UDC 004.052.32
Doi: 10.31772/2712-8970-2021-22-3-459-467

Для цитирования: Повышение надежности программного обеспечения для распределенных систем управления / О. Д. Стрелавина, С. Н. Ефимов, В. А. Терсков, М. А. Лихарев // Сибирский аэрокосмический журнал. 2021. Т. 22, № 3. С. 459–467. Doi: 10.31772/2712-8970-2021-22-3-459-467.

For citation: Strelavina O. D., Efimov S. N., Terskov V. A., Likharev M. A. Increasing software reliability of a distributed control systems. *Siberian Aerospace Journal.* 2021, Vol. 22, No. 3, P. 459–467. Doi: 10.31772/2712-8970-2021-22-3-459-467.

Повышение надежности программного обеспечения для распределенных систем управления

О. Д. Стрелавина, С. Н. Ефимов*, В. А. Терсков, М. А. Лихарев

Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева
Российская Федерация, 660037, г. Красноярск, просп. им. газ. «Красноярский рабочий», 31
*E-mail: efimov@bk.ru

Рассматривается подход для оценки и улучшения основных параметров эффективности вычислительной сети. Для распределенных систем управления надежность, при обеспечении требуемой производительности, является главным критерием. Для повышения надежности функционирования вычислительной сети вводится как аппаратная, так и программная избыточность. Для обеспечения программной избыточности разрабатываются новые версии для тех модулей программного обеспечения (ПО), в которых возможны программные сбои. Рассматривается применение методов N-версионного программирования и блока восстановления для введения программной избыточности, а также оцениваются затраты на разработку сетевого ПО с учетом мультиверсионности. Для реализации предлагаемого подхода приводится математическая модель оценки надежности ПО, которая учитывает архитектуру программного обеспечения вычислительной сети и затраты на его разработку. На основе данной модели создана программная система для проведения исследования программной надежности вычислительной сети, с помощью которой можно находить зависимость надежности сетевого программного обеспечения (СПО) от количества версий одного из выделенных программных модулей. Сравнение динамики изменения показателей надежности СПО и трудовых затрат специалистов на его разработку указывает на достаточное количество новых версий для тех модулей СПО, программную надежность которых необходимо повысить на этапе проектирования. Делается вывод о значимости как определения параметра трудозатрат на разработку СПО, так и его использования при проектировании вычислительной сети, в которой надежность повышается методом программной избыточности.

Ключевые слова: надежность вычислительной сети, надежность программного обеспечения, программная избыточность, модель надежности, трудозатраты.

Increasing software reliability of a distributed control systems

O. D. Strelavina, S. N. Efimov*, V. A. Terskov, M. A. Likharev

Reshetnev Siberian State University of Science and Technology
31, Krasnoyarskii rabochii prospekt, Krasnoyarsk, 660037, Russian Federation
*E-mail: efimov@bk.ru

The article considers a method of assessing and improving main parameters of the computer network efficiency. Reliability is the main criteria for ensuring the required performance of distributed control systems. To improve reliability of the computer network hardware and software redundancy are used. Software redundancy requires new versions to be developed for software modules in which failures are likely to occur. The article considers the N-version programming and recovery block as methods of introducing software redundancy and, taking the need to develop multiple versions of the same software module into account, estimates the costs of network software development. To implement the proposed approach article presents mathematical reliability model that takes into consideration the architecture of a computer network software and the labor costs that its development is going to require. This model becomes a basis for a software created to research computer network software reliability, which allows finding the dependence of network software reliability on the number of one of its software module versions. Comparison of the dynamics changes of reliability indicators and labor intensity of software development indicated a sufficient amount of software module versions that need to be developed. The article concludes by pointing out the importance of determining the labor intensity of network software development and of its usage in the design of a computer networks in which reliability is increased through software redundancy.

Keywords: computer network reliability, software reliability, software redundancy, reliability model, labor intensity.

Introduction

The quality of any computer network (CN) can be assessed using its main characteristics. These include the completeness of the functions performed, performance, throughput, reliability, security, transparency, scalability and versatility [1]. None of these criteria can definitely be called the most important, but among them there are several of the most significant. One of such criteria is reliability [2–5].

