Criteria and Rules for Classification of Software Failures and Vulnerabilities

Tetiana Hovorushchenko[a]

[a] *Khmelnytskyi National University, Institutska str., 11, Khmelnytskyi, 29016, Ukraine*

Abstract

The paper proves that despite numerous studies in the field of improving the reliability and security of system and application software, conducted over the years, despite building a variety of approaches by many leading scientists in the field, vulnerabilities and software failures still exist and manifest. An actual task now is to predict software failures and vulnerabilities, for which the problem of classification of software failures and vulnerabilities should be solved first. The purpose of this study is to: build mathematical models of software failures and vulnerabilities; construct criteria and production rules for the classification of software failures and vulnerabilities. The classification of software failures and vulnerabilities depending on their manifestation, mathematical models of software failure and vulnerabilities, as well as criteria for the classification of software failures and vulnerabilities, which allow formulating rules for classification of software failures and vulnerabilities, were developed. The proposed rules for the classification of software failures and vulnerabilities provide an opportunity to facilitate the process of identifying errors in the software. A promising area of research is to build a method and algorithm for predicting software failures and vulnerabilities based on developed mathematical models of failure and vulnerabilities, classification criteria, and rules for software failures and vulnerabilities.

Keywords

Software failure, insignificant failure, significant failure, critical failure, software vulnerability, correct work vulnerability, information integrity vulnerability, information confidentiality vulnerability, information availability vulnerability.

1. Introduction & Related Works

In terms of the reliability and security of the computer as a whole, system software errors are the biggest threat. For example, a buffer overflow error can cause the system to fail completely (reliability issue) or allow a professionally written virus to intercept control of the computer (security issue). Application software also contains many defects, but these errors can cause only limited damage if the system software does not contain errors.

The modern system software has features that make it unreliable and dangerous – it is huge and has a very weak localization of errors [1]. For example, the Linux core has more than 2.5 million lines of code, and the Windows core has twice as many. The operation system (OS) contains hundreds of thousands of procedures, which are assembled together as a single binary program operating in core mode.

The study conducted in [2] shows that as the size of the software increases, the error rate in the software increases, and for software larger than 512K is 4-100 errors per 1000 lines of code. Another study [3] provides a density of 2 to 75 errors per 1000 lines of executable code, depending on the size of the module. Using a moderate estimate of 4 errors per 1000 lines of code, we have 10,000 possible

ITTAP'2021: The 1st International Workshop on Information Technologies: Theoretical and Applied Problems, November 16–18, 2021, Ternopil, Ukraine
EMAIL: tat_yana@ukr.net (T. Hovorushchenko)
ORCID: 0000-0002-7942-1857 (T. Hovorushchenko)

errors in the Linux core and 20,000 possible errors in the Windows core. An even bigger problem is that usually, about 70% of the operating system consists of drivers that have 3-7 times more errors than simple code [4], so the calculated number of errors is probably much underestimated. It is clear that finding and correcting all these errors is not feasible. Analysis of the Android mobile OS showed 359 software bugs, including 88 high-risk bugs and 271 medium-risk bugs, which is a serious threat to OS security [5].

As for the weakness of error localization in the OS, each of the millions of lines of core code can overwrite the basic data structures used by its independent components, disabling the system in ways that are difficult to detect. In addition, if a virus infects one core procedure, there is no way to keep it from spreading quickly to other procedures and prevent it from invading the computer as a whole.

Currently, there are various approaches used to increase the reliability and security of the OS: 1) the Nooks approach [6], which protects the core from device drivers by placing each driver in a shell of protective software; 2) paravirtualization approach [7], which allows the simultaneous launch of several operating systems on one computer using the mechanism of virtual machines; 3) the approach of creating multi-server operating systems [8], in which only the micro core is run in core mode, and the rest of the OS works in user mode as a set of completely isolated server processes and drivers; 4) Microsoft Research approach [1], which rejects the concept of OS as a single program running in core mode, with some set of user processes running in user mode, and replaces it with a system written in new type languages that provide the security of types. All of these approaches are based on preventing a complete system failure caused by multiple errors in device drivers, but are not aimed at predicting and resolving system software failures and vulnerabilities.

