Method and Technology for Ensuring the Software Security by Identifying and Classifying the Failures and Vulnerabilities

Tetiana Hovorushchenko[a], Peter Popov[b], Dmytro Medzatyi[a] and Yurii Voichur[a]

[a] *Khmelnytskyi National University, Institutska str., 11, Khmelnytskyi, 29016, Ukraine*
[b] *City University of London, Northampton Square, London, EC1V 0HB, United Kingdom*

Abstract

The conducted literature review on known methods and technologies for providing the software security and for identifying the failures and vulnerabilities of software showed that, although the analyzed methods and technologies have great potential for the field of software engineering, none of the known solutions are intended for identification and classification of software failures and vulnerabilities. Therefore, it is necessary to develop a method for ensuring the software security by identifying and classifying the failures and vulnerabilities, as well as to design and implement a technology for ensuring the software security by identifying and classifying the failures and vulnerabilities, which is the goal of this study. The developed in this paper method for ensuring the software security by identifying and classifying the failures and vulnerabilities provides a conclusion as to whether a failure occurred, and if a failure occurred, its type is issued to the user. In addition, the developed method for ensuring the software security by identifying and classifying the failures and vulnerabilities provides a conclusion as to whether a feature is a vulnerability, and if the feature is a vulnerability, its type is issued to the user. The paper also develops a technology for ensuring the software security by identifying and classifying the failures and vulnerabilities, which provides a conclusion on the presence or absence of software failure(s); conclusion on the presence or absence of software vulnerability(s); conclusion about the type of failure and the type of vulnerability in case of their presence, thanks to which the proposed technology is useful for software users due to the identification and classification of failures and vulnerabilities.

Keywords

Software security, failure of software, vulnerability of software, identifying the failures and vulnerabilities, classifying the failures and vulnerabilities.

1. Introduction

Modern software is a complex multifunctional product, during the creation of which errors, unintentional software defects, and unprotected functions inevitably occur. In today's digital era, software is widely adapted and has become an integral part of human society. Such widespread use of software is associated with the use of large and critical data that inevitably needs protection. It is critical to ensure that this software not only meets user needs or functional requirements, but it is equally important to ensure that this software is secure. Creating the secure software is a complex process. It is a process informally guided by common knowledge, best practices and undocumented expertise. In general, software security can be considered as one of the most important issues in the field of software development, as it can affect the performance of a software product through various technological vulnerabilities and threats.

ITTAP'2022: 2nd International Workshop on Information Technologies: Theoretical and Applied Problems, November 22–24, 2022, Ternopil, Ukraine
EMAIL: tat_yana@ukr.net (T. Hovorushchenko); p.t.popov@city.ac.uk (P. Popov); medza@ukr.net (D. Medzatyi); voichury@khmnu.edu.ua (Y. Voichur)
ORCID: 0000-0002-7942-1857 (T. Hovorushchenko); 0000-0002-3434-5272 (P. Popov); 0000-0002-1879-2945 (D. Medzatyi); 0000-0003-3085-7315 (Y. Voichur)

Software security is the property of certain software to function without various negative consequences for a specific computer system. The reasons leading to a security breach can be different: software failures, software vulnerabilities due to programmer errors and defects in programs.

Failure of software is an event characterized by software malfunction, as a result of which the software stops performing its functions (in whole or in part) [1-5].

Vulnerability of software is a software flaw (software design flaw, programming error, use of malicious software), when used, it is possible to intentionally violate the integrity of the software and cause its incorrect operation; it is the software's inability to resist the implementation of a certain threat or set of threats [6-10].

Thousands of new vulnerabilities are discovered every year, requiring companies to patch operating systems and applications, as well as reconfigure security settings across their entire network environment. To proactively address vulnerabilities before they can be exploited for a cyberattack, organizations that take the security of their network environment seriously conduct vulnerability management to ensure the highest level of security possible.

Detecting the vulnerability of software code is an important method of ensuring software security. Today, as the size and complexity of software grows rapidly, vulnerabilities become more diverse and harder to identify.