Reliability is the ability of the computer network (CN) to reliably perform certain functions under given conditions for a given period of time with a sufficiently high probability [6-8]. With a low level of reliability of the CN, the functionality of systems responsible for indicators of other quality criteria will also be affected. Therefore, ensuring the reliability of the CN is a priority task when building a computer network [9].

It is impossible to eliminate the possibility of failures, and therefore ensuring reliability is to reduce the number of errors that a user may encounter during the operation of the network. One of the most proven and trusted ways to increase reliability is the introduction of redundancy [10; 11].

In the hardware of the CN, redundancy serves to deal with periodically failing processors and buses. Hardware redundancy is introduced by reserving processor elements and interface buses. The required number of duplicated hardware components varies and depends on the failure rate and recovery time. Calculating the optimal number of these components is the main difficulty of introducing hardware redundancy.

Software redundancy cannot be achieved by duplication as errors that occur in software modules have an internal nature [12], which leads to the appearance of the same errors in identical copies. Therefore, instead of copies, it is necessary to create new versions that differ from each other in the programming language, the programmers who developed them, and the algorithms used. Due to internal differences between different versions, the probability of similar failures is minimized [13; 14].

Software redundancy is not applied to the entire program or software package – it is used to increase the reliability of modules that are critical for the functioning of the entire network as a whole or to which users and other modules most often turn [15]. As in the case of hardware redundancy, the number of introduced versions of software modules must be selected for each individual network, which, combined with the labor costs of developing new versions, indicates a non-trivial task of effectively using software

redundancy. To conduct research on this problem, a mathematical model to assess the reliability of the software is used.

Software reliability assessment model

The reliability of the network software is affected by its hierarchical levels – the dependence of software modules on each other can lead to a failure in one module spreading across the architecture of the entire network software [16]. The model describing the reliability of the network software should take into account the influence of the hierarchy of software modules on failures and downtime [17].

Designations used in the model:

1) M – the number of architectural levels in the architecture of the network software;

2) N_j is the number of modules at the level j, $j \in \{1, ..., M\}$;

3) D_{ij} is the set of module indexes that depend on module i at the level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$;

4) F_{ij} is a failure event that occurred in module i at level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$;

5) PU_{ij} is the probability of using module i at level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$;

6) PF_{ij} is the probability of a failure in module i at level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$;

7) PL_{nm}^{ij} - conditional probability of a failure in module m at level n when a failure occurs in module i at level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, $n \in \{1, ..., N_m\}$, $m \in \{1, ..., M\}$;

8) TA_{ij} is the relative access time to module i at level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, defined as the ratio of the average access time to the module i at level j to the number of failed modules at small levels of the architecture at the same time;

9) TC_{ij} is the relative time of the failure analysis module i to level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, which is defined as the average time of failure analysis in the module i at the level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, to the number of failed modules at all levels of architecture, analyzed at the same time;

10) TE_{ij} is the relative time for Troubleshooting in the module i to level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, defined as the ratio of the mean recovery time in the module i at the level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, to the number of failed modules at all levels of the architecture, which is a fault at the same time;

11) TU_{ij} – relative time of the module being used i at the level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, which is defined as the average usage time of module i at the level j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, to the number of modules at all levels of the architecture used at the same time;

12) Z_{ij} – the set of versions of module i, at the level j, $k = 1, ..., K$;

13) T_{ij} – the labor intensity of developing module i at level j;

14) T_{ij}^{k} – the labor intensity of developing version k of module i at level j, $k \in Z_{ij}$ in person-hours;

15) NVX_{ij} – the labor intensity of developing an acceptance test (for *RB*) or a voting algorithm (for *NVP*);

16) T_s – total network labor intensity;

17) B_{ij} is a dichotomous variable that takes the value 1 (then $NVP_{ij} = 0$, $RB_{ij} = 0$), if in the software module software redundancy is not used , otherwise it is equal to 0;

18) NVP_{ij} is a dichotomous variable that takes the value 1 (then $B_{ij} = 0$, $RB_i = 0$) if the software module uses *N*-version programming software redundancy, otherwise it is equal to 0;

19) RB_{ij} is a dichotomous variable that takes the value 1 (then $B_{ij} = 0$, $NVP_{ij} = 0$) if the software module uses software redundancy according to the recovery block method, otherwise it is equal to 0;

20) *TR* is the average downtime of the CN software, defined as the time during which the system cannot perform its functions;

21) *MTTF* is the average time of occurrence of a failure in the network software of the CN, defined as the time during which there are no failures in the system;

22) S is the readiness coefficient of the CN software;

23) T_s is the total labor intensity of the implementation of the network software.