Despite numerous studies in the field of improving the reliability and security of software over the years, despite building a variety of approaches by many leading scientists in the field, vulnerabilities and software failures still exist and manifest themselves in the form of OS crashes, inability to load, "loss" of certain peripherals, information leaks, financial losses, environmental and man-made disasters [9-14]. The consequences of insufficient software security are presented in Table1.

Table 1
Consequences of insufficient software security

Description	Consequences and losses	Source
Attack the system process svchost.exe by error antivirus (2010)	Continuous overload of the computer on which such antivirus was installed	[15]
IOS 5 operating system error (2012)	Fast iPhone battery loss	[16]
Detection of 0.6% of critical vulnerabilities and 19.5% of high-risk vulnerabilities in system software	Problems with the functioning of the system software	[17]
As a result of the software vulnerability, 500 million records of data from users of Yahoo services were opened	Potential leakage of 500 million records of users of Yahoo services	[18]
Equifax has lost personal and financial information about 140 million people due to software vulnerabilities. The company was unable to correct the critical vulnerability and did not inform the public about the situation that arose within a few weeks after the incident was discovered	Equifax's total loss was $ 575 million dollars	[19]

The vulnerability in the software allowed attackers to gain access to 50 million profiles of users of the social network Facebook	Potential data leakage of 50 million users of the social network Facebook	[20]
The Uber taxi app was hacked and information about 600,000 drivers and 57 million user accounts was stolen. Instead of reporting the incident, Uber paid the attacker $ 100,000 for his silence	Uber was fined $ 148 million dollars. The total loss amounted to 148.1 million dollars	[21]

Therefore, software behavior is indeterminate due to errors, the success of software security attempts is possible only by improving the quality of software and reducing the number of errors, for this process of identifying errors should be facilitated, so *the actual task* now is to predict software failures and vulnerabilities, for which we must first decide the task of classifying the software failures and vulnerabilities.

In this case, *the purpose of this study* is to:

1. Develop mathematical models of software failures and vulnerabilities
2. Construct criteria and production rules for the classification of software failures and vulnerabilities

2. Mathematical models of software failure and vulnerabilities

During developing the mathematical models, we will use the following symbols:

∃ - quantifier of existence ("exists for some");
∈ - membership in the set;
∩ - simultaneous fulfillment of conditions (logical operation "AND");
∀ - quantifier of generality ("for all");
→ - implication.

From the point of view of the theory of reliability, a *failure* is an event that is a violation of the operational state of the object, as a result of which the system ceases to perform all or part of its functions, i.e. an event that involves the transition of the object from one level of operation to another, lower, or to the completely inoperable state [22-24].

Insignificant failure Isf from the point of view of the software will be called the termination of operation of the program for the time exceeding the set threshold, without data loss and requirement of the restart of the computer.

Significant failure Sf from the point of view of software will be called the termination of the program for a time exceeding a specified threshold, with the loss of all or part of the data, but without the need to restart the computer.

Critical failure Cf from the point of view of the software will be called the termination of the program, which requires a restart of the computer on which the software operates.

Then *the mathematical model of failure* has the following form:

$$Fl = \begin{cases} Isf \ \mid \exists(st_i \in ST) \cap (Df_i = Df_{i-1}) \\ Sf \ \mid \exists(st_i \in ST) \cap (Df_i < Df_{i-1}) \\ Cf \ \mid \exists(st_i \in ST) \end{cases}$$

where i – time after software failure; $(i\text{-}1)$ – time before a software failure; st_i – software state after software failure; ST = $\{st_1,..,st_n\}$ – the set of working states of the software (n – the total number of working states of the software); Df_i – set of data after software failure; Df_{i-1} – set of data before the failure of the software.

In the context of functional security, the focus is on the functional features of the software, the use of which may disrupt its proper operation, as well as the integrity, availability, or confidentiality of information.

In computer security, the term "*vulnerability*" is used to refer to a flaw in a system that can be used to intentionally compromise its integrity and cause it to malfunction. Vulnerabilities can be the result of programming errors, system design flaws, unreliable passwords, viruses, and other malicious programs [25].

In addition, software vulnerabilities are defined as undeclared capabilities – software functionality that is not described or does not correspond to those described in the documentation, the use of which may violate the confidentiality, availability, or integrity of processed information.

