\frac{\partial L}{\partial C_D} = \sum_{i=1}^{k-1}\left[\frac{1}{C_D} - (E_0 - i + 1)t_i\right] = 0, \quad (7)$$

$$\frac{\partial L}{\partial N} = \sum_{i=1}^{k-1}\left[\frac{1}{E_0-i+1} - C_D t_i\right] = 0. \quad (8)$$

From (8) we get

$$C_D = \frac{\sum_{i=1}^{k-1} 1/(E_0-i+1)}{\sum_{i=1}^{k-1} t_i}. \quad (9)$$

Substituting (8) into (7), we obtain

$$(k-1)\frac{\sum_{i=1}^{k-1} t_i}{\sum_{i=1}^{k-1} 1/(E_0-i+1)} = \sum_{i=1}^{k-1}(E_0 - i + 1)t_i, \quad (10)$$

from which we find E_0 by trying its values. Since E_0 is an integer, we find the minimal difference between the left and right parts of (10). The closest integer value E_0, at which the difference between the left and right parts of formula (6) is minimal, is usually given in the range k – 1...2·k, since the initial number of errors is not less than the known value of the number sum of corrected errors, and the error remaining number is usually not greater than the number of detected errors, i.e. the final total value is equal to the doubled detected value.

The manifestation intensity of the remaining errors of the program is determined. According to the methodology in [11], such intensity is calculated by the formula

$$\lambda_k = \frac{\sum_{j=1}^{k} \frac{j}{E_0 - \sum_{i=1}^{j-1} i}}{\sum_{j=1}^{k} t_j}\left(E_0 - \sum_{j=1}^{k} i\right). \quad (11)$$

But this intensity is bound to the volume of lines with errors. In reality, the failure rate is statistically defined as [12]

$$\overline{\lambda}(t) \approx \frac{m(t)}{n(t)\Delta t}, \quad (12)$$

where $m(t)$ is the number of failed elements (lines with errors) in the considered period Δt, $n(t)$ is the average number of equipment elements (in our case, code lines or program commands) work-

ing in this interval. Therefore, the obtained in (9) intensity should be recalculated to the full volume of lines or commands under study software, i.e.

$$\lambda = \lambda_k \frac{E_0}{N_\Sigma}, \tag{13}$$

where N_Σ is the number of lines under study software.

Assuming a constant error rate in accordance with the Jelinsky-Moranda model concept, we calculate the average time to error in the software:

$$t_E = \frac{1}{\lambda}. \tag{14}$$

When the error detection and correction time is assumed to be 2 hours (μ = 0.5), the software availability coefficient is

$$A_{SW} = \frac{\mu}{\lambda+\mu}. \tag{15}$$

The calculation algorithm is shown in Fig. 2. The initial data of this calculation are:
$N_{\Sigma L}$ – number of program lines in programming languages, which are converted by means of Table 2 into N_Σ – number of commands reduced to assembler codes; N_T – number of executed tests; array $[E_i]$ – number of detected and corrected errors at the i-th stage of testing, which time is determined by the array $[t_i]$. DE traces the minimal discrepancy between the left (LP) and the right (RP) part of the formula (10) in the E_0 search. ΣE_i is the sum of errors known from the tests up to position i. Σt_i is the sum of times between tests up to position i. The variable LA corresponds to the manifestation intensity of the remaining program errors (λ). A_{SW} is the software availability index.

