ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)*

UDC 040.43 DOI: https://doi.org/10.30837/ITSSI.2023.26.078

V. FEDORCHENKO, O. SHMATKO, I. MYKHAILENKO, V. TRETIAK, O. KOLOMIITSEV

INTEGRATING ANALYTICAL STATISTICAL MODELS, SEQUENTIAL PATTERN MINING, AND FUZZY SET THEORY FOR ADVANCED MOBILE APP RELIABILITY ASSESSMENT

The study presents a new method for evaluating the reliability of mobile applications using the Corcoran model. This model includes several aspects of reliability, including performance, reliability, availability, scalability, security, usability, and testability. The Corcoran model can be applied to evaluate mobile applications by analysing key reliability metrics. Using the model significantly improves the reliability assessment of applications compared to traditional methods, which are primarily focused on desktop and server configurations. The aim of the study is to offer a more optimised approach to evaluating the reliability of mobile applications. The paper examines the problems faced by mobile app developers. This study represents a new application of the Corcoran model in evaluating the reliability of mobile applications. This model is characterised by an emphasis on the use of quantitative statistics and the ability to provide an accurate estimate of the probability of failure without any inaccuracies, which distinguishes this model from other software reliability models. The paper suggests using a combination of analytical statistical models, data extraction methods such as sequential pattern analysis, and fuzzy set theory to implement the Corcoran model. The application of the methodology is demonstrated by studying software error reports and conducting a comprehensive statistical analysis of them. To improve the results of future research, the paper suggests making more extensive use of the Corcoran model in various mobile applications and environments. It is recommended to change the model to take into account the constantly changing characteristics of mobile applications and their increasing complexity. In addition, it is advisable to conduct additional research to improve the data mining methods used in the model and explore the possibility of integrating artificial intelligence for more advanced software reliability analysis. Applying the Corcoran model to the mobile app development process to evaluate reliability can significantly improve the quality of applications, leading to increased customer satisfaction and trust in mobile apps. This model can serve as a guide for developers and companies to evaluate and improve their applications, driving innovation and continuous improvement in the competitive mobile app sector.

Keywords: mobile application; software development; reliability assessmen; the Corcoran model.

Introduction

The term "software reliability" refers to the degree to which the procedures of the software development life cycle (SDLC) can be managed and controlled to produce reliable programs. This metric will be used until the conditions for completing the testing procedure are met. In addition, software reliability helps to maintain and predict the correctness of the software [1]. Software reliability engineering was developed to evaluate and quantify software quality. It demonstrates the fault-free operation of a program [2, 3]. Software reliability models are constantly being improved by both researchers and practitioners.

The following factors make it difficult to assess and predict the stability of a mobile application. First, mobile environments differ from traditional desktop computers and servers. Secondly, new forms of deficiencies are generated by the introduction of functionality and characteristics specific to the mobile context, such as energy, network, incompatibility, modified and restricted graphical user interface (GUI), interruptions, and notifications [4]. Third, there is a wide variety of mobile platforms and hardware capabilities. Fourth, due to consumer demand, the development of mobile applications has accelerated, and the functionality of mobile applications has become more complex [5]. And, of course, mobile devices break when an app is published. Software engineers rely primarily on problem reports submitted by users in addition to testing.

Researchers should spend more time analyzing software stability to determine its value for mobile apps. More accurate results and analyses can be obtained if software reliability testing takes into account the specifics of mobile applications.

Software engineers, enterprises, and research institutions are interested in being able to predict failures in mobile applications. Thus, we propose to evaluate the reliability of mobile applications based on bug reports and generate more accurate results.

Literature review

We identified several studies and systematic literature reviews (SLRs) related to software reliability [4, 5–8].

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4(26) ISSN 2524-2296 (online)

Several studies and literature reviews were found [9, 10–13] that focused on software dependency.

However, none of these studies specifically addressed the current state of mobile application reliability; rather, they all focused on traditional software. In order to determine what is the most up-to-date research in software reliability, Singhal [14] conducted a SLR analysis that included materials released before 2011. Ten years ago, when the widespread use of mobile applications was just beginning, this was the case. The study categorized 141 publications based on the research question, methodology (e.g., survey, theory), and environment (e.g., academic, industrial). Since the information available at the time was not sufficient to prove industrial validity, the study recommended additional industrial research. Due to the lack of standardized usage of words related to software reliability, the authors emphasized the importance of manual searching to find relevant material on the topic.