The correct work vulnerability Cwv will be called a functional feature that leads to the termination of the program functioning for a time that exceeds a specified threshold, i.e. to software failure.

The information integrity vulnerability Iiv is called the functional feature of the software, which leads to loss of data completeness, to unauthorized (malicious or accidental) change of data when performing a certain operation associated with this functional feature.

The information confidentiality vulnerability Icv is the functional feature of the software, which leads to leakage, unauthorized disclosure, illegal access, or use of any information.

The information availability vulnerability Iav will be called the functional feature of the software, which leads to a state in which entities that have access rights to information cannot implement them without hindrance.

Obviously, vulnerabilities can be complex - for example, vulnerabilities that simultaneously threaten the integrity and confidentiality of information, or vulnerabilities that simultaneously threaten the correct operation of software, integrity, confidentiality, and availability of information, etc.

Given the above, let's develop a *mathematical model of software vulnerabilities* in terms of manifestations of vulnerabilities:

$$V = \begin{cases} Cwv\ \forall(Cwv \in FT) \cap \exists(Cwv\ \to F) \\ Iiv\ \forall(Iiv \in FT) \cap \exists(Iiv\ \to (D_j\ \neq\ D_{j-1})) \\ Icv\ \forall(Icv \in FT) \cap \exists\left(Icv\ \to \left(I_j\ \neq\ I_{j-1}\right) \cap \left(I_j >\ I_{j-1}\right)\right) \\ Iav\ \forall(Iav \in FT) \cap \exists\left(Iav\ \to \left(A_j\ \neq\ A_{j-1}\right) \cap \left(A_j <\ A_{j-1}\right)\right) \\ Cwv\ \cap Iiv \\ Cwv\ \cap Icv \\ Cwv\ \cap Iav \\ Iiv\ \cap Icv \\ Iiv\ \cap Iav \\ Icv\ \cap Iav \\ Cwv\ \cap Iiv\ \cap Icv \\ Cwv\ \cap Icv\ \cap Iav \\ Cwv\ \cap Iiv\ \cap Iav \\ Iiv\ \cap Icv\ \cap Iav \\ Cwv\ \cap Iiv\ \cap Icv\ \cap Iav \end{cases}$$

where: $FT = \{ft_1,...,ft_m\}$ – set of all software functional features; *m* is the total number of all software functional features; j – the time after performing the functional feature with the vulnerability; *(j-1)* – the time before performing the functional feature with the vulnerability; D_j – set of data after the execution of the functional feature with the vulnerability; D_{j-1} – set of data before performing the functional feature with a vulnerability; I_j is the set of data that can be used after performing a functional feature with a vulnerability; I_{j-1} – set of data that can be used before performing the functional feature with a vulnerability; A_j – set of data that can be freely used by an entity that has access to them, after performing a functional feature with a vulnerability; A_{j-1} is a set of data that can be freely used by an entity that has access to them before performing the functional feature with a vulnerability.

3. Criteria and rules for classifying the software failures and vulnerabilities

As proved in Section 1, the current challenge is to predict software failures, for which the problem of classification of software failures should be solved first. To solve the problem of software failure classification, it is necessary to formulate failure classification criteria. Given the proposed definitions of failure types, let’s propose the following *failure classification criteria*:

1. Loss of operability (ability to function) of the software
2. Loss of data (all or part) of the software
3. The need to restart the computer

Given the proposed definitions of types of vulnerabilities, let's propose the following *criteria for classifying the vulnerabilities*:

1. Occurrence of software failure
2. Loss of completeness of data
3. Leakage of unauthorized information
4. Impossibility to obtain permitted information

Using the developed mathematical model of failure, and also the proposed criteria of classification of failures of the software, let's construct *rules of classification of failures*:

1. If the state s_i of the software after cessation of operation is operational ($s_i \epsilon S$) and during the cessation of operation there was no data loss, i.e. $D_i = D_{i-1}$, the failure is insignificant
2. If the state s_i of the software after cessation of operation is operational ($s_i \epsilon S$), but during the cessation of operation there was a loss of data, i.e. $D_i < D_{i-1}$, the failure is significant
3. If the state s_i of the software after cessation of operation is not operational (s_i is not an element of the set S), the failure is critical