The main reasons for the appearance of vulnerabilities are:

1. Shared use of resources and simplification of information exchange between network nodes
2. Significant complication of software
3. Lack of complete information about the object and the use of search mechanisms
4. Unreliable data sources and a huge number of attackers
5. Low qualification of software users, especially in matters of information protection - the software is unable to resist threats from attackers, if users under their influence unknowingly perform destructive actions
6. Complexity of new technologies
7. The trend of combining data and program code, embedding program code (macros, scripts) into documents
8. The lag in the development of the legal framework, standards from changes in information processing methods and technologies
9. Lack of safe processes in the life cycle of software development

The rapid growth of computing power of computers and volumes of processed data, the expansion of the range of tasks that are solved by software, make it difficult to carry out a full and detailed analysis of possible vulnerabilities and exclude the conditions for their appearance.

Currently, many leading scientists have conducted a number of studies on improving software security, but software vulnerabilities and failures still pose serious problems for software users, manifesting in information leaks, information loss, leading to financial and reputational losses. So, for example, due to software vulnerabilities, there was a leak of information in the form of access to 500 million records of Yahoo users [11]; the Equifax company lost information about 140 million people, which led to financial losses of 575 million dollars of USA [12]; attackers gained access to 50 million Facebook user profiles [13]; abduction of information about 600 thousand drivers and 57 million accounts of users of the Uber service, which led to financial losses of 148.1 million dollars of USA [14]; a hacker attack on Ukrainian government websites on January 14, 2022, caused by a vulnerability in the October CMS website content management system [15].

All major software security approaches are aimed at preventing total software failure, but not at identifying software failures and vulnerabilities. The success of software security approaches is only possible due to the identification and reduction of the number of errors (currently, the density of errors in software ranges from 2 to 100 errors per 1000 lines of code [16, 17]), therefore, the identification of software failures and vulnerabilities is *an urgent task* at the moment.

2. Literature Review

Let’s conduct the literature review on known methods and technologies for providing the software security and for identifying the failures and vulnerabilities of software – Table 1.

Table 1
Literature review on known methods and technologies for providing the software security and for identifying the failures and vulnerabilities of software

Method and/or technology	Providing the software security	Identifying the failures and vulnerabilities of software
Threatened-based Software Security Evaluation method and Security Evaluation Assistant (SEA) tool for improving the security evaluation process of software with focus on existing threatened entities of software and software threats [18]	yes	no
Method for security reassurance of software increments to ensure producing acceptably secure-by the business owner-software increments at the end of each iteration [19]	yes	no
Data-driven model for software security and methods for learning detailed software statistics while providing differential privacy for its users [20]	yes	no
Method for software security (CM-Sec) focusing on the end product by prioritizing countermeasures, which provides an extension to attack trees and a process for identification and prioritization of countermeasures [21]	yes	no
Q-learning method embedded as part of the software itself for providing the security mechanism that has ability to learn by itself for development of a temporary repair mechanism [22]	yes	no
Methods, techniques, and best practice requirements engineering and management as an emerging cloud service (SSREMaaES) and as a guideline on software security as a service [23]	yes	no
Methodology for minimizing software vulnerability for enhancing its security implemented in the processes of the software development life cycle [24]	yes	yes
Hierarchical software security case development method [25]	yes	no
Security modeling and verification framework of embedded software based on semiformal and formal methods ZMsec (Z-MARTE security model) [26]	yes	no
SMASHUP: a toolchain for unified verification of software co-designs [27]	yes	no
Method for identifying software security vulnerabilities from software requirement specifications written in Structured Object-oriented Formal Language [28]	yes	no
Formal method for modeling software architectures and evaluating their quality attributes (include security, dependability and performance) quantitatively and in a unified manner [29]	yes	no

