RELIABILITY FACTORS OF SOFTWARE FOR MICROPROCESSOR PROTECTIONS

M. I. Uspensky
•
Institute for Socio-Economic & Energy Problems of the North of FRC Komi Science Center,
Ural Branch, Russian Academy of Sciences, Syktyvkar, Russia
uspensky@energy.komisc.ru

Abstract

The features of the program functioning reliability that are typical for critical applications operating in real mode for microprocessor protections are considered. Among the main characteristics of the relay protection functioning are reliability indicators. With the transition to the execution of such protections on a microprocessor-based basis, in addition to hardware reliability, it became necessary to characterize its operation by software reliability. The importance of the solve tasks in the operation process refers it to programs used in safety-critical software and operating in real mode. This, in turn, tightens the requirements for their reliability evaluation. Comparative analysis made it possible to assess the generality of the considered reliability factors for the functioning of the application software under consideration. At the same time, we were able to highlight some of the features that are typical for the implementation of relay protection on microprocessors. An example of such an assessment is given, showing that despite all the difficulties of complete testing for microprocessor protection programs the software erroneous contribution still amounts to 2.5% of the total contribution. It is shown that microprocessor protection programs are related to programs of critical applications operating in real mode, which makes it possible to use the experience and characteristics of such software in solving relay protection challenges. Nevertheless, some features are given that are specific for relay protection based on a microprocessor. Further tasks are defined.

Key words: safety-critical software, microprocessor relay protection, software reliability.

1. INTRODUCTION

Recently, programmers have been paying a lot of attention to assessing the reliable operation of programs used in mission-critical applications and operating in real mode. Among a whole range of such programs that perform important functions in nuclear power plants, for military equipment, etc., a large share of them is involved in control and protection of electric power system (EPS) equipment. Such software is required to perform its functions with high reliability.

In the engineering community, software systems have a reputation for being unstable. There have been cases where programmers' mistakes have led to major accidents with large financial losses. So on 14.08.2013 on the east coast of the USA, 55 million people were without electricity, because the operator of the power plant turned off the alarm about the violation, so as not to interfere with the work, corrected the mode and forgot to turn on the alarm, and the program did not remind about it, and the next violation of the mode led to a major accident, because the alarm about it was not received. Because the designers forgot to cross the date line, 12 fighters on their way from the United States to Okinawa lost access to fuel data, speed and altitude sensors, and communications were partially disrupted. For hours, America's most advanced F-22 Raptor fighters flew across the ocean completely helpless. In the end they were only able to land thanks to the skill of their pilots. And there are hundreds of similar examples.

2. SPECIFICS OF SAFETY-CRITICAL SOFTWARE

So why don't engineers avoid software if it is not trustworthy? There are usually three main advantages of replacing hardware with software:

1. Software technology allows more logic to be built into the system. Software-controlled computer systems can distinguish between a large number of situations and produce outputs corresponding to each situation. Hard logic systems could not achieve this behavior without an exorbitant amount of hardware. Programmable hardware is less expensive than equivalent hardwired logic because it is regularly structured and mass produced. The economics of the situation also allow program-controlled systems to perform more checks; reliability can be increased by periodically running programs that check the hardware.
2. On the other hand, the cost of writing and certifying truly reliable software is very high; in addition, maintenance costs must be taken into account – again, one that does not undermine reliability and security. An illustrative example: only maintenance of relatively simple and not very big in volume (about 400 thousand words) software for the onboard computer installed on the American spacecraft, type Shuttle, has cost NASA 100 million dollars per year [1].
3. The logic implemented in the software is theoretically easier to change than the logic implemented in the hardware. Many changes can be made without adding new components. If the system is replicated or in a hard to reach physical location, it is much easier to make changes to the software than to the hardware.
4. On the other hand, changes to software modules are easy to make technically, but difficult to do without introducing new bugs. The verification and certification required for security guarantees mean new high costs. In addition, the longer the lifetime of a program, the greater the danger of introducing bugs along with changes – for example, because some developers cease to be such over time, and documentation is rarely exhaustive. Meanwhile, the scale of changes in software can be very large. For example, the Shuttle software underwent 14 modifications over 10 years of maintenance since 1980, which resulted in 152,000 lines of code changes (the total software volume is 400,000 lines). The need for software upgrades was dictated by periodic hardware upgrades, adding functionality, and by the need to correct identified defects.
5. Computer technology and software flexibility allow more information to be made available to operators and provided in a more useful form. The operator of a modern software-controlled system can be provided with information that would be unthinkable in a purely hardware system. All of this can be accomplished with less space and power than was used in noncomputerized systems [2].