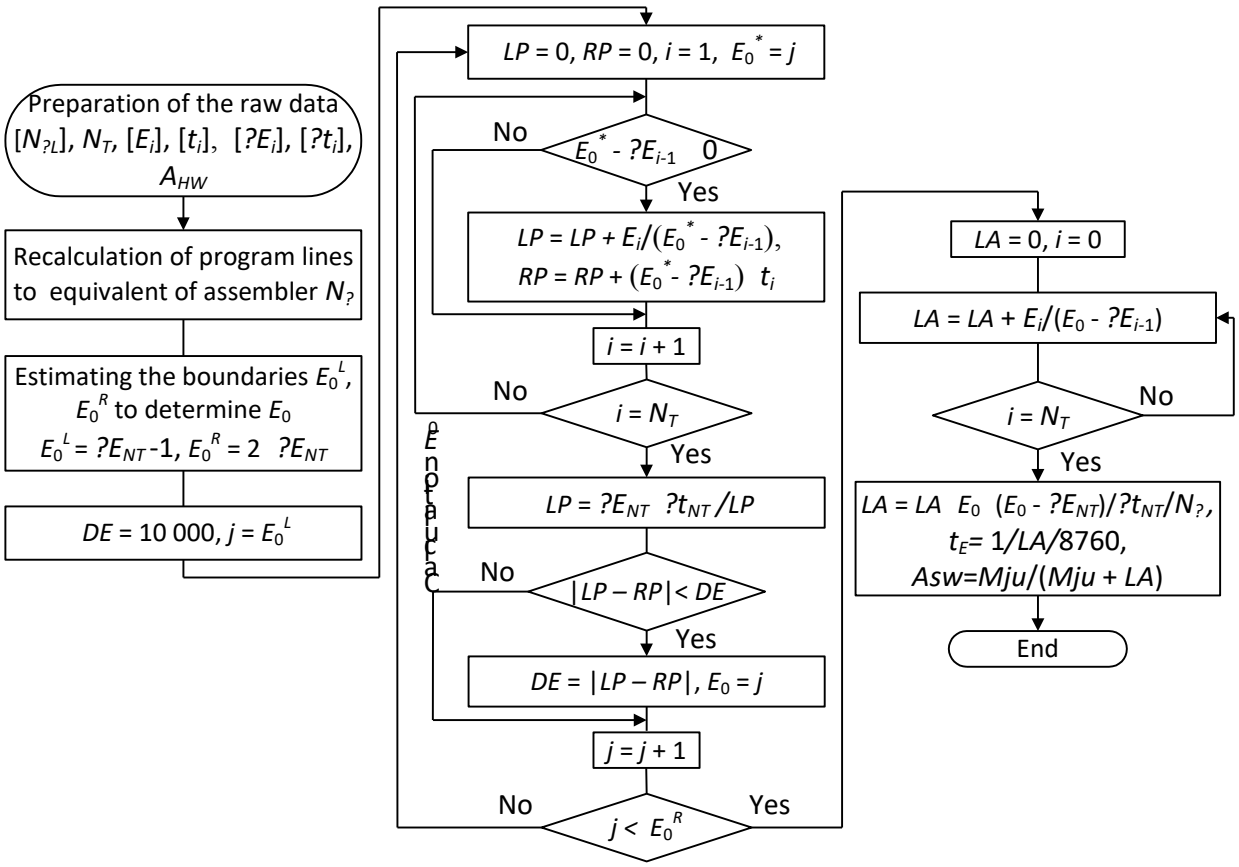


Fig. 2. Calculation algorithm for software availability characteristics.

V Calculation of software contributions to the availability of relay protections

Let's evaluate the software availability of the protection and control modules for the 35 kV bus section and transformer section [3]. The necessary data for the studied modules are presented in Tables 3 and 4. The results of calculations – in Table 5.

Table 3. Transformer section protection and control software module

Initial data	E_i, errors	ΣE_i, errors	t_i, hours	Σt_i, hours
$N_t = 4$	0	0	0	0
$N_\Sigma = 1439_{asm}+200_{C++}$	1	1	77	77
$N_\Sigma = 3639_{asm}$	1	2	63	140
$\mu = 0.5\ h^{-1}$	1	3	7	147
	1	4	187	334
C++ – in codes C++, *asm* – in codes assembler.				

Table 4. 35 kV busbar section protection and control software module

Initial data	E_i, errors	ΣE_i, errors	t_i, hours	Σt_i, hours
$N_t = 3$	0	0	0	0
$N_\Sigma = 1346_{asm}+130_{C++}$	1	1	63	63
$N_\Sigma = 2776_{asm}$	1	2	11	74
$\mu = 0.5\ h^{-1}$	1	3	117	191
C++ – in codes C++, *asm* – in codes assembler.				

Table 5. Calculating results of the software availability characteristics of the modules

Transformer section protection and control module				35 kV busbar section protection and control module				Flexible logic module			
E_0	5	λ, years^{-1}	0.04621	E_0	4	λ, years^{-1}	0.07153	E_0	4	λ, years^{-1}	0.01016
E_i	4	t_E, years	21.64	E_i	3	t_E, years	13.98	E_i	3	t_E, years	98.4
ΔE	1	A_{SW}	0.99998945	ΔE	1	A_{SW}	0.99994558	ΔE	1	A_{SW}	0.99998367