Model of Trustworthy Scrum (TS) enabling the security activities to cooperate with the agile methods and to work in Scrum framework [30]	yes	no
Software failure analysis method based on the system reliability modeling with the System-Theoretic Accident Modeling and Processes (STAMP) [31]	yes	yes
Using the pattern position distribution as features for detecting the software failure [32]	no	yes
Taxonomy for identifying software failure modes, which provide input to the risk analysis of software-intensive systems [33]	no	yes
Cascade fault localization method and software tool called CaFL for help of speed up labor-intensive process of identification of the root cause of a manifested failure via a combination of weakest precondition computation and constraint solving [34]	yes	yes
Method, which the causes of failures detects by conducting root cause analysis [35]	yes	yes
Failure Identification for Complex Mission Analysis (FICMA) method provides both an overall failure analysis on a system's functionality as well as a mission-based failure analysis [36]	yes	yes
Failure prediction algorithm based on multi-layer Bidirectional Long Short Term Memory (Bi-LSTM) [37]	yes	yes
A method for identifying software data flow vulnerabilities based on the dendritic cell algorithm and the improved convolutional neural network for effectively solving the transmission errors in software data flow [38]	yes	yes
Method based on the concept of mutual information that detect and isolate software vulnerabilities at a fine-grained level in both unsupervised and semi-supervised contexts [39]	yes	yes
Pangr: an entire system for automatic vulnerability detection, exploitation, and patching [40]	no	yes
Automated method for determining the code evidence for the presence of vulnerabilities in retro software versions [41]	yes	yes
Pattern-based vulnerability discovery approach based on static analysis, machine learning, and graph mining with a high focus on practical requirements [42]	yes	yes
Software source code vulnerability detection method based on Convolution Neural Networks (CNN) and Global Average Pooling (GAP) interpretability model [43]	yes	yes
VUDENC (Vulnerability Detection with Deep Learning on a Natural Codebase): a deep learning-based vulnerability detection tool that automatically learns features of vulnerable code from a large and real-world Python codebase [44]	yes	yes

The conducted literature review on known methods and technologies for providing the software security and for identifying the failures and vulnerabilities of software showed that, although the analyzed methods and technologies have great potential for the field of software engineering, none of the known solutions are intended for identification and classification of software failures and vulnerabilities according to the rules for classifying the failures and to the rules for classifying the vulnerabilities. Therefore, it is necessary to develop a method for ensuring the software security by identifying and classifying the failures and vulnerabilities based on the developed by authors in [45] rules for classifying the failures and the vulnerabilities, as well as to design and implement a technology for ensuring the software security by identifying and classifying the failures and vulnerabilities, which is *the goal of this study.*

3. Method and Technology for Ensuring the Software Security by Identifying and Classifying the Failures and Vulnerabilities

Considering the rules for classifying the failures and vulnerabilities of software developed by the authors in [45], let's develop questionnaires for collecting the information about failure(s) and vulnerability(s) that occurred during the software's operation.

Questionnaire for collecting the information about failure(s):

1. Has the software operational (workable) state after termination of the operation of the software?
2. Was there a loss of data during the termination of the operation of the software?

Each of the questions in the questionnaire for collecting the information about the failure(s) can have "yes" or "no" answer.

Rules for the classification of failures based on the analysis of answers to questions of questionnaire for collecting the information about the failure(s):

1. If software user gives the answer "yes" to the first question of the questionnaire for collecting the information about the failure(s) and the answer "no" to the second question of the questionnaire for collecting the information about the failure(s), then the variable *sf = 1*
2. If software user gives the answer "yes" to the first question of the questionnaire for collecting the information about the failure(s) and the answer "yes" to the second question of the questionnaire for collecting the information about the failure(s), then the variable *sf = 2*
3. If software user gives the answer "no" to the first question of the questionnaire for collecting the information about the failure(s), then the variable *sf = 3*

Questionnaire for collecting the information about vulnerability(s):

1. Did the software stop functioning for a time exceeding the specified threshold time during the execution of a certain feature?
2. Has there been a loss of data completeness after performing a certain feature?
3. Has there been a data leak after performing a certain feature?
4. Did it become impossible to obtain the information permitted to the user after performing a certain feature?

Each of the questions in the questionnaire for collecting the information about vulnerability(s) can have "yes" or "no" answer.

Rules for the classification of vulnerabilities based on the analysis of answers to questions of questionnaire for collecting the information about vulnerability(s):

1. If software user gives the answer "yes" to the first question of the questionnaire for collecting the information about vulnerability(s), then the element of the matrix *sv[1,1] = 1*
2. If software user gives the answer "yes" to the second question of the questionnaire for collecting the information about vulnerability(s), then the element of the matrix *sv[1,2] = 1*
3. If software user gives the answer "yes" to the third question of the questionnaire for collecting the information about vulnerability(s), then the element of the matrix *sv[1,3] = 1*
4. If software user gives the answer "yes" to the fourth question of the questionnaire for collecting the information about vulnerability(s), then the element of the matrix *sv[1,4] = 1*

Therefore, questionnaires for collecting the information about failure(s) and for collecting the information about vulnerability(s), as well as rules for the classification of failures based on the analysis of answers to questions of questionnaire for collecting the information about the failure(s) and rules for

the classification of vulnerabilities based on the analysis of answers to questions of questionnaire for collecting the information about vulnerability(s) have been developed. The developed rules make it possible to identify and classify failure(s) and vulnerability(s) of software that occurred during the software's operation.