The average failure time of the network software of the computer network is equal to:

$$MTTF = \sum_{j=1}^{M}\sum_{i=1}^{Nj}\Big\{PU_{ij}\times\left(1-PF_{ij}\right)\times\Big[TU_{ij}+$$

$$+\sum_{(m=1)\&(m\neq j)}^{M}\sum_{n=1}^{N_m}\left(\left(1-PL_{nm}^{ij}\right)\times\left(TU_{nm}+\sum_{l\in D_{nm}}\left(\left(1-PL_{lm}^{nm}\right)\times TU_{im}\right)\right)\right)+$$

$$+\sum_{k\in D_{ij}}\left(\left(1-PL_{kj}^{ij}\right)\times\left(TU_{kj}+\sum_{(m=1)\&(m\neq j)}^{M}\sum_{n=1}^{N_m}\left(\left(1-PL_{nm}^{kj}\right)\times\left(TU_{nm}+\right.\right.\right.$$

$$\left.\left.\left.+\sum_{l\in D_{nm}}\left(\left(1-PL_{lm}^{nm}\right)\times TU_{lm}\right)\right)\right)\right)\Big]\Big\}.$$

The average downtime of the network software of the computer network is equal to:

$$TR = \sum_{j=1}^{M}\sum_{i=1}^{M_j}\Big\{PU_{ij}\times PF_{ij}\times\Big[\left(TA_{ij}+TC_{ij}+TE_{ij}\right)+$$

$$+\sum_{(m=1)\&(m\neq j)}^{M}\sum_{n=1}^{Nm}\left(PL_{nm}^{ij}\times\left(\left(TA_{nm}+TC_{nm}+TE_{nm}\right)+\sum_{l\in D_{nm}}\left(PL_{lm}^{nm}\times\left(TA_{lm}+TC_{lm}+TE_{lm}\right)\right)\right)\right)\times$$

$$\times\sum_{k\in D_{ij}}\Big[PL_{kj}^{ij}\times\Big[\left(TA_{kj}+TC_{kj}+TE_{kj}\right)+$$

$$+\sum_{(m=1)\&(m\neq j)}^{M}\sum_{n=1}^{N_m}\left(PL_{nm}^{kj}\times\left(\left(TA_{nm}+TC_{nm}+TE_{nm}\right)+\sum_{l\in D_{nm}}\left(PL_{lm}^{nm}\times\left(TA_{lm}+TC_{lm}+TE_{lm}\right)\right)\right)\right)\Big]\Big]\Big]\Big]\Big\}.$$

Both of these formulas take into account the hierarchy of modules and therefore are universal for any network software with architectural levels. According to them, the readiness coefficient of the software part of the CN is also determined:

$$S = \frac{MTTF}{MTTF+TR}.$$

By themselves, these indicators consider the reliability of the network software of the computer network only in its initial state. Two main methods of introducing multiversion are described in [18]: *NVP* (*N*-version programming) and *RB* (recovery block).

NVP implies that all versions of the program are executed in parallel, and the result of their work is determined using the voting algorithm [19]. The reliability of the multiversion module i at the level j, built from K versions by the method of multiversion programming for any k, is equal to

$$R_{ij} = p_{ij}^{v}\left(1-\prod_{k=1}^{K}\left(1-p_{ij}^{k}\right)\right),$$

where p_{ij}^{v} is the probability of failure-free operation of the voting algorithm; p_{ij}^{k} - the probability of failure-free operation of version $k\in Z_{ij}$.