Using the developed mathematical model of vulnerability and the proposed criteria of classification of vulnerabilities, let's construct *rules of classification of vulnerabilities*:

1. If during the execution of the *h*-th functionality of the software it stopped functioning for a time t_h, which exceeds the specified for the software of this type threshold time t_{lim} ($t_h > t_{lim}$), then the *h*-th functionality of the software is correct work vulnerability
2. If after the execution of the *h*-th functionality of the software there was a loss of completeness of data, i.e. $D_{j\ (h)} \neq D_{j-1\ (h)}$, then *h*-th functionality of the software is information integrity vulnerability
3. If after the execution of the *h*-th functionality of the software there was a data leak, i.e. $I_{j\ (h)} \neq I_{j-1\ (h)}$, and $I_{j\ (h)} > I_{j-1\ (h)}$, then *h*-th functionality of the software is information confidentiality vulnerability
4. If after the execution of the *h*-th functionality of the software there is an impossibility to obtain the information allowed by the user, i.e. $A_{j\ (h)} \neq A_{j-1\ (h)}$, and $A_{j\ (h)} < A_{-1\ (h)}$, then *h*-th functionality of the software is information availability vulnerability
5. If during the execution of the *h*-th functionality of the software it stopped functioning for a time t_h, which exceeds the specified for the software of this type threshold time t_{lim} ($t_h > t_{lim}$), there was a loss of completeness of data, i.e. $D_{j\ (h)} \neq D_{j-1\ (h)}$, then the *h*-th functionality of the software is correct work and information integrity vulnerability
6. If during the execution of the *h*-th functionality of the software it stopped functioning for a time t_h, which exceeds the specified for the software of this type threshold time t_{lim} ($t_h > t_{lim}$), and there was a data leak, i. e. $I_{j\ (h)} \neq I_{j-1\ (h)}$, and $I_{j\ (h)} > I_{j-1\ (h)}$, then the *h*-th functionality of the software is correct work and information confidentiality vulnerability
7. If during the execution of the *h*-th functionality of the software it stopped functioning for a time t_h, which exceeds the specified for the software of this type threshold time t_{lim} ($t_h > t_{lim}$), and there is an impossibility to obtain the information allowed by the user, i.e. $A_{j\ (h)} \neq A_{j-1\ (h)}$, and $A_{j\ (h)} < A_{-1\ (h)}$, then the *h*-th functionality of the software is correct work and information availability vulnerability
8. If after the execution of the *h*-th functionality of the software there was a loss of completeness of data, i.e. $D_{j\ (h)} \neq D_{j-1\ (h)}$, and there was a data leak, i. e. $I_{j\ (h)} \neq I_{j-1\ (h)}$, and $I_{j\ (h)} > I_{j-1\ (h)}$, then *h*-th functionality of the software is information integrity and information confidentiality vulnerability
9. If after the execution of the *h*-th functionality of the software there was a loss of completeness of data, i.e. $D_{j\ (h)} \neq D_{j-1\ (h)}$, and there is an impossibility to obtain the information allowed by the user, i.e. $A_{j\ (h)} \neq A_{j-1\ (h)}$, and $A_{j\ (h)} < A_{-1\ (h)}$, then *h*-th functionality of the software is information integrity and information availability vulnerability
10. If after the execution of the *h*-th functionality of the software there was a data leak, i. e. $I_{j\ (h)} \neq I_{j-1\ (h)}$, and $I_{j\ (h)} > I_{j-1\ (h)}$, and there is an impossibility to obtain the information allowed by the user,

i.e. $A_{j\,(h)} \neq A_{j-1\,(h)}$, and $A_{j\,(h)} < A_{-1\,(h)}$, then *h*-th functionality of the software is information confidentiality and information availability vulnerability