Method for ensuring the software security by identifying and classifying the failures and vulnerabilities consists of the following steps:

1. variable *sf = 0*; filling the first row of the *sv* matrix with zeros; filling in the second line of the matrix *sv* in order to further form a conclusion about the type of vulnerability(s): *sv[2,1] = "the feature of the software is a vulnerability of correct operation"; sv[2,2] = "the feature of the software is a vulnerability of integrity of information"; sv[2,3] = "the feature of the software is a vulnerability of privacy of information"; sv[2,4] = "the feature of the software is the vulnerability of the availability of information"*
2. conducting the software user survey (using compiled questionnaires for collecting the information about failure(s) and vulnerability(s))
3. analysis of the answers given by the user to the questions of questionnaire for collecting the information about failure(s) using the rules for the classification of failures, and forming the value of the variable *sf*
4. if *sf=1*, then the user is given the conclusion "the software failure is insignificant", otherwise, if *sf=2*, the user is given the conclusion "the software failure is significant", otherwise if *sf=3*, the user is given the conclusion "the software failure is critical", otherwise, if *sf=0*, the user is given the conclusion "software failures did not occur"
5. analysis of the answers given by the user to the questions of questionnaire for collecting the information about vulnerability(s) using the rules for the classification of vulnerabilities, and filling the first row of the *sv* matrix
6. if *sv[1,i]=1 (i=1..4)*, then the user is given the conclusion about the type(s) of vulnerability – element *sv[2,i] (i=1..4)* of the *sv* matrix, otherwise, if all the elements of the first row of the *sv* matrix are equal to 0, then the user is given the conclusion "the feature of the software is not a vulnerability"

The developed method for ensuring the software security by identifying and classifying the failures and vulnerabilities provides a conclusion as to whether a failure occurred, and if a failure occurred, its type is issued to the user. In addition, the developed method for ensuring the software security by identifying and classifying the failures and vulnerabilities provides a conclusion as to whether a feature is a vulnerability, and if the feature is a vulnerability, its type is issued to the user.

The developed method is the basis for designing the technology for ensuring the software security by identifying and classifying the failures and vulnerabilities – Fig. 1.

The developed technology for ensuring the software security by identifying and classifying the failures and vulnerabilities provides a conclusion on the presence or absence of software failure(s); conclusion on the presence or absence of software vulnerability(s); conclusion about the type of failure and the type of vulnerability in case of their presence, thanks to which the proposed technology is useful for software users due to the identification and classification of failures and vulnerabilities.

4. Results & Discussion

Let's consider the operation of the developed method and technology for ensuring the software security by identifying and classifying the failures and vulnerabilities.

According to the first stage of the developed method for ensuring the software security by identifying and classifying the failures and vulnerabilities, the variable *sf* and the elements of the first row of the *sv* matrix were reset to zero, as well as the filling of the second row of the *sv* matrix.

According to the second stage of the developed method, a survey of the user of the software for keeping accounting was carried out using compiled questionnaires for collecting the information about failure(s) and vulnerability(s).