In considering reliability relative to software, it should be remembered that:

- The most obvious difference between software and hardware technologies is their complexity. Thus, accurate documentation in fairly general notation for small software systems can fill a bookcase, whereas for hardware a few diagrams and a brief description are sufficient. Another indicator of complexity is the time it takes the programmer to become intimately familiar with the system. Even with small software systems, it is often the case that it takes a programmer a year to work with a program before he or she can be trusted to make improvements on his or her own.
- The next notable property of software is its sensitivity to small errors. In conventional mechanical engineering, every design and manufacturing measurement can be characterized by

a tolerance. One is not required to be an exact match; it is sufficient to be within the specified tolerance of the desired value. The use of tolerances is justified by the assumption that small errors have small consequences. It is well known that in software, trivial clerical errors can have the most serious consequences. For software, there is no known useful interpretation of a tolerance. A single punctuation error can be catastrophic, even though fundamental missteps sometimes have minor consequences.

- Software is notoriously difficult to adequately test. It often happens that a piece of software, subjected to careful and disciplined testing, is seriously flawed. Analog device testing is based on interpolation. It is assumed that devices that perform well at two close points will also perform well at intermediate points. In software, this assumption is wrong. The number of cases that must be tested in order to generate confidence in the software is usually very large.
- Many of the assumptions that are commonly made when designing high-reliability equipment are invalid for software. Developers of high-reliability equipment are concerned about manufacturing failures and wear and tear phenomena. They may conduct their analysis under the assumption that failures are not highly correlated and simultaneous failures are unlikely. Those evaluating the reliability of hardware systems would have to be concerned about design errors and correlated failures; however, in many situations, the effects of other types of errors are dominant. Software has few errors introduced at the production (compilation) stage; when such errors do exist, they are systematic rather than random. Software does not wear out. The errors that software reliability specialists must deal with are design errors. These errors cannot be considered statistically independent. There is ample evidence that even when programs for a given task are written by people who do not know each other, they have closely related errors [3, 4].

These properties are a fundamental consequence of the fact that the mathematical functions implemented by the software are not continuous functions, but functions with an arbitrary number of discontinuities. The lack of continuity constraints on functions describing software effects makes it difficult to find compact descriptions of the software. The lack of such constraints gives software flexibility, but also complexity. Similarly, sensitivity to small errors and difficulty in testing can be attributed to fundamental mathematical properties; a miracle cure is unlikely to be found. Reliability-critical software systems will always require a great deal of discipline and scrutiny.

In contrast to the situation with hardware systems, you cannot achieve higher reliability by duplicating software components. Errors are simply duplicated. Even if programs are written independently, errors made by one programmer are often shared by others. As a result, one cannot expect to improve the reliability of software systems only by having three computers where one is enough, although even here everything is not so simple.

It is appropriate to recall that in the practice of developing and using software, great importance is attached to solving the problem of its safety by executing a standard series of the IEC 61508-N-2010 "«Functional safety of electrical/electronic/programmable electronic safety-related systems", where N = 1, …, 4. This series details the goals and requirements for managing the safety of functioning, including programmable systems. But the actual filling of these requirements remains with the software developers.