11. If during the execution of the *h*-th functionality of the software it stopped functioning for a time t_h, which exceeds the specified for the software of this type threshold time t_{lim} ($t_h > t_{lim}$), there was a loss of completeness of data, i.e. $D_{j\,(h)} \neq D_{j-1\,(h)}$, and there was a data leak, i. e. $I_{j\,(h)} \neq I_{j-1\,(h)}$, and $I_{j\,(h)} > I_{j-1\,(h)}$, then the *h*-th functionality of the software is correct work, information integrity and information confidentiality vulnerability
12. If during the execution of the *h*-th functionality of the software it stopped functioning for a time t_h, which exceeds the specified for the software of this type threshold time t_{lim} ($t_h > t_{lim}$), there was a data leak, i. e. $I_{j\,(h)} \neq I_{j-1\,(h)}$, and $I_{j\,(h)} > I_{j-1\,(h)}$, and there is an impossibility to obtain the information allowed by the user, i.e. $A_{j\,(h)} \neq A_{j-1\,(h)}$, and $A_{j\,(h)} < A_{-1\,(h)}$, then the *h*-th functionality of the software is correct work, information confidentiality and information availability vulnerability
13. If during the execution of the *h*-th functionality of the software it stopped functioning for a time t_h, which exceeds the specified for the software of this type threshold time t_{lim} ($t_h > t_{lim}$), there was a loss of completeness of data, i.e. $D_{j\,(h)} \neq D_{j-1\,(h)}$, and there is an impossibility to obtain the information allowed by the user, i.e. $A_{j\,(h)} \neq A_{j-1\,(h)}$, and $A_{j\,(h)} < A_{-1\,(h)}$, then the *h*-th functionality of the software is correct work, information integrity and information availability vulnerability
14. If after the execution of the *h*-th functionality of the software there was a loss of completeness of data, i.e. $D_{j\,(h)} \neq D_{j-1\,(h)}$, there was a data leak, i. e. $I_{j\,(h)} \neq I_{j-1\,(h)}$, and $I_{j\,(h)} > I_{j-1\,(h)}$, and there is an impossibility to obtain the information allowed by the user, i.e. $A_{j\,(h)} \neq A_{j-1\,(h)}$, and $A_{j\,(h)} < A_{-1\,(h)}$, then *h*-th functionality of the software is information integrity, information confidentiality and information availability vulnerability
15. If during the execution of the *h*-th functionality of the software it stopped functioning for a time t_h, which exceeds the specified for the software of this type threshold time t_{lim} ($t_h > t_{lim}$), there was a loss of completeness of data, i.e. $D_{j\,(h)} \neq D_{j-1\,(h)}$, there was a data leak, i. e. $I_{j\,(h)} \neq I_{j-1\,(h)}$, and $I_{j\,(h)} > I_{j-1\,(h)}$, and there is an impossibility to obtain the information allowed by the user, i.e. $A_{j\,(h)} \neq A_{j-1\,(h)}$, and $A_{j\,(h)} < A_{-1\,(h)}$, then the *h*-th functionality of the software is correct work, information integrity, information confidentiality and information availability vulnerability

4. Experiments & Results

For example, let's consider the software for traffic light control at the intersection of streets, developed by one of the IT companies in Khmelnytskyi (Ukraine).

Two failures occurred during the test operation of this software. After the first failure f_1, the software remained operational, but data was lost during the failure. After the second failure f_2, the software stopped working at all.

Using the developed rules of classification of failures, it was established that:

1. Failure f_1 is significant, i.e. $f_1 \in Sf$
2. Failure f_2 is critical, i.e. $f_2 \in Cf$

A detailed study of software for traffic light control at intersections showed that the failures were due to a number of vulnerabilities in the analyzed software. Thus, one of the available functional features v1 led to the termination of the software, another functional feature v2 led to the loss of data completeness, the third functional feature v3 led to the impossibility of obtaining the information allowed by the user, the fourth functional feature v4 led to the loss of data completeness and the inability to obtain the authorized user information at the same time, and the fifth functional feature v5 led to the termination of the software, loss of data completeness and the inability to obtain the information allowed by the user at the same time.