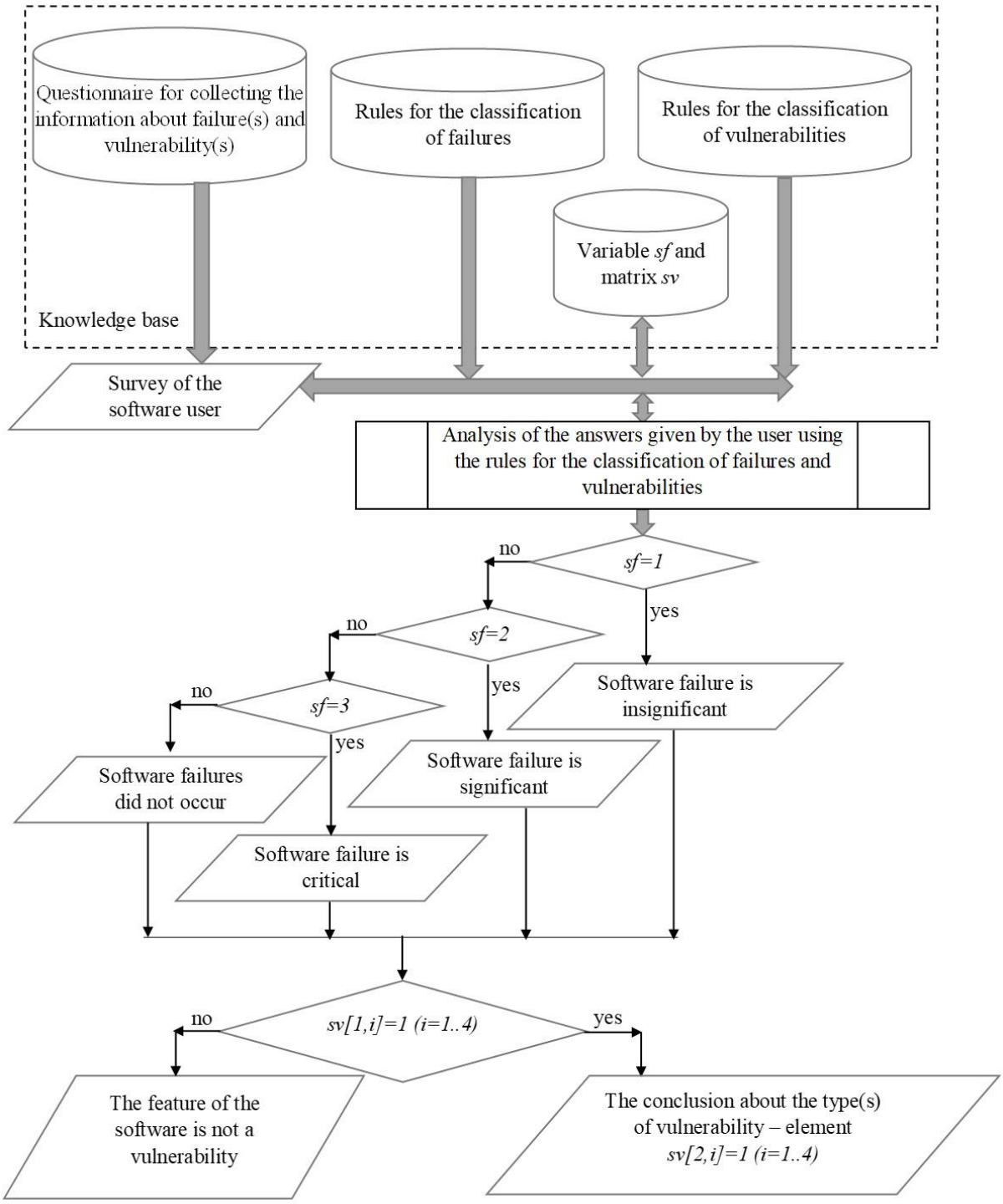

Figure 1: Technology for ensuring the software security by identifying and classifying the failures and vulnerabilities

According to the third stage of the developed method, the analysis of the answers given by the user to the questions of the questionnaire for collecting the information about failure(s) was performed using the rules for the classification of failures, and the formation of the value of the variable *sf* was performed. Since the user of the software for keeping accounting gives the answer "yes" to the first question of the questionnaire for collecting the information about failure(s) and answer "no" to the second question of the questionnaire for collecting the information about failure(s), then the variable *sf = 1*.

According to the fourth stage of the developed method, since *sf=1*, the user is given the conclusion "software failure is insignificant".

According to the fifth stage of the developed method for ensuring the software security by identifying and classifying the failures and vulnerabilities, an analysis of the answers given by the user to the questions of questionnaire for collecting the information about vulnerability(s) was performed using the rules for the classification of vulnerabilities, and filling the first row of the *sv* matrix was performed. The user of the software for keeping accounting answered "yes" to the first, third and fourth questions, so the *sv* matrix looks like – Table 2.