In their analysis of the literature from 1990 to 2010, Shahrokhni and Feldt [15] focused on software resilience, which is described as a reliability characteristic in various standards such as IEEE-STD 610.12-1990. For this study, the authors analyzed and classified 144 articles according to the following criteria: type of study (e.g., evaluation, experience report), contribution (e.g., tool, metrics), type of evaluation (e.g., academic, industrial), and development stage (e.g., requirements, design, and implementation). The lack of research on identifying and defining software sustainability requirements was the largest gap identified in the study.

Most studies have mainly focused on one component of reliability (invalid inputs), while others, such as unexpected events, timeouts, interruptions, and stressful execution settings, have been completely ignored. Febrero et al. [16] analyzed 503 articles from 2003 to 2014 as part of their modeling of SMS software reliability. Static and architectural reliability models, as well as software reliability growth models, Bayesian approaches, test-based methods, AI-based methods, and other types of reliability models were divided into five groups.

Finding that many studies do not meet the established quality requirements, the study identified a knowledge gap. To fill this gap, the same authors conducted a systematic literature review (SLR) on software reliability assessment using the ISO/IEC 25000 SQuaRE quality standard for 1991–2014.

According to the results of the latter study, insufficient attention has been paid to adjusting quality and reliability standards to take into account the interests of multiple stakeholders.

They also noted that the complexity of existing reliability models does not allow them to be used in routine situations. Lack of agreement and different definitions of reliability have also hindered the development of useful models. The authors noted, for example, that "reliability" and "fault tolerance" are often used synonymously, despite their differences.

They were more focused on how reliability models apply reliability requirements (e.g., ISO/IEC-25000 SQuaRE), whereas our work explores the current state of reliability in mobile applications. In addition, we review research conducted over the past six years or so.

Alhazzaa and Andrews [17] performed a state-of-the-art SMS in which they examined reliability growth patterns that take into account the development of software systems. They summarized the trends in terms of year of publication, location, and nature of the study (academic, industrial). The studies were categorized based on the proposed approach (analytical and curve fitting) and research style (empirical or non-empirical), as well as the scale of the solution (type and number of changes: one change point, multiple change points). In addition, they used the criteria of Ali et al. [18] to assess the reliability of empirical studies. They suggested that researchers look for higher quality empirical studies with closer cooperation with industry. In addition, these authors recommended further research on the following questions: how far into the future can these models look? When do professionals need to change the models or adjust their settings? All of these previous studies (including Alhazzaa and Andrews) agree that the solutions lacked industry validation because they were mostly studied in an academic context without involving or collaborating with practitioners throughout the study.

The proposed model

Thus, in order to successfully fulfill one of the main tasks of this work – creating an integrated model for assessing software reliability – we need to develop an idea of which model of software reliability analysis is most suitable for our project and how the statistical data for this model will be obtained.

Of all the software reliability assessment models considered, the Corcoran model was chosen as the most suitable for use in this work. There are several reasons for this, but the most important is the absence of the need for additional work (e.g., introduction of artificial errors)

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)

and the focus of this model on the use of quantitative statistical data about the project.

Corcoran's model is an example of an analytical statistical model of software reliability because it does not use test time parameters and only takes into account the result of N tests in which Ni errors of the i-th type are detected. The model uses variable failure probabilities for different types of errors.

Unlike other methods of this type, the Corcoran model estimates the probability of fault-free program execution at the time of evaluation [19].

– The model requires knowledge of the following indicators;

– The model contains non-static failure probabilities for different sources of errors and, accordingly, different probabilities of their correction;

– The model uses only such parameters as the result of N tests in which N_i errors of the i-th type are observed;

– Detection of errors of the i-th type during N tests occurs with probability a_i.

The reliability level indicator R is calculated by the formula:

$$R = \frac{N_0}{N} + \sum_{i=1}^{K} \frac{Y_i \times (N_i - 1)}{N}, \quad (1)$$

where N_0 – is the number of failed (or unsuccessful) tests performed in a series of N tests;

K – known number of error types;

Y_i – probability of errors

$$Y_i = \begin{cases} a_i, & if \ N_i > 0, \\ 0, & if \ N_i = 0; \end{cases} \quad (2)$$

a_i – probability of detecting errors of the i-th type during testing.