With the *RB* approach, multiversion is introduced through the addition of several versions of the computing module, the creation of an acceptance test that verifies the operation of the versions, and a

subprogramme that, based on the test results, either accepts the result of the module, or selects another version and restarts the calculation [19]. Reliability of such a module:

$$R_{ij} = \sum_{k \in Z_{ij}}^{K} p_{ij}^{k} p_{ij}^{AT} \prod_{l=1}^{k-1} \left(\left(1 - p_{ij}^{l}\right) p_{ij}^{AT} + p_{ij}^{l} \left(1 - p_{ij}^{AT}\right) \right),$$

where p_{ij}^{AT} is the probability of failure-free operation of the acceptance test for module i, $i = 1, ..., N$ at level j, $j = 1, ..., M$; p_{ij}^{k} is the probability of failure of version $k \in Z_{ij}$.

For any of the approaches, the probability of failure will be calculated as

$$PF_{ij} = 1 - R_{ij}.$$

When introducing software redundancy, the question always arises: exactly how many versions is needed to introduce? The possibility of using multiversion is limited by many factors. One of the most significant of them is the labor intensity, reflecting the labor costs on building a network, directly dependent on the number of versions of software modules that need to be developed. To reflect this limitation, the model has the formula [20]:

$$T_s = \sum_{j=1}^{M} \sum_{i=1}^{N_i} \left[B_{ij} T_{ij} + \left(NVP_{ij} + RB_{ij} \right) \times \left(NVX_{ij} + \sum_{k \in Z_{ij}}^{K} T_{ij}^{k} \right) \right].$$

In the study of CN, this indicator can be used to find the optimal number of versions for each software module. There are different ways to explore the network, but they all have the same points: the labor intensity of T_s should strive for a minimum, and the readiness factor S should strive for a maximum.

Investigation of the reliability of the network software

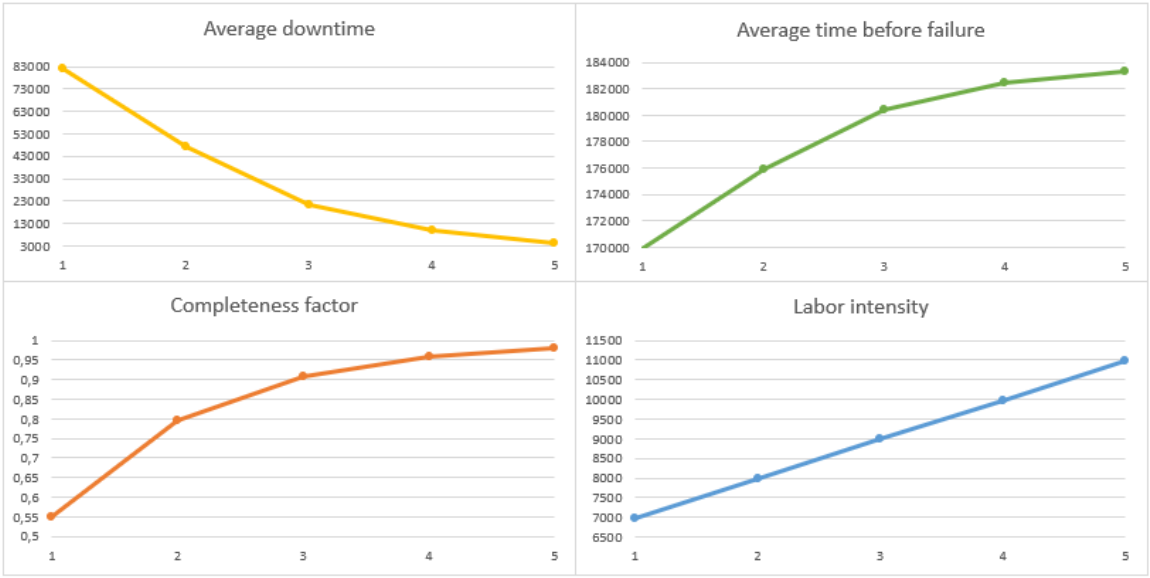
Based on the considered model, a software system has been developed that allows calculating the reliability of the network software and conducting research on the specified input data.