Table 2

Matrix *sv*, which contains signs of the presence or absence of a software vulnerability, as well as the type of vulnerability

	I column	II column	III column	IV column
I row	1	0	1	1
II row	the feature of the software is a vulnerability of correct operation	the feature of the software is a vulnerability of integrity of information	the feature of the software is a vulnerability of privacy of information	the feature of the software is the vulnerability of the availability of information

According to the sixth stage of the developed method, since *sv[1,1]=1*, the user is given the conclusion about the type of vulnerability – "The feature of the software is a vulnerability of correct operation" (element *sv[2,1]* of the *sv* matrix). Since *sv[1,3]=1*, the user is given the conclusion about the type of vulnerability – "The feature of the software is a vulnerability of privacy of information" (element *sv[2,3]* of the *sv* matrix). Since *sv[1,4]=1*, the user is given the conclusion about the type of vulnerability – "The feature of the software is the vulnerability of the availability of information" (element *sv[2,4]* of the *sv* matrix). Therefore, the considered feature of the software is the vulnerability of correct operation, privacy and availability of information.

The conducted experiment with the applying the developed method and technology for ensuring the software security by identifying and classifying the failures and vulnerabilities for software for keeping accounting showed that, based on a survey of the user of software for keeping accounting, a conclusion was given regarding the presence of an insignificant failure of the software for keeping accounting, as well as a conclusion regarding the presence of a vulnerability of the correct work, privacy and availability of information in the considered software for keeping accounting.

5. Conclusions

All major software security approaches are aimed at preventing total software failure, but not at identifying software failures and vulnerabilities. The success of software security approaches is only possible due to the identification and reduction of the number of errors, therefore, the identification of software failures and vulnerabilities is an urgent task at the moment.

The conducted literature review on known methods and technologies for providing the software security and for identifying the failures and vulnerabilities of software showed that, although the analyzed methods and technologies have great potential for the field of software engineering, none of the known solutions are intended for identification and classification of software failures and vulnerabilities. Therefore, it is necessary to develop a method for ensuring the software security by identifying and classifying the failures and vulnerabilities, as well as to design and implement a technology for ensuring the software security by identifying and classifying the failures and vulnerabilities, which is the goal of this study.

The questionnaires for collecting the information about failure(s) and for collecting the information about vulnerability(s), as well as rules for the classification of failures based on the analysis of answers to questions of questionnaire for collecting the information about the failure(s) and rules for the classification of vulnerabilities based on the analysis of answers to questions of questionnaire for

collecting the information about vulnerability(s) have been developed in this paper. The developed rules make it possible to identify and classify failure(s) and vulnerability(s) of software that occurred during the software's operation.

The developed in this paper method for ensuring the software security by identifying and classifying the failures and vulnerabilities provides a conclusion as to whether a failure occurred, and if a failure occurred, its type is issued to the user. In addition, the developed method for ensuring the software security by identifying and classifying the failures and vulnerabilities provides a conclusion as to whether a feature is a vulnerability, and if the feature is a vulnerability, its type is issued to the user.

The paper also develops a technology for ensuring the software security by identifying and classifying the failures and vulnerabilities, which provides a conclusion on the presence or absence of software failure(s); conclusion on the presence or absence of software vulnerability(s); conclusion about the type of failure and the type of vulnerability in case of their presence, thanks to which the proposed technology is useful for software users due to the identification and classification of failures and vulnerabilities.

The conducted experiment with the applying the developed method and technology for ensuring the software security by identifying and classifying the failures and vulnerabilities for software for keeping accounting showed that, based on a survey of the user of software for keeping accounting, a conclusion was given regarding the presence of an insignificant failure of the software for keeping accounting, as well as a conclusion regarding the presence of a vulnerability of the correct work, privacy and availability of information in the considered software for keeping accounting.

6. References

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

[2] 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.

[3] T. Hovorushchenko, O.Pomorova. Methodology of Evaluating the Sufficiency of Information on Quality in the Software Requirements Specifications, in: Proceedings of 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies, DeSSerT-2018, 2018, pp. 385-389. doi: 10.1109/DESSERT.2018.8409161.

[4] 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.

[5] 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.

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

[7] 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.

[8] T. Hovorushchenko, O. Pavlova, M. Bodnar. Development of an Intelligent Agent for Analysis of Nonfunctional Characteristics in Specifications of Software Requirements. Eastern-European Journal of Enterprise Technologies 1 2(97) (2019) 6-17. doi: 10.15587/1729-4061.2019.154074.