In this model, the probability of occurrence of a certain event is estimated based on a priori information or statistics for the previous period of software operation.

The number of tests N_i for the Corcoran formula for an incomplete set of test reports is defined as:

$$N_i = \frac{R_i \times N_t \times 0.6}{R_t}, \quad (3)$$

where R_i – number of reports imported to the system;

R_t – total number of reports on the Socorro server;

N_t – total number of product installations.

The algorithm of sequential pattern mining was chosen as a method of data mining. This choice was made under the influence of the availability of a large number of algorithms for solving similar problems, the ease of understanding the principles of these algorithms, and the high adaptability of this method to the required tasks. Sequential pattern mining is a data mining activity aimed at finding statistically significant patterns between data in which values are presented sequentially. As a rule, the values are considered discrete, which distinguishes this activity from data extraction from a time series. Sequential pattern mining is a special case of structured data mining [37]. In this paper, we will use algorithms to find the longest unified sequence. In computer science, the problem of finding the longest common sequence is to find the longest sequence (substring) or substrings that are common to two or more strings. For example, the longest common sequence of the strings "ABABC", "BABCA", and "ABCBA" is the string "ABC", which is three letters long. Other common sequences include "a", "AB", "B", "BA", and "C".

The problem of finding these sequences can be formulated as follows: given two lines S of length p and T of length q, you need to find the longest line that is common to S and T. Another interpretation of this problem is the problem of generalizing k-common sublines: given a set of lines $S = \{S_1, \ldots, S_K\}$, where $|S_i| = n_i i$, $|S_i| = n_i$, for every $2 \le k \le k$, you need to find the longest lines that occur inside at least K lines.

In this paper, we will continue to consider and use only the dynamic programming approach, since the length of lines in the subject area of this paper usually does not exceed 20 elements, but the simplicity of implementation and the visibility of the dynamic programming algorithm are much higher. To solve this problem using dynamic programming, you must first find the longest common suffix for all pairs of prefixes in the lines. The longest common suffix is calculated by the following formula:

$$LCSuff(S_{1..p}, T_{1..q}) = \begin{cases} LCSuff(S_{1..p-1}, T_{1..q-1}) + 1, & if \ S[p] = T[q] \\ 0, & otherwise, \end{cases}$$

where line S of length p;

line T of length q;

$LCSuff(S_{1..p}, T_{1..q})$ – is the longest line that is common to S and T.

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4(26)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

For example, for the strings "ABAB" and "BABA", the result of this algorithm is the following table 1.

For example, for the strings "ABAB" and "BABA", the result of this algorithm is the following table:

Table 1. *Example of using the dynamic programming algorithm*

		A	**B**	**A**	**B**
	0	0	0	0	0
B	0	0	1	0	1
A	0	1	0	2	0
B	0	0	2	0	3
A	0	1	0	3	0

The maximum of these common longest suffixes for possible prefixes must be the longest common sequence (subline) of lines S and T. These suffixes are underlined on the diagonals of Table 1. For this example, the longest common sequences are "BAB" and "ABA".

$$LCSubstr(S,T) = \max_{1\le i\le m, 1\le j\le n} LCSuff\left(S_{1..i}, T_{1..j}\right),$$

where $LCSubstr(S,T)$ – is the largest common sequence (subline) of lines S and T.

This formula can be extended for the case of comparing more than two lines by adding additional dimensions to the table, but this is not necessary in our case. To better establish patterns and relationships between error reports, it is also necessary to consider algorithms for measuring edit distance. The reason for this is the discrepancy between data in related reports and the need to reduce noise between data samples.

In computer science, edit distance is a way to quantify the similarity of two strings (e.g., two words) to each other by counting the minimum number of operations required to transform one string into another. Editing distance is used in natural language processing tasks where automatic spelling correction can identify possible edits for a misspelled word by selecting those words from the dictionary that have a low editing distance to that word. In bioinformatics, this distance can be used to quantify the similarity of DNA sequences, which can be represented as strings of letters A, C, G, and T.