3. FEATURES OF RELAY PROTECTION PROGRAMS ON MICROPROCESSOR

Let us consider application of the above properties and peculiarities to relay protection programs on microprocessor which also refer to reliability-critical software systems.

Applied program modules of relay protection are characterized by relatively small volume of programs (in average 3-5 thousand lines in assembler equivalent), that is determined, first of all, by requirements to speed of protection. On the other hand, such volumes create a good visibility of the program text, which reduces the appearance of design errors. This also includes the fact that in some cases such application programs are prepared in the languages of programmable logic controllers (PLCs) [5], which also reduces the likelihood of program errors. However, the operating environment is written in more traditional software languages, such as C, Java, etc.

Another characteristic of these modules is related to the fact that this or that violation of the mode of EPS components can be detected in different ways on the basis of the same input data and often on different microprocessors. In traditional protection, this approach is known as redundancy. In microprocessor protection, it allows duplicating software when processing the input data, because the algorithms of their processing are not repeated. As a result, at such organization the restriction on duplication of programs is to some extent weakened, because such organization of their work is rather a redundancy than a duplication, although in reliability modeling they are included in parallel in terms of task performance (detection of an unacceptable mode). Of course, here too, each of the programs may have its own errors, which are not detected and are not similar to each other, but this is closer to the duplication of the hardware part of protection.

And of course it is important that the protection system is not loaded with some auxiliary background tasks, which can be a source of errors and crashes. It should be as autonomous as possible if we consider it a critical application. The tendency to combine several protection tasks on a single processor leads to an increase in the size of the software, hence to a worse visibility of the programs, and hence to an increase in the probability of software errors.

Thus, various probabilistic assessments of reliability indicators of relay protection software functioning, similar to hardware assessments, give only a qualitative picture in a rather wide range. Quantitative assessment can be judged only from statistics, which appears after years of operation of a significant number of devices.

So according to statistical data, the number of microprocessor-based relay protection and automatic devices (RPAD) in operation in 2013 were 274062 devices, and in 2014 - 319912 devices ([6], Table 4). From the data "Distribution of cases of malfunction of RPAD devices by types of technical reasons and types of RPAD devices for the period from 01.01.2020 to 30.06.2020" [7] we know that from 727 cases of RPAD failure 29 cases are related to failure of microprocessor protections hardware, and 18 cases - to failure or failure of their software. Then the forecast number of relay protection devices for 2020 in relation to 2013 (in 7 years) can be made from the formula

$$a_n = a_1(1 + r)^n$$

with $r = (a_2 - a_1)/a_1$, where $a_1 = 274062$ is the number of devices of the first year (2013), $a_2 = 319912$ (2014) is the number of devices of the next year, a_n is the number of devices for year n (2020), r is the average annual growth of devices, then

$a_7 = 274062\left(1 + \frac{319912-274062}{274062}\right)^7 = 809321$ devices.

Let's take Rosseti's share of RPAD to be 70% of all devices in [7]. Then a rough estimation of intensity of failures $\lambda = \frac{m}{0.7 \cdot N \cdot t} = \frac{18 \cdot 2}{0.7 \cdot 809321} = 6.35 \cdot 10^{-5}$ years–1. Here m - the number of device failures due to software, for half a year (2 is multiplier up to a year), 0.7·N - the number of all Rosseti's microprocessor protections, t - design period (year). When the recovery time tr = 2 h

($\mu = \frac{1}{t_r} = \frac{8760}{2} = 4380$ years–1),

software availability factor

$$K_{av.sw} = \frac{\mu}{\lambda + \mu} = \frac{4380}{6.35 \cdot 10^{-5} + 4380} = 0.999999985$$

And the average operating time before the error

$$t_e = 1/\lambda = 15748 \text{ years}$$

The proportion of errors due to software from the number of all protection failures is

$$Pr_{sw} = \frac{m_{sw}}{m_{sw} + m_{oth}} \cdot 100\% = \frac{18}{727} \cdot 100\% \approx 2.5\%$$

where $m_{sw} = 18$ is failures of devices due to software, $m_{sw} + m_{oth} = 727$ all cases of incorrect operation of microprocessor protections. 29 of them are related to hardware. Note that the proportion of failures due to software errors was 2.5%.