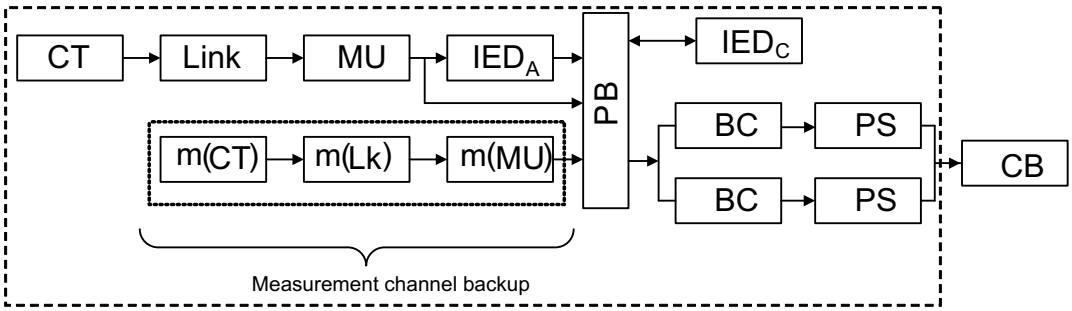
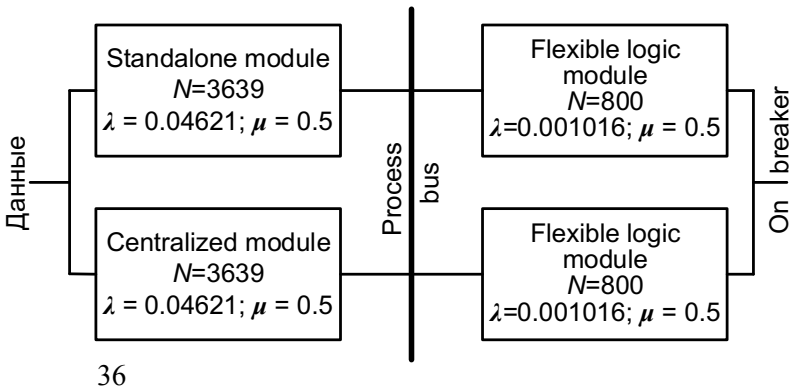


Fig. 3. Reliability block diagram of protection.

According to the protection reliability model (Fig. 3) for the hardware part, presented in [3], and software organization (Fig. 4), we explain that the software part of the model consists of two software protection blocks, autonomous and centralized, included by reliability in parallel with the output to the process bus,



and two parallel blocks of flexible logic program. In the software evaluation of the model, the process bus is not taken into account, because it is taken into account in the hardware.

In accordance with the scheme of Fig. 4 we determine the equivalent failure rate and recovery rate of the corresponding transformer protection programs based on the known relations:
$\lambda_e = \sum_i \lambda_i$; $\mu_e = \lambda_e / \sum_i \frac{\lambda_i}{\mu_i}$ – for series connection and $\mu_e = \sum_i \mu_i$; $\lambda_e = \mu_э / \sum_i \frac{\mu_i}{\lambda_i}$ - for parallel connection. Then for Fig. 4, the equivalent values are:

left-hand side $\mu_{el} = 2 \cdot 0.5 = 1$; $\lambda_{el} = 1/\left(2 \cdot \frac{0.5 \cdot 8760}{0.04621}\right) = 5.2751 \cdot 10^{-6}$ h^{-1};

right-hand side $\mu_{er} = 2 \cdot 0.5 = 1$; $\lambda_{er} = 1/\left(2 \cdot \frac{0.5 \cdot 8760}{0.01016}\right) = 1.1598 \cdot 10^{-6}$ h^{-1};

$\lambda_e = 5.2751 \cdot 10^{-6} + 1.1598 \cdot 10^{-6} = 6.4349 \cdot 10^{-6}$ h^{-1} or 0.05637 years $^{-1}$;

Fig. 4. Reliability model of transformer protection software.

$\mu_e = 6.4349 \cdot 10^{-6} / \left(\frac{5.2751 \cdot 10^{-6} + 1.1598 \cdot 10^{-6}}{0.5}\right) = 0.5$ h^{-1} or 4380 years $^{-1}$.

Consequently, $A_{SW} = \frac{\mu_e}{\lambda_e + \mu_e} = \frac{4380}{0.05637 + 4380} = 0.999987$.

From [3] in the worst case for the hardware model of transformer protection A_{HW} = 0.9999999764, i.e. the contribution to the unavailability of protection from the hardware is essentially less, than from the software, and its total value $A_\Sigma = A_{HW} \cdot A_{SW}$ = 0.9999869, and the average time to failure $t_\Sigma = A_\Sigma / (\mu_e \cdot (1 - A_\Sigma))$ = 17.6 years or 153565 hours.