Different definitions of edit distance use different sets of operations on strings. For example, Levenshtein distance uses deletion, insertion, or replacement of characters in a string. Since it is the most common metric, it is the Levenshtein distance that is usually referred to as "edit distance". The most common algorithm for finding edit distance uses a standard set of Levenshtein operations and determines the distance between $a = a_1...a_n$ and $b = b_1...b_m$ as d_{mn}, which is recursively calculated by the following formulas:

$$d_{i0} = \sum_{k=1}^{i} w_{del}(b_k) \qquad for\ 1 \le i \le m,$$

$$d_{0j} = \sum_{k=1}^{j} w_{ins}(a_k) \qquad for\ 1 \le j \le n,$$

$$d_{ij} = \begin{cases} d_{i-1,j-1} & for\ a_j = b_i \\ \min\begin{cases} d_{i-1,j} + w_{del}(b_i) \\ d_{i,j-1} + w_{ins}(a_i) \\ d_{i-1,j-1} + w_{sub}(a_j, b_i) \end{cases} & for\ a_j \ne b_j \end{cases} \qquad for\ 1 \le i \le m, 1 \le j \le n,$$

simple recursive method of calculating these formulas takes exponentially long. Therefore, as a rule, the calculations are performed using the Wagner and Fisher dynamic programming algorithm. After completing the Wagner–Fisher algorithm, the minimal sequence of editing operations can be read as the Return Path of the operations (starting with dmn) used during the dynamic programming algorithm.

Example from the practice

Below is an example of using this method to assess software reliability using the proposed approach.

The input data are data about 100 tests of the program. Out of 100 tests, 20 were successful (without failures), and in other cases, the data shown in Table 2 was obtained.

When all the necessary data are calculated, the Corcoran model must be applied to find the probability of program failure at the time of evaluation.

Thus, this approach requires a tool to analyze data from similar projects or analyze available statistics from the current project to establish the a_i parameter. Such a tool would be data mining methods using sequential pattern mining. Information on the total number of installations, the number of worlds and error groups, and the probability of each error group occurring will be used to calculate the software reliability index.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Table 2. *Example of using the Corcoran model (part 1)*

Type of error	Probability of error a_i	Number of N_i errors that occur during testing	Y_i *	$\left(Y_i * \left(N_i - 1\right)\right) / N$ **
1. calculation errors	0,09	5	0,09	0,0036
2. logical errors	0,26	25	0,26	0,0624
3. input/output errors	0,16	3	0,16	0,0032
4. data manipulation errors	0,18	0	0	0
5. communication errors	0,17	11	0,17	0,017
6. data definition errors	0,08	3	0,08	0,0016
7. database errors	0,06	4	0,06	0,0018

* – value is calculated by the formula (2)
** – value is calculated by the formula (1)

Table 2. *Example of using the Corcoran model (part 2)*

Initial data	
$N =$	100
$N_0 =$	20
$R =$	0,2896

Mozilla Firefox receives 2.5 million error messages every day. That is why analyzing and finding software errors (bugs) is a very difficult task. Although errors can appear in different system modules and components or on different pages of web applications, they can also be the result of a general program flaw (bug). That is why there is a need to analyze and automatically search for duplicate and related reports.

The whole process of grouping reports is as follows:

1) Reports are sent to the server where they are stored. If it works, the reports will be automatically imported from the Socorro server, an open source bug report server for Mozilla products.

2) The server automatically groups reports into categories according to the cause of the bug. Each category has at least one or more reports.

3) Developers (in our case, a user from the moderator group) assign the corresponding software defect record to the general report categories. One record can correspond to one or more categories, and one category can have zero or more defect records. Programmers (in our case, users of any group) can also be assigned to defect records to resolve an existing issue.

A typical bug report for the Fennec Android mobile browser (Firefox for Android) consists of two parts, shown in Figures 1–3.

As part of the Mozilla Crash Reports project, information from the Socorro server is processed and presented in the form of statistics. For example, you can view the number of reports per day, the number of product installations, or statistics on the number of errors and reports for different versions of the product at different times. But the most interesting thing is the ability to view automatically created groups of bug reports and assigned records of software defects.

The Mozilla algorithm is quite simple and sometimes inefficient. This algorithm compares only the error signature from the top form. This leads to the appearance of double groups of errors shown in Figures 4–6, which should actually be combined into one group.

This is the reason for considering the problem of grouping related reports. Since there is very little data to analyze this problem, this paper only considers data obtained from the Socorro server.