Guided by the developed rules of classification of vulnerabilities, it is established that:

1. Functionality v_1 is correct work vulnerability, i.e. $v_1 \in Cwv$ (rule 1)
2. Functionality v_2 is information integrity vulnerability, i.e. $v_2 \in Iiv$ (rule 2)
3. Functionality v_3 is information availability vulnerability, i.e. $v_3 \in Iav$ (rule 4)
4. Functionality v_4 is information integrity and information availability vulnerability, i.e. $v_4 \in Iiv \cap Iav$ (rule 9)

5. Functionality v_5 is correct work, information integrity and information availability vulnerability, i.e. $v_5 \in Cwv \cap Iiv \cap Iav$ (rule 13)

Then the distribution of the 5 vulnerabilities found in the software by their types is as follows – Figure 1:

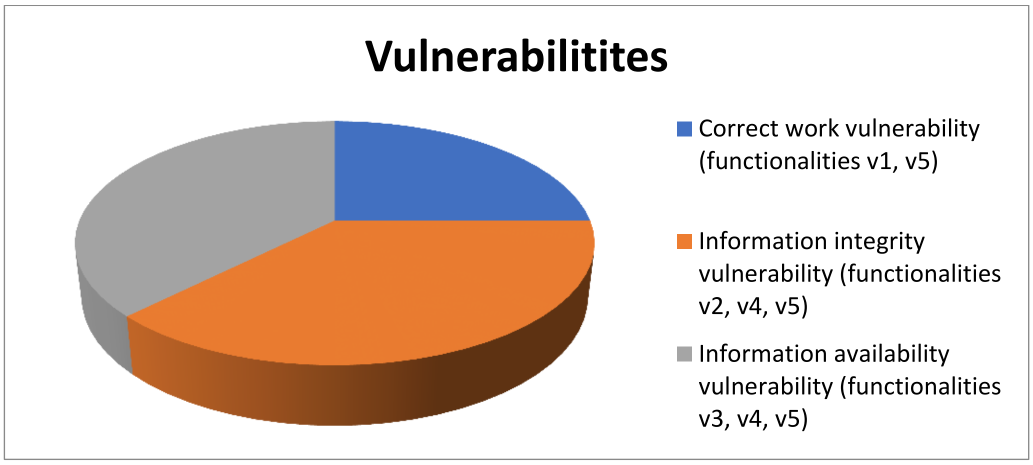

Figure 1: Distribution of the vulnerabilities in software for traffic light control at street intersections

Therefore, the proposed rules for classifying the software failures and vulnerabilities provide an opportunity to facilitate the process of identifying errors and incorrect functionality in the software.

5. Conclusions

The paper analyzes the current state of the field of software reliability and security, as a result of which the features of the software that make it unreliable and dangerous are identified. An analysis of the approaches currently used to improve software reliability and security has shown that the approaches are based on preventing complete system failure, but are not aimed at predicting and resolving software failures and vulnerabilities. The paper proves that despite numerous studies in the field of improving the reliability and security of system and application software, conducted over the years, despite building a variety of approaches by many leading scientists in the field, vulnerabilities and software failures still exist and manifest.

The classification of software failures and vulnerabilities depending on their manifestation, mathematical models of software failure and vulnerabilities, as well as criteria for the classification of software failures and vulnerabilities, which allow formulating rules for classification of software failures and vulnerabilities, were developed. The proposed rules for the classification of software failures and vulnerabilities provide an opportunity to facilitate the process of identifying errors in the software.

A promising area of research is to develop a method and algorithm for predicting software failures and vulnerabilities based on developed mathematical models of failure and vulnerabilities, classification criteria, and rules for the classification of software failures and vulnerabilities.

Practical areas of application of the proposed approach are predicting the failures and vulnerabilities of both application and system software, as well as the classification of failures and vulnerabilities of both application and system software.

6. References

[1] A. Mitrokotsa, M. Rieback, A. Tanenbaum, Classifying RFID attacks and defenses. Information Systems Frontiers 12 5 (2010) 491-505. doi: 10.1007/s10796-009-9210-z.
[2] S. McConnell, Code complete, Microsoft Press, Redmond, 2013.
[3] T. Ostrand, E. Weyuker, Predicting bugs in large industrial software systems. Lecture Notes in Computer Science 7171 (2013) 71-93. doi: 10.1007/978-3-642-36054-1_3.

[4] D. Cotroneo, D.Di Leo, R. Natella, R. Pietrantuono, A case study on state-based robustness testing of an operating system for the avionic domain. Lecture Notes in Computer Science 6894 (2011) 213-227. doi: 10.1007/978-3-642-24270-0_16.