To conduct the study, consider the network software consisting of 10 modules. Assume that all modules, with the exception of one, work without failures (i.e. have reliability equal to 1). The overall reliability of the software will be determined by the reliability of the critical module with a value of 0.55. Software redundancy is introduced into it by the *NVP* method, the complexity of developing new versions of this module is taken for 1000.

The table shows the results of the dependence of the reliability indicators of the network software on the versions of one of its modules, and the figure clearly shows the dynamics of parameter changes.

Results of the network software study

Number of critical module versions	Average downtime	Average time before the failure	Readiness ratio	Labor intensity
1	82115	169838	0.5500	6981
2	47509	175844	0.7975	7981
3	21373	180379	0.9089	8981
4	9619	182420	0.9590	9981
5	4340	183336	0.9815	10981



Computer network software parameters' change dynamics
Динамика изменения параметров ПО ВС

Conclusion

The obtained graphs allow us to determine the number of new versions of the dedicated module, at which the increase in the cost of developing a new software element begins to exceed the increase in software reliability. At some point, the introduction of new versions ceases to be expedient, since the costs begin to exceed the returns received.

In the situation chosen for the example, the reliability of the entire network software depends on one of its modules. When designing real network software, there will be significantly more such modules, which will greatly complicate the task of research. This underlines the importance of the labor intensity parameter, which is determined by the model along with the reliability of the network software. Comparing the increase in labor costs with the increase in the software reliability of the network, you can see at what point you should stop adding new versions of software modules.

Thus, an approach was considered that allows both evaluating and improving such parameters of the network software as reliability and labor costs. To do this, on the basis of a mathematical model for assessing reliability and labor costs for development, the software was created to study the dependence of the reliability of the software and labor costs for its development on the number of versions for problematic, from the point of view of reliability, software modules. The dynamics of changes in the parameters of reliability of network software and labor costs for its development from the number of versions of one of its software modules is analyzed. From the results obtained, it can be concluded that it is important to take into account the complexity parameter when studying the reliability of a network into which software redundancy is introduced.

References

1. Kuzin A. V. *Komp'yuternye seti* [Computer networks]. Moscow, Forum: Infra-M Publ., 2011, 192 p.

2. Makaruk R. V. [Fuzzy models and a software package for analyzing the characteristics of a computer network]. *Nauchnye vedomosti belgorodskogo gosudarstvennogo universiteta. Seriya: ekonomika. informatika,*. 2013, No. 22, P. 161–166 (In Russ.).

3. Efimov S. N., Tynchenko V. V., Tynchenko V. S. [Design of computing network with efficient architecturefor complex problems distributed solving]. *Vestnik SibGAU*. 2007. No. 3, P. 15–19 (In Russ.).

4. Efimov S. N. [The industrial distributed control system reliability estimation]. *Promyshlennye ASU i kontrollery*. 2011, No. 9, P. 15–19 (In Russ.).

5. Rasulov M. M. [Software reliability assessment]. *Aktual'nye nauchnye issledovaniya v sovremennom mire*. 2020, No. 6, P. 112–116 (In Russ.).

6. Lozhkov A. V. [Methodology for assessing the reliability of a computer network]. *Nauchnye zapiski molodykh issledovateley*. 2014, No. 4, P. 28–31 (In Russ.).

7. Gurov S. V., Polovko A. M. *Osnovy teorii nadezhnosti* [Fundamentals of the theory of reliability]. St.Petersburg, BKhV-Peterburg Publ., 2006, 704 p.

8. Brzhozovskiy B. M., Martynov V. V., Skhirtladze A. G. *Diagnostika i nadezhnost' avtomatizirovannykh sistem* [Diagnostics and reliability of automated systems]. Moscow, TNT Publ., 2013, 352 p.

9. Vorotnikova T. Y. [Research of the development of increasing the software reliability issue]. *Globus*. 2019, No. 11, P. 42–45 (In Russ.).

10. Shubinskiy I. B. *Nadezhnye otkazoustoychivye informatsionnye sistemy. Metody sinteza* [Reliable fault-tolerant information systems. Synthesis methods]. Ulyanovsk, Pechatnyy dvor Publ., 2016, 547 p.