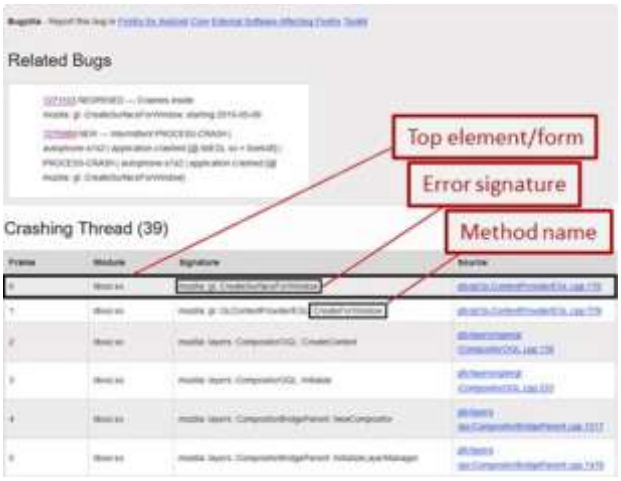

Fig. 1. Information from the defective flow

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4(26)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Details | Metadata | Modules | Raw Dump | Extensions

Signature	mozilla::gl::CreateSurfaceForWindow **More Reports** **Search**
UUID	7b376646-492e-4a56-8501-cddbc2160518
Date Processed	2016-05-18T22:03:50.122684+00:00
Uptime	7
Last Crash	16 seconds before submission
Install Age	14440 since version was first installed.
Install Time	2016-05-18 18:03:01
Product	FennecAndroid
Version	49.0a1
Build ID	20160518030234
Release Channel	nightly
OS	Android
OS Version	0.0.0 Linux 3.4.0-perf-gdfcced7 #1 SMP PREEMPT Thu Aug 27 17:06:58 2015 armv7l
Build Architecture	arm
Build Architecture Info	ARMv7 Qualcomm Krait features: swp,half,thumb,fastmult,vfpv2,edsp,neon,vfpv3,tls,vfpv4,idiva,idivt | 4
Crash Reason	SIGSEGV
Crash Address	0x0
User Comments	
App Notes	EGL? EGL+ GL Context? GL Context+ AdapterDescription: 'Model: SOL23, Product: SOL23_jp_kdi, Manufacturer: Sony, Hardware: qcom, OpenGL: Qualcomm -- Adreno (TM) 330 -- OpenGL ES 3.0 V@66.0 AU@ (CL@)' FP(D000-L10100-W00000000-T0100) Sony SOL23 KDDI/SOL23_jp_kdi/SOL23:4.4.2/14.3.C.0.300/035_jg:user/release-keys
Processor Notes	processor_prod-processor-i-9e74b743_17927; MozillaProcessorAlgorithm2015; skunk_classifier: reject - not a plugin hang
EMCheckCompatibility	True
Winsock LSP	None
Adapter Vendor ID	Qualcomm
Adapter Device ID	Adreno (TM) 330
Android CPU ABI	armeabi-v7a
Android Manufacturer	Sony
Android Model	SOL23
Android Version	19 (REL)

Fig. 2. General information from the report

FennecAndroid 49.0a1 Crash Report [@ mozilla::gl::CreateSurfaceForWindow]

ID: 7b376646-492e-4a56-8501-cddbc2160518
Signature: mozilla::gl::CreateSurfaceForWindow

Fig. 3. Report ID and signature

Fig. 4. Example of double error groups

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)

Fig. 5. Report of the first group and the corresponding signature

Fig. 6. Report of the second group and the corresponding signature

Due to the imperfection or lack of tools for automatic categorization of bug reports in common bug trackers, in this article we use three rules to find related reports more accurately. These rules were built based on the analysis of the structure of reports from the Socorro server and look like this:

1) Comparison of error signatures

Examples of using this rule are the following cases:

– nsDiskCacheStreamIO:FlushBufferToFile()
Strstr | nsDiskCacheStreamIO:FlushBufferToFile()
– nsStyleContext::Release()
nsStyleContext::nsStyleContext

As you can see from the above examples, the comparison should not be performed carefully, letter by letter, taking into account the structure and special notation of the record. Please note that the signature of the highest form method with fully filled fields will be used as the signature.

To implement this rule, we will use the method of measuring the edit distance, namely the Levenshtein distance, using the Wagner–Fisher algorithm. To use this algorithm, the signature will be split into a sequence of components that will act as individual letters.

2) Uppercase comparison

This rule works on the same principle as the comparison of error signatures, but it compares not the signatures, but the file path specified in the upper forms. It's important to remember that in this article, the data from the highest form that has non-zero attributes in all its fields will be used to compare top forms and signatures.