[5] Study: 359 Android code flaws pose security risks, 2015. URL: http://news.cnet.com/8301-30685_3-20021437-264.html?part=rss&subj=news&tag=2547-1_3-0-20.

[6] N. Shalev, Improving system security and reliability with OS help, Research Thesis, 2018.

[7] J. Deepbawa, Boosting the cloud meta-operating system with heterogeneous kernels. a novel approach based on containers and micro services. International Journal of Advances in Electronics and Computer Science 6 9 (2019) 40-44.

[8] Even More Alternative Operating Systems: MINIX STILL ALIVE, 2017. URL: https://dwaves.de/2017/03/25/even-more-alternative-operating-systems-minix-still-alive/.

[9] T. Hovorushchenko, Methodology of evaluating the sufficiency of information for software quality assessment according to ISO 25010. Journal of Information and Organizational Sciences 42 1 (2018) 63-85. doi: 10.31341/jios.42.1.4.

[10] T. Hovorushchenko, Information technology for assurance of veracity of quality information in the software requirements specification. Advances in Intelligent Systems and Computing 689 (2018) 166–185. doi: 10.1007/978-3-319-70581-1_12.

[11] O. Pomorova, T. Hovorushchenko, Research of artificial neural network's component of software quality evaluation and prediction method, in: Proceedings of the 2011 IEEE 6-th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS’2011, Prague, 2011, vol.2, pp. 959-962. doi: 10.1109/IDAACS.2011.6072916.

[12] T. Hovorushchenko, O. Pavlova, D. Medzatyi, Ontology-based intelligent agent for determination of sufficiency of metric information in the software requirements. Advances in Intelligent Systems and Computing 1020 (2020) 447-460. doi: 10.1007/978-3-030-26474-1_32.

[13] A. Drozd, M. Drozd, V. Antonyuk, Features of hidden fault detection in pipeline components of safety-related system. CEUR-WS 1356 (2015) 476–485.

[14] O. Drozd, K. Zashcholkin, O. Martynyuk, O. Ivanova, J. Drozd, Development of checkability in fpga components of safety-related systems. CEUR-WS 2762 (2020) 30-42.

[15] McAfee DAT 5958 Update Issues, 2010. URL: http://isc.sans.edu/diary/McAfee+DAT+5958+Update+Issues/8656.

[16] How to fix battery life issues with iOS 6 or iPhone 5, 2012. URL: http://www.imore.com/how-fix-battery-life-problems-ios-6-or-iphone-5.

[17] Vulnerabilities, 2021. URL: http://www.securitylab.ru/vulnerability/.

[18] Yahoo says 500 million accounts stolen, 2016. URL: https://money.cnn.com/2016/09/22/technology/yahoo-data-breach.

[19] Equifax Made Major Errors That Led to Hack, Ex-CEO Concedes, 2017. URL: https://www.bloomberg.com/news/articles/2017-10-02/ex-equifax-ceo-says-human-tech-failures-allowed-breach-to-occur.

[20] Facebook Says Breach Affected About 50 Million Accounts, 2018. URL: https://www.bloomberg.com/news/articles/2018-09-28/facebook-says-security-breach-affected-about-50-million-accounts.

[21] A.G. Underwood Announces Record $148 Million Settlement With Uber Over 2016 Data Breach, 2018. URL: https://ag.ny.gov/press-release/2018/ag-underwood-announces-record-148-million-settlement-uber-over-2016-data-breach.

[22] What is Software Failure, 2021. URL: https://www.igi-global.com/dictionary/investigation-of-software-reliability-prediction-using-statistical-and-machine-learning-methods/59093.

[23] M. Drozd, A. Drozd, Safety-related instrumentation and control systems and a problem of the hidden faults, in: Proceedings of the 10th International Conference on Digital Technologies, Zhilina, 2014, pp. 137–140. doi: 10.1109/DT.2014.6868692.

[24] O. Pomorova, T. Hovorushchenko, Artificial neural network for software quality evaluation based on the metric analysis, in: Proceedings of IEEE East-West Design & Test Symposium, EWDTS’2013, Kharkiv, 2013, pp.200-203. doi: 10.1109/EWDTS.2013.6673193

[25] M. Howard, D. LeBlanc, J. Viega, 24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them, McGraw-Hill Education, Redmond, 2010.