The scheme of the 35 kV busbar protection software model also corresponds to fig. 4. Equivalent values for the left part: N = 2776 lines of assembler code; λ = 0.007153 years $^{-1}$; μ = 0.5 h^{-1}. The right part is similar to the transformer model. Then the equivalent values of the left part

$\mu_{el} = 2 \cdot 0.5 = 1$; $\lambda_{el} = 1/\left(2 \cdot \frac{0.5 \cdot 8760}{0.07153}\right) = 8.1655 \cdot 10^{-6}$ h^{-1};

$\lambda_e = 8.1655 \cdot 10^{-6} + 1.1598 \cdot 10^{-6} = 9.3253 \cdot 10^{-6}$ h^{-1} or 0.08169 years $^{-1}$;

$\mu_e = 9.3253 \cdot 10^{-6} / \left(\frac{8.1655 \cdot 10^{-6} + 1.1598 \cdot 10^{-6}}{0.5}\right) = 0.5$ h^{-1} or 4380 years $^{-1}$.

Consequently, $A_{SW} = \frac{\mu_e}{\lambda_e + \mu_e} = \frac{4380}{0.08169 + 4380} = 0.999981$.

From [3] in the worst case for the hardware model of busbar protection A_{HW} = 0.999999884, i.e. the contribution to the unavailability of protection also from the hardware is much less, than from the software, and its total value $A_\Sigma = A_{HW} \cdot A_{SW}$ = 0.9999809, and the average time between errors $t_\Sigma = A_\Sigma / (\mu_e \cdot (1 - A_\Sigma))$ = 12.2 years or 106872 hours.

To estimate the contribution of software to the total unavailability of protection, we can use the expression

$$Ctb_{SW} = \frac{n_{E,SW}}{n_{E,SW} + n_{E,HW}} \cdot 100\% = \frac{\lambda_{SW}}{\lambda_{SW} + \lambda_{AHW}} \cdot 100\% = \frac{t_{E,HW}}{t_{E,SW} + t_{E,HW}} \cdot 100\%. \quad (16)$$

Here n_E is the number of failures, t_E is the average time to failure, the index A_{HW} is the availability of the hardware part.

When discussing the results of the presented work, it should be understood that, despite their outward resemblance to a quantitative assessment, they represent only qualitative indicators of software readiness. On the other hand, the obtained results do not take into account the software test control, which improves the studied indicators. Unfortunately, as noted in [13], there are no reliable methods for quantitative software evaluations other than statistics for a significant period of program operation. Nevertheless, they show that software errors can have a significant influence on reliability indicators of microprocessor relay protection.

VI Other approaches to assessing contributions to the availability of relay protections

Let's try to estimate the impact of software on RPA functioning from the following statistics. According to [14], the number of microprocessor-based RPA devices in operation in 2013 was 274062 devices, and in 2014 – 319912 devices (Table 4, [14]). From the data [15] "Distribution of cases of device RPA malfunction by types of technical reasons and device RPA types for the period from 01.01.2020 to 30.06.2020" we know that out of 727 cases of RPA failure 18 cases are related to software failure or malfunction. Then the forecast number of RPA devices for 2020 in relation to 2013 from the formula $d_n = d_1(1+r)^n$ at $r = 100\cdot(d_2 - d_1)/d_1$ %, where d_1 – number of devices of the first year, d_n – number of devices for n year, r – average annual growth of devices, can make

$$d_7 = 274062\left(1 + \frac{319912 - 274062}{274062}\right)^6 = 693328 \text{ devices}$$

Let's take Rosseti's share of RPA as 70% of all devices in [15]. Then a rough estimate of the failure rate $\lambda = \frac{n}{0.7\cdot Nt} = \frac{18\cdot 2}{0.7\cdot 693328} = 7.42\cdot 10^{-5}$ years^{-1}. Here n – the number of devices, failed due to software, for half year (2 in the numerator), $0.7\cdot N$ – the number of all microprocessor protections, t – design period (year). For the recovery time t_r = 2 hours $A_{SW} = \frac{\mu}{\lambda+\mu} = \frac{4380}{7.42\cdot 10^{-5}+4380} = 0.999999983$. And the average time between errors $t_\Sigma = \frac{1}{\lambda} = 13477$ years, which is of course unreal. From the relation $Ctb_{\text{ПО}} = \frac{n_{E,SW}}{n_{E,SW}+n_{E,HW}}\cdot 100\% = \frac{18}{727}\cdot 100\% \approx 2.5\%$ we will note, that the share of failures because of program errors was 2.5%.

One more approach on the basis of data of work [16] where at small sample a share of failures because of software errors in total number of failures can be estimated as (3+1+3+4)/(11+15+18+17) =11/61·100% =18%, where in numerator failures because of software, and in denominator - total failures. Of course, the small sample does not allow us to confidently judge the representativeness of the figures, but, nevertheless, some idea of the ratio is given.