3) Comparing frequent, closely spaced subsets of forms

This rule means that two reports are related if they have one or more of the same call stack paths or forms. For example, the reports "ABCDEF", "DEFA", and "BDEF" have the longest common sequence – "DEF". In our case, instead of letters, we will use parts of the call stack.

To determine the length of a common element sufficient to establish a relationship, a threshold function will be used that takes the total length of the stack, the length of the common sequence, and its distance from the highest form. To determine the longest common sequences, a sequential pattern extraction algorithm will be used, namely the dynamic programming algorithm discussed above.

The previously mentioned mathematical methods and functions related to fuzzy sets will be used to evaluate the performance of these rules.

To calculate the degree of similarity between two reports, two fuzzy models were used: a model for analyzing the similarity of forms available in the reports and a model for analyzing the similarity of the reports themselves.

Conclusions

In summary, the Corcoran model offers a valuable and comprehensive approach to assessing the reliability of mobile applications, taking into account various dimensions such as performance, reliability, availability, scalability, security, usability, maintainability, and testability. By implementing this model, developers and organizations can gain valuable insights into the strengths and weaknesses of their applications, allowing them to make informed decisions and prioritize improvements.

Implementing the Corcoran model in the software development process can lead to higher quality mobile apps, increased end-user satisfaction, and increased

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4(26)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

trust in the mobile app ecosystem. In addition, this model can serve as a benchmark for developers and organizations to compare their apps to industry standards and competitors, promoting innovation and continuous improvement of mobile apps.

In summary, the Corcoran model for assessing mobile application reliability represents a significant advancement in mobile application assessment, enabling organizations to better meet user needs and expectations while ensuring a high level of reliability in the increasingly competitive mobile application market.

In the future, it is planned to expand the use of the proposed approach based on the Corcoran model for various mobile applications and environments. In the future, it is proposed to modify the model to take into account the ever-changing characteristics of mobile applications and their growing complexity. In addition, it is desirable to conduct additional research to improve the data mining methods used in the proposed approach and to explore the possibility of integrating artificial intelligence for more advanced software reliability analysis.

References

1. Mangla, M., Sharma, N., Mohanty, S. N. (2021), "A sequential ensemble model for software fault prediction", *Innovations in Systems and Software Engineering*, P. 1–8. DOI: https://doi.org/10.1007/s11334-021-00390-x

2. Khuat, T. T., Le, M. H. (2019), "Ensemble learning for software fault prediction problem with imbalanced data", *International Journal of Electrical & Computer Engineering (2088–8708)*, Vol. 9, No 4. DOI: 10.11591/ijece.v9i4.pp3241-3246

3. Sales, A. M. A. et al. (2023), "Proposal of fault detection and diagnosis system architecture for residential air conditioners based on the Internet of Things", *2023 IEEE International Conference on Consumer Electronics (ICCE)*, P. 1–5. DOI: 10.1109/ICCE56470.2023.10043408

4. Joorabchi, M. E., Mesbah, A., Kruchten, P. (2013), "Real challenges in mobile app development", *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, P. 15–24. DOI: 10.1109/ESEM.2013.9

5. Heitkötter, H., Hanschke, S., Majchrzak, T. A. (2012), "Evaluating cross-platform development approaches for mobile applications", *Web Information Systems and Technologies: 8th International Conference, WEBIST 2012*, Revised Selected Papers 8, P. 120–138. DOI: https://doi.org/10.1007/978-3-642-36608-6_8

6. Zhang, H., Babar, M. A. (2013), "Systematic reviews in software engineering: An empirical investigation", *Information and Software Technology*, Vol. 55, No 7, P. 1341–1354. DOI: https://doi.org/10.1016/j.infsof.2012.09.008

7. Garousi, V., Mäntylä, M. V. (2016), "A systematic literature review of literature reviews in software testing", *Information and Software Technology*, Vol. 80, P. 195–216. DOI: https://doi.org/10.1016/j.infsof.2016.09.002

8. Felizardo, K. R. et al. (2017), "Defining protocols of systematic literature reviews in software engineering: a survey", *43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, P. 202–209. DOI: 10.1109/SEAA.2017.17

9. Pachouly, J. et al. (2022), "A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools", *Engineering Applications of Artificial Intelligence*, Vol. 111, 104773 p. DOI: https://doi.org/10.1016/j.engappai.2022.104773