According to the data of work [8] at small sampling a error share of software in the total number of failures can be estimated, as (3+1+3+4) / (11+15+18+17) = 11/61·100% = 18%, where in numerator - failures of microprocessor devices because of software, and in denominator - total failures. Of course, the small sample does not allow to judge confidently about the representativeness of the figures, but nevertheless some idea of the ratio is given.

4. CONCLUSIONS

Microprocessor-based protection programs refer to the programs of critical applications operating in real mode, and therefore they are required to perform their functions with high reliability. Given the development and widespread use of computer technology and the flexibility of microprocessor-based protection software, we must not forget the complexity of their description, sensitivity to even small deviations from the basic algorithm, testing problems associated with the lack of continuity limitations of the functions of programs describing the algorithms of solutions, which does not contribute to the design with errors that have little effect on the performance of the required functions.

There are some properties of these programs that improve their reliability performance, such as brevity, programmable logic controller languages, but this is not enough to solve the reliability problems of microprocessor protection programs as programs, critical applications. On the other hand, sufficient autonomy of individual protection devices is necessary, because the desire for comprehensiveness often reduces the reliability parameters of protection. Here it is necessary to look for optimal solutions, taking into account both the functionality of programmable protections, and the achievement of the required characteristics of the reliability of the execution of these functions.

The work was done within the framework of the topic AAA-A20-120051590026-3 "Models and methods of adaptation of power engineering systems in modern conditions".

References

[1] An Assessment of Space Shuttle Flight Software Development processes. - Committee for Review of Oversight Mechanisms for Space Shuttle Flight Software Development Processes, National Research Council, 1993. 206 p. https://ntrs.nasa.gov/api/citations/19930019745/downloads/19930019745.pdf

[2] Parnas D.L., Schouwen A.J., Kwan S.P. Evaluation of Safety-Critical Software // Communications of the ACM. 1990, Vol. 33, No. 6. Pp. 636-648

[3] Bishop P.G., Pullen F.D. Error Masking: A Source of Failure Dependency in Multi-Version Programs / 1991, January. 17 p. DOI: 10.1007/978-3-7091-9123-1.3.

[4] Eckhardt D.E. et al. An experimental evaluation of software redundancy as a strategy for improving reliability / D.E. Eckhardt A.K. Caglayan J.C. Knight L.D. Lee D.F. McAllister J.P.J. Kelly // IEEE Transactions on Software Engineering. Vol.17, Is.7, 1991. Pp.692-702. DOI: 10.1109/32.83905

[5] Livshits, Yu. E. Programmable Logic Controllers for Process Control / Minsk: BNTU, 2014. - Part 1. - 206 p. (In Russian).

[6] The concept of development of relay protection and automation of the electric grid complex // Appendix No. 1 to the Protocol of the Board of JSC "Rosseti" dated 06/22/2015 No. 356pr. M., 2015. – 49 p. Available in: https://mig-energo.ru/wp-content/uploads/2015/12/rza-fsk.pdf (In Russian).

[7] Distribution of cases of incorrect operation of relay protection devices by types of technical reasons and types of relay protection devices for the period from 01.01.2020 to 30.06.2020// Available in: https://www.so-ups.ru/fileadmin/files/company/rza/rza_rez_info/rza_rez_vid_teh_ 1-2k2020.xl

[8] Zakharov O. G. Reliability of digital relay protection devices. Indicators. Requirements. Estimates. - Moscow: Infra-Engineering, 2018. - 128 p. (In Russian).