VII Conclusion

The approach according to formulas (1) and (2) gives quite a large uncertainty range, depending on the choice of the error content coefficient per 100 thousand lines of code and the program complexity coefficient. Its result can be considered as an upper bound of A_{SW} under the chosen conditions. An estimation of A_{SW} contribution values showed that software unavailability was 1.3% of the total unavailability.

The Jelinsky-Moranda reliability model can be considered a lower bound for A_{SW} since the initial conditions are more restrictive here.

Calculations of software availability with the Jelinsky-Moranda reliability model showed that the main unavailability of the considered protections is determined by software unavailability, which was 99.8% for transformer protection and 49.8% for busbar protection. Nevertheless, even in this case, the average total error time is more than 150 thousand hours for transformer protection and more than 100 thousand hours for 35 kV busbar protection.

In contrast to the calculated data from statistics [14,15] showed that the error rate due to software is about 2.5%, and from [16] - 18%.

The work was carried out within the framework of the theme "Models and methods of adaptation of power systems in modern conditions".

References

1. Morozov Yu.M. Reliability of hardware and software systems. St. Petersburg, 2011. 136 p. . (In Russian).
2. Uspensky M.I. Contribution of Hardware, Software, and Traffic to the WAMS Communication Network Availability // Reliability: Theory & Applications Vol. 15, No 3. 2020, pp.70-83. DOI:https:// doi.org/10.24411/1932-2321-2020-13007
3. Uspensky M.I. Reliability Assessment of the Digital Relay Protection System // Reliability: Theory & Applications Vol. 14, No 3. 2019, pp. 10-17. DOI: https://doi.org/10.24411/1932-2321-2019-13001.
4. Shklyar VN. Reliability of control systems. Tomsk, Russia: Publishing house of Tomsk Polytechnic University. 2009;126 p. (In Russian).
5. Livshits, Yu. E. Programmable Logic Controllers for Process Control / Minsk: BNTU, 2014, Ch. 1, 206 p. (In Russian).
6. Baranov S.P., Domaratsky A.N., Lastochkin N.K., Morozov V.P. Defects prevention during software products creation // Software Products, #1, 2000, pp. 59-63. (In Russian).
7. Borovikov SM, Dik SS, Fomenko NK. A method for predicting applied software tools at the early stages of their development// Reports of the Belarusian State University of Informatics and Radioelectronics. 2019, #5, pp. 45-51. (In Russian).
8. Chukanov VO, Gurov VV, Prokopyeva EV. Methods of ensuring software and hardware reliability for computing systems// Russia, Presentation of the report at the seminar, pp. 1-44. Available: http://www.mcst.ru/files/5357 ec/dd0cd8/50af39/000000/seminar_metody_ obespecheniya_apparatno-programmnoy_nadezhnosti_vychislitelnyh_sistem.pdf (In Russian). (accessed 12.03.2019)
9. Bubnov V. P., Safonov V. I., Shardakov K. S. Review of existing models of nonstationary service systems and methods of their calculation // Control, Communication and Security Systems. 2020, # 3, pp. 65-121. DOI: 10.24411/2410-9916-2020-10303. (In Russian).
10. Vasilenko N.V., Makarov V.A. Software Reliability Assessment Models // Bulletin of Novgorod State University, 2004, # 28, pp. 126-132. (In Russian).
11. Iyudu K.A. Reliability and diagnostics of computing machines and systems: Textbook on special "Computing machines, complexes, systems and networks" / M.:Vyssh. shk. 1989, 216 p. (In Russian).
12. Shalin A.I. Reliability and diagnostics of relay protection of power systems. Novosibirsk: Publishing house of NSTU, 2002, 384 p. (In Russian).
13. Littlewood B., Strigini L. "Validation of ultra-high dependability…" – 20 years on // BL-LS-SCSS newsletter2011_02_v04distrib.pdf, 5 p. Available: http:// www.staff.city.ac.uk
14. Concept for the Development of Relay Protection and Automation in the Electric Grid Sector // Appendix # 1 to Rosseti's Management Board Protocol # 356pr dated June 22, 2015. M.,2015, 49 p. Available: https://mig-energo.ru›wp-content/uploads/2015/12/rza-fsk.pdf. (In Russian).
15. Distribution of malfunctions of RPA devices by types of technical reasons and types of RPA devices for the period from 01.01.2020 to 30.06.2020// Available: https://www.so-ups.ru/ fileadmin/files/ company/rza/rza_rez_info/rza_rez_vid_teh_1-2k2020.xls. (In Russian).
16. Zakharov O. G. Reliability of Digital Relay Protection Devices. Indicators. Requirements. Estimates. Moscow: Infra-engineering, 2018, 128 p. (In Russian).