10. Son, L. H., Pritam, N., Khari, M., Kumar, R., Phuong, P. T. M., & Thong, P. H. (2019), "Empirical study of software defect prediction: a systematic mapping", *Symmetry*, 11(2), 212 p. DOI: https://doi.org/10.3390/sym11020212

11. Li, Z., Jing, X. Y., Zhu, X. (2018), "Progress on approaches to software defect prediction", *Iet Software*, Vol. 12, No 3, P. 161–175. DOI: https://doi.org/10.1049/iet-sen.2017.0148

12. Zhou, T. et al. (2019), "Improving defect prediction with deep forest", *Information and Software Technology*, Vol. 114, P. 204–216. DOI: https://doi.org/10.1016/j.infsof.2019.07.003

13. Thota, M. K., Shajin, F. H., & Rajesh, P. (2020), "Survey on software defect prediction techniques", *International Journal of Applied Science and Engineering*, 17(4), P. 331–344. DOI: https://doi.org/10.6703/IJASE.202012_17(4).331

14. Singhal, S. et al. (2021), "Systematic literature review on test case selection and prioritization: A tertiary study", *Applied Sciences*, Vol. 11, No 24, P. 12121. DOI: https://doi.org/10.3390/app112412121

15. Shahrokni, A., Feldt R. (2013), "A systematic review of software robustness", *Information and Software Technology*, Vol. 55, No 1, P. 1–17. DOI: https://doi.org/10.1016/j.infsof.2012.06.002

16. Febrero, F., Calero, C., Moraga, M. Á. (2016), "Software reliability modeling based on ISO/IEC SQuaRE", *Information and Software Technology*, Vol. 70, P. 18–29. DOI: https://doi.org/10.1016/j.infsof.2015.09.006

17. Ali, S. et al. (2009), "A systematic review of the application and empirical investigation of search-based test case generation", *IEEE Transactions on Software Engineering*, Vol. 36, No 6, P. 742–762. DOI: 10.1109/TSE.2009.52

18. Rathi, G., Tiwari, U. K., Singh, N. (2022), "Software Reliability: Elements, Approaches and Challenges", *International Conference on Advances in Computing, Communication and Materials (ICACCM)*. P. 1–5. DOI: 10.1109/ICACCM56405.2022.10009422

Received 28.11.2023

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)*

Відомості про авторів / About the Authors

Шматко Олександр Віталійович – PhD, доцент, Національний технічний університет "Харківський політехнічний інститут", доцент кафедри програмної інженерії та інтелектуальних технологій управління, Харків, Україна; e-mail: oleksandr.shmatko@khpi.edu.ua; ORCID ID: http://orcid.org/0000-0002-2426-900X

Коломійцев Олексій Володимирович – доктор технічних наук, професор, Національний технічний університет "Харківський політехнічний інститут", професор кафедри комп'ютерної інженерії та програмування, Харків, Україна; e-mail: alexus_k@ukr.net; ORCID ID: http://orcid.org/0000-0001-8228-8404

Федорченко Володимир Миколайович – PhD, доцент, Харківський національний університет радіоелектроніки, доцент кафедри електронних обчислювальних машин, Харків, Україна; e-mail: volodymyr.fedorchenko@nure.ua; ORCID ID: http://orcid.org/0000-0001-7359-1460

Михайленко Ірина Володимирівна – PhD, доцент, Харківський національний автомобільно-дорожній університет, доцент кафедри вищої математики, Харків, Україна; e-mail: irinaamih@gmail.com; ORCID ID: http://orcid.org/0000-0002-5961-3616

Третяк Вячеслав Федорович – кандидат технічних наук, доцент, Харківський національний університет Повітряних Сил імені Івана Кожедуба, науковий співробітник наукового центру Повітряних Сил, Харків, Україна; e-mail: slava_tr@ukr.net; ORCID ID: http://orcid.org/0000-0003-2599-8834

Shmatko Oleksandr – PhD, Associate Professor, National Technical University "Kharkiv Polytechnic Institute", Associate Professor at the Department of Software Engineering and Intelligent Management Technologies, Kharkiv, Ukraine.

Kolomiitsev Oleksii – Doctor of Sciences (Engineering), Professor, National Technical University "Kharkiv Polytechnic Institute", Professor at the Department of Computer Engineering and Programming, Kharkiv, Ukraine.