11. Gruzenkin D. V., Kamysov S. S. [Application of software redundancy to increase software reliability]. *Novaya nauka: Ot idei k rezul'tatu*. 2016, No. 9, P. 9–11 (In Russ.).

12. Naumov A. A., Aydinyan A. R. [Software reliability and methods to improve it: Don's Engineering Bulletin]. *Nadezhnost' programmnogo obespecheniya i metody ee povysheniya. Inzhenernyy vestnik Dona*. 2018, No. 2. (In Russ.). Available at: http://ivdon.ru/ru/magazine/archive/ n2y2018/4946 (accessed 10.05.2021).

13. Kovalev P. V. [The reliability research of n-version software using methods of network analysis]. *Vestnik SibGAU*. 2009, Vol. 22, № 1-2, P. 56–59 (In Russ.).

14. Pozdnyakov D. A. *Komponentnaya programmnaya arkhitektura mul'tiversionnykh sistem obrabotki informatsii i upravleniya. Dis. kand. tekhn. nauk.* [Component software architecture of multiversion information processing and control systems. Cand. techn. sci. diss.]. Krasnoyarsk, 2006, 126 p.

15. Tynchenko V. V., Tsarev R. Yu. [Toward the problem of evaluation of the reliability of software with multiple level architecture]. *K voprosu otsenki nadezhnosti programmnogo obespecheniya s mnogourovnevoy arkhitekturoy. Sovremennye problemy nauki i obrazovaniya.* 2015, No. 2-1 (In Russ.). Available at: http://science-education.ru/ru/article/view?id=20878 (accessed: 18.04.2021)

16. Karavanov A. V., Ivanov N. D [Software architecture for highly reliable systems]. *Kosmicheskie apparaty i tekhnologii*. 2018, No. 2, P. 100–104 (In Russ.).

17. Rusakov M. A. *Mnogoetapnyy analiz arkhitekturnoy nadezhnosti v slozhnykh informatsionno-upravlyayushchikh sistemakh. Dis. kand. tekhn. nauk* [Multi-stage analysis of architectural reliability in complex information management systems. Cand. techn. sci. diss.]. Krasnoyarsk, 2005, 168 p.

18. Novoy A. V. *Sistema analiza arkhitekturnoy nadezhnosti programmnogo obespecheniya. Dis. kand. tekhn. nauk.* [Software architectural reliability analysis system. Cand. techn. sci. diss.]. Krasnoyarsk, 2011, 131 p.

19. Kovalev I. V., Novoy A. V. [Software architecture for highly reliable systems]. *Vestnik SibGAU*. 2007, No. 4, P. 14–17 (In Russ.).

20. Sheenok D. A. *Mnogokriterial'naya optimizatsiya otkazoustoychivoy programmnoy arkhitektury spetsializirovannymi evolyutsionnymi algoritmami. Dis. kand. tekhn. nauk.* [Multi-criteria optimization of fail-safe software architecture by specialized evolutionary algorithms. Cand. techn. sci. diss.]. Krasnoyarsk, 2013, 84 p.

Библиографические ссылки

1. Кузин А. В. Компьютерные сети. М. : Форум: Инфра-М, 2011. 192 с.

2. Макарук Р. В., Гиляров В. Н. Нечёткие модели и программный комплекс для анализа характеристик вычислительной сети // Научные ведомости белгородского государственного университета. Серия: экономика. Информатика. 2013. № 22. С. 161–166.

3. Ефимов С. Н., Тынченко В. В., Тынченко В. С. Проектирование вычислительной сети эффективной архитектуры для распределенного решения сложных задач // Вестник СибГАУ. 2007. № 3 (16). С. 15–19.

4. Ефимов С. Н. Оценка надежности распределенных автоматизированных систем управления технологическим процессом // Промышленные АСУ и контроллеры. 2011. № 9. С. 9–13.

5. Расулов М. М. Оценка надежности программного обеспечения // Актуальные научные исследования в современном мире. 2020. № 6 (62). С. 112–116.