[9] T. Hovorushchenko, O. Pavlova. Method of Activity of Ontology-Based Intelligent Agent for Evaluating the Initial Stages of the Software Lifecycle. Advances in Intelligent Systems and Computing 836 (2019) 169-178. doi: 10.1007/978-3-319-97885-7_17.

[10] T. Hovorushchenko, O. Pavlova. Evaluating the Software Requirements Specifications Using Ontology-Based Intelligent Agent, in: Proceedings of 2018 IEEE International Scientific and

Technical Conference “Computer Science and Information Technologies”, CSIT-2018, Lviv, 2018, vol.1, pp. 215-218. doi: 10.1109/STC-CSIT.2018.8526730.
[11] Yahoo says 500 million accounts stolen, 2016. URL: https://money.cnn.com/2016/09/22/technology/yahoo-data-breach.
[12] 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.
[13] 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.
[14] 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.
[15] It could have been prevented: it became known why government websites "went down", 2022. URL: https://www.epravda.com.ua/news/2022/01/14/681448/.
[16] S. McConnell, Code complete, Microsoft Press, Redmond, 2013.
[17] 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.
[18] M. Razian, H. Sangchi. A Threatened-based Software Security Evaluation Method, in: Proceedings of 11th International ISC Conference on Information Security and Cryptology, ISCISC-2014, Tehran, 2014, pp. 120-125. doi: 10.1109/ISCISC.2014.6994034.
[19] L. ben Othmane, P. Angin, H. Weffers, B. Bhargava. Extending the Agile Development Process to Develop Acceptably Secure Software. IEEE Transactions on Dependable and Secure Computing 11 6 (2015) 497-509. doi: 10.1109/TDSC.2014.2298011.
[20] U. Erlingsson. Data-driven Software Security: Models and Methods, in: Proceedings of 29th IEEE Computer Security Foundations Symposium, CSF-2017, Lisbon, 2017, pp. 9-15. doi: 10.1109/CSF.2016.40.
[21] D. Baca, K. Petersen. Prioritizing Countermeasures through the Countermeasure Method for Software Security (CM-Sec). Lecture Notes in Computer Science 6156 (2010) 176-190. doi: 10.1007/978-3-642-13792-1_15.
[22] A. Randrianasolo, L. Pyeatt. Q-Learning: From Computer Network Security To Software Security, in: Proceedings of 13th International Conference on Machine Learning and Applications, ICMLA-2014, Detroit, 2014, pp. 257-262. doi: 10.1109/ICMLA.2014.47.
[23] M. Ramachandran. Software security requirements management as an emerging cloud computing service. International Journal of Information Management 36 4 (2016) 580-590. doi: 10.1016/j.ijinfomgt.2016.03.008.
[24] S. Farhan, M. Mostafa. A Methodology for Enhancing Software Security During Development Processes, in: Proceedings of 21st Saudi-Computer-Society National Computer Conference, NCC-2018, Riyadh, 2018, pp. 1-6. doi: 10.1109/NCG.2018.8593135.
[25] B. Xu, M. Lu, D. Zhang. A Layered Argument Strategy for Software Security Case Development, in: Proceedings of 28th IEEE International Symposium on Software Reliability Engineering, Toulouse, 2017, pp. 331-338. doi: 10.1109/ISSREW.2017.52.
[26] X. Hu, Y. Zhuang, F. Zhang. A security modeling and verification method of embedded software based on Z and MARTE. Computers & Security 88 (2020) No 10615. doi: 10.1016/j.cose.2019.101615.
[27] F. Lugou, L. Apvrille, A. Francillon. SMASHUP: a toolchain for unified verification of hardware/software co-designs. Journal of Cryptographic Engineering 7 1 (2017) 63-74. doi: 10.1007/s13389-016-0145-2.
[28] B. Emeka, S. Liu. Assessing and Extracting Software Security Vulnerabilities in SOFL Formal Specifications, in: Proceedings of 17th Annual International Conference on Electronics, Information, and Communication, ICEIC-2018, Honolulu, 2018, pp. 374-377. doi: 10.23919/ELINFOCOM.2018.8330613.
[29] A. Sedaghatbaf, M. Azgomi. Software Architecture Modeling and Evaluation Based on Stochastic Activity Networks. Lecture Notes in Computer Science 939 (2015) 46-53. doi: 10.1007/978-3-319-24644-4_3.