Fedorchenko Volodymyr – PhD, Associate Professor, Kharkiv National University of Radio Electronics, Associate Professor at the Department of Electronic Computers, Kharkiv, Ukraine.

Mykhailenko Iryna – PhD, Associate Professor, National Automobile and Road University, Associate Professor at the Department of Higher Mathematics, Kharkiv, Ukraine.

Tretiak Viacheslav – PhD, Associate Professor, Ivan Kozhedub Kharkiv National Air Force University, Senior Researcher, Kharkiv, Ukraine.

ІНТЕГРАЦІЯ АНАЛІТИЧНИХ СТАТИСТИЧНИХ МОДЕЛЕЙ, ПОСЛІДОВНОГО АНАЛІЗУ ЗАКОНОМІРНОСТЕЙ ТА ТЕОРІЇ НЕЧІТКИХ МНОЖИН ДЛЯ РОЗШИРЕНОГО ОЦІНЮВАННЯ НАДІЙНОСТІ МОБІЛЬНИХ ЗАСТОСУНКІВ

Дослідження є новим методом оцінки надійності мобільних додатків за допомогою моделі Коркорана. Ця модель включає в себе кілька аспектів надійності, включаючи продуктивність, надійність, доступність, масштабованість, безпеку, зручність використання і тестованість. Модель Коркорана може бути застосована для оцінки мобільних додатків шляхом аналізу основних показників надійності. Використання моделі значно поліпшує оцінку надійності застосунків в порівнянні з традиційними методами, які в першу чергу орієнтовані на конфігурації настільних комп'ютерів і серверів. Мета дослідження-запропонувати більш оптимізований підхід до оцінки надійності мобільних додатків. В роботі розглянуто проблеми з якими стикаються розробники мобільних застосунків. Це дослідження представляє нове застосування моделі Коркорана в області оцінки надійності мобільних додатків. Ця модель відрізняється акцентом на використання кількісної статистики та здатністю надавати точну оцінку ймовірності збою без будь-яких неточностей, що відрізняє цю модель від інших моделей надійності програмного забезпечення. В роботі пропонується використання комбінації аналітичних статистичних моделей, методів видобутку даних, таких як послідовний аналіз шаблонів, і теорію нечітких множин для реалізації моделі Коркорана. Застосування методології продемонстровано на прикладі дослідження звітів про помилки програмного забезпечення та проведення їх всебічного статистичного аналізу. Щоб покращити результати майбутніх досліджень, в роботі пропонується більш широко використовувати модель Коркорана у різних мобільних додатках та середовищах. Рекомендується змінити модель, щоб врахувати постійно мінливі характеристики мобільних додатків і їх зростаючу складність. Крім того, бажано провести додаткові дослідження для вдосконалення методів видобутку даних, що використовуються в моделі, та вивчити можливість інтеграції штучного інтелекту для більш просунутого аналізу надійності програмного забезпечення. Застосування моделі Коркорана у процесі розробки мобільних додатків для оцінки надійності може значно підвищити якість додатків, що призведе до підвищення рівня задоволеності клієнтів та довіри до мобільних додатків. Ця модель може слугувати орієнтиром для розробників та компаній при оцінці та вдосконаленні своїх додатків, сприяючи інноваціям та постійному вдосконаленню в конкурентному секторі мобільних додатків.

Ключові слова: мобільний застосунок; розробка програмного забезпечення; оцінювання надійності; модель Коркорана.

Бібліографічні описи / Bibliographic descriptions

Федорченко В. М., Шматко О. В., Михайленко І. В., Третяк В. Ф., Коломійцев О. В. Інтеграція аналітичних статистичних моделей, послідовного аналізу закономірностей та теорії нечітких множин для розширеного оцінювання надійності мобільних застосунків. *Сучасний стан наукових досліджень та технологій в промисловості*. 2023. № 4 (26). С. 78–86. DOI: https://doi.org/10.30837/ITSSI.2023.26.078

Fedorchenko, V., Shmatko, O., Mykhailenko, I., Tretiak, V., Kolomiitsev, O. (2023), "Integrating analytical statistical models, sequential pattern mining, and fuzzy set theory for advanced mobile app reliability assessment", *Innovative Technologies and Scientific Solutions for Industries*, No. 4 (26), P. 78–86. DOI: https://doi.org/10.30837/ITSSI.2023.26.078