6. Ложков А. В. Методика оценки надежности вычислительной сети // Научные записки молодых исследователей. 2014. № 4. С. 28–31.

7. Гуров С. В., Половко А. М. Основы теории надежности. СПб.: БХВ-Петербург, 2006. 704 с.

8. Бржозовский Б. М., Мартынов В. В., Схиртладзе А. Г. Диагностика и надежность автоматизированных систем. М.: ТНТ, 2013. 352 с.

9. Воротникова Т. Ю. Исследование развития вопроса повышения надежности программного обеспечения // Globus. 2019. № 11 (44). С. 42–45.

10. Шубинский И. Б. Надежные отказоустойчивые информационные системы. Методы синтеза. Ульяновск: Печатный двор, 2016. 547 с.

11. Грузенкин Д. В., Камысов С. С. Применение программной избыточности для повышения надежности программного обеспечения // Новая наука: от идеи к результату. 2016. № 9. С. 9–11.

12. Наумов А. А., Айдинян А. Р. Надежность программного обеспечения и методы ее повышения // Инженерный вестник Дона. 2018. № 2 [Электронный ресурс]. URL: http://ivdon.ru/ru/magazine/archive/n2y2018/4946 (дата обращения: 10.05.2021).

13. Ковалев П. В. Определение надежности мультиверсионного программного обеспечения с использованием методов анализа сетей // Вестник СибГАУ. 2009. № 1-2. С. 56–59.

14. Поздняков Д. А. Компонентная программная архитектура мультиверсионных систем обработки информации и управления: дис. канд. техн. наук. Красноярск, 2006. 126 с.

15. Тынченко В. В., Царев Р. Ю. К вопросу оценки надежности программного обеспечения с многоуровневой архитектурой [Электронный ресурс] // Современные проблемы науки и образования. 2015. № 2-1. URL: http://science-education.ru/ru/article/view?id=20878 (дата обращения: 18.04.2021).

16. Караванов А. В., Иванов Н. Д. Архитектура программного обеспечения для высоконадежных систем // Космические аппараты и технологии. 2018. № 2. С. 100–104.

17. Русаков М. А. Многоэтапный анализ архитектурной надежности в сложных информационно-управляющих системах: дис. канд. техн. наук. Красноярск, 2005. 168 с.

18. Новой А. В. Система анализа архитектурной надежности программного обеспечения: дис. канд. техн. наук. Красноярск, 2011. 131 с.

19. Ковалев И. В., Новой А. В. Расчет надежности отказоустойчивых архитектур программного обеспечения // Вестник СибГАУ. 2007. № 4. С. 14–17.

20. Шеенок Д. А. Многокритериальная оптимизация отказоустойчивой программной архитектуры специализированными эволюционными алгоритмами: дис. канд. техн. наук. Красноярск, 2013. 84 с.

Стрелавина Олеся Денисовна – магистрант информатики и вычислительной техники; Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева. E-mail: strelavi@mail.ru.

Ефимов Сергей Николаевич – кандидат технических наук, доцент, доцент кафедры информационно-управляющих систем; Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева. E-mail: efimov@bk.ru.

Терсков Виталий Анатольевич – доктор технических наук, профессор, профессор кафедры информационно-управляющих систем; Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева. E-mail: terskovva@mail.ru.

Лихарев Михаил Андреевич – магистрант информатики и вычислительной техники; Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева. E-mail: misha.likharev@inbox.ru.

Strelavina Olesya Denisovna – magistrand of computer science; Reshetnev Siberian State University of Science and Technology. E-mail: strelavi@mail.ru.

Efimov Sergei Nikolaevich – Cand. Sc., associate professor of the department of information management systems; Reshetnev Siberian State University of Science and Technology. E-mail: efimov@bk.ru.

Terskov Vitaliy Anatol'evich – Dr. Sc., professor, professor of the department of information management systems; Reshetnev Siberian State University of Science and Technology. E-mail: terskovva@mail.ru.

Likharev Mikhail Andreevich – magistrand of computer science; Reshetnev Siberian State University of Science and Technology. E-mail: misha.likharev@inbox.ru.