[30] G. Koc, M. Aydos, M. Tekerek. Evaluation of Trustworthy Scrum Employment for Agile Software Development based on the Views of Software Developers, in: Proceedings of 4th International Conference on Computer Science and Engineering, UBMK-2019, Samsun, 2019, pp. 63-67. doi: 10.1109/UBMK.2019.8907213.
[31] J. Song, H. Zhao, X. Li, Y. Yang, C. Liu, H. Li. A new software failure analysis method based on the system reliability modeling, in: Proceedings of IEEE 8th Joint International Information Technology and Artificial Intelligence Conference, ITAIC-2019, Chongqing, 2019, pp. 1143-1149. doi: 10.1109/ITAIC.2019.8785794.
[32] C. Li, Z. Chen, H. Du, H. Wang, G. Wilkie, J. Augusto, J. Liu. Using Pattern Position Distribution for Software Failure Detection. International Journal of Computational Intelligence Systems 6 2 (2013) 234-243. doi: 10.1080/18756891.2013.768442.
[33] C. Thieme, A. Mosleh, I. Utne, J. Hegde. Incorporating software failure in risk analysis - Part 1: Software functional failure mode classification. Reliability Engineering & System Safety 197 (2020) No 106803. doi: 10.1016/j.ress.2020.106803.
[34] Q. Yi, Z. Yang, J. Liu, C. Zhao, C. Wang. Explaining Software Failures by Cascade Fault Localization. ACM Transactions on Design Automation of Electronic Systems 20 3 (2015) No 41. doi: 10.1145/2738038.
[35] T. Lehtinen, M. Mantyla, J. Vanhanen, J. Itkonen, C. Lassenius. Perceived causes of software project failures - An analysis of their relationships. Information and Software Technology 56 6 (2014) 623-643. doi: 10.1016/j.infsof.2014.01.015.
[36] C. DeStefano, D. Jensen. Failure Identification for Mission Analysis for Complex Systems, in: Proceedings of ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Boston, 2015, No V01AT02A044. doi: 10.1115/DETC2015-47599.
[37] J. Gao, H. Wang, H. Shen. Task Failure Prediction in Cloud Data Centers Using Deep Learning. IEEE Transactions on Services Computing 15 3 (2022) 1411-1422. doi: 10.1109/TSC.2020.2993728.
[38] C. Luo, W. Bo, H. Kun, Y. Lou. Study on Software Vulnerability Characteristics and Its Identification Method. Mathematical Problems in Engineering (2020) No 1583132. doi: 10.1155/2020/1583132.
[39] V. Nguyen, T. Le, O. De Vel, P. Montague, J. Grundy, D. Phung. Information-theoretic Source Code Vulnerability Highlighting, in: Proceedings of International Joint Conference on Neural Networks, IJCNN-2021, Electr. Network, 2021. doi: 10.1109/IJCNN52387.2021.9533907.
[40] D. Liu, J. Wang, Z. Rong, X. Mi, F. Gai, T. Yong, B. Wang. Pangr: A Behavior-based Automatic Vulnerability Detection and Exploitation Framework, in: Proceedings of 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 12th IEEE International Conference on Big Data Science and Engineering, New York, 2018, pp. 705-712. doi: 10.1109/TrustCom/BigDataSE.2018.00103.
[41] V. Nguyen, S. Dashevskyi, F. Massacci. An automatic method for assessing the versions affected by a vulnerability. Empirical Software Engineering 21 6 (2016) 2268-2297. doi: 10.1007/s10664-015-9408-2.
[42] F. Yamaguchi. Pattern-based methods for vulnerability discovery. IT - Information Technology 59 2 (2017) 101-106. doi: 10.1515/itit-2016-0037.
[43] J. Wang, H. Kuang, R. Li, Y. Su. Software Source Code Vulnerability Detection Based on CNN-GAP Interpretability Model. Journal of Electronics & Information Technology 44 7 (2022) 2568-2575. doi: 10.11999/JEIT210412.
[44] L. Wartschinski, Y. Noller, T. Vogel, T. Kehrer, L. Grunske. VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python. Information and Software Technology 144 (2022) No 106809. doi: 10.1016/j.infsof.2021.106809.
[45] T. Hovorushchenko. Criteria and Rules for Classification of Software Failures and Vulnerabilities. CEUR-WS 3039 (2021) 217-224.