

В. ЯКОВИНА, Н. ШАХОВСЬКА, Я. МАТВІЙЧУК, Є. ЗАСОБА

Національний університет «Львівська політехніка»

## РОЗРОБЛЕННЯ АЛГОРИТМУ ПРОГНОЗУВАННЯ ДЕФЕКТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ КАРТ КОХОНЕНА ТА ІЄРАРХІЧНОЇ КЛАСТЕРИЗАЦІЇ

У роботі розроблено вдосконалений алгоритм прогнозування дефектів програмного забезпечення на основі поєднання карти Кохонена та ієрархічної кластеризації. Раніше для побудови моделей прогнозування дефектів програмного забезпечення використовувались різні методи класифікації, починаючи від простих, таких як логістична регресія, і закінчуючи сучасними методами, наприклад багатовимірне адаптивне зрощення регресії. Однак наявна література все ще не дозволяє зробити однозначний висновок щодо вибору найкращого класифікатора та спроб різних вимірів для подолання потенційних упереджень пропонується. Метою статті є аналіз метрик програмного коду для виявлення залежностей між схильністю до дефектів програмного модуля та його метриками. У цьому дослідженні було використано JM1 загальнодоступний набір даних NASA з PROMISE Software Engin-Reering Repository.

Ключові слова: аналіз дефектів програмного забезпечення, прогнозування, ієрархічна кластеризація, карти Кохонена.

V. YAKOVYNA, N. SHAKHOVSKA, Ya. MATVIYCHUK, Ye. ZASOBA

Lviv Polytechnic National University

## DEVELOPMENT OF SOFTWARE DEFECT PREDICTION ALGORITHM BASED ON KOHONEN MAPS AND HIERARCHICAL CLUSTERING

In this work, an improved software defect prediction algorithm based on a combination of the Kohonen map and hierarchical clustering was developed. The algorithm uses a gradually decreasing learning rate to fine-tune a new era. As a result, the center is set in a position that satisfactorily clusters the examples for which the neuron is the winner. The property of topological ordering is achieved in the algorithm using the concept of neighborhood. Neighborhood is the result of agglomerative hierarchical clustering. The proposed algorithm shows higher accuracy than other classification algorithms. Pre-processing allows us to improve the quality of analysis by dividing all data into two clusters. This study used a public dataset from the PROMISE software engineering repository. The JM1 dataset for software defect prediction is selected. The source of this dataset is NASA Metrics Data Program.

Previously, various classification methods were used to build software defect prediction models, from simple ones, such as logistic regression, to modern methods, such as multidimensional adaptive regression splicing. However, the available literature still does not allow to make an unambiguous conclusion about the choice of the best classifier and attempts of different dimensions to overcome potential biases are offered. The aim of the article is to analyze the metrics of the program code to identify the relationships between the susceptibility to defects of the software module and its metrics.

The task of classification is to assign an object to one of the predefined classes based on its formalized characteristics. Each of the classified objects is represented as an N-dimensional vector, each dimension of which corresponds to one of the features of the object. The binary classification model should evaluate the impact of each parameter or group of parameters on the classification of defects. The analysis process consists of two phases. In the first phase, all parameters were considered. In the second phase, the influence of each parameter was studied.

We use an ensemble of models - the Kohonen map and hierarchical clustering. The heat map shows the weight of attributes and their grouping, as well as clusters data. The Kohonen map uses unsupervised learning, and the learning set consists only of the values of the input variables. Kohonen's map is studied by successive approximation. Starting with a randomly selected initial location of the centers, the algorithm is gradually improved to collect training data. Kohonen's basic iterative algorithm successively goes through a number of epochs; one case study is processed for each epoch. The input signals are fed sequentially to the network. The desired output signals are not determined. After processing a sufficient number of input vectors, the synaptic weights of the network are determined by clusters. In addition, the scales are organized in such a way that topologically close nodes are sensitive to such input signals.

Keywords: software defects analysis, prediction, hierarchical clustering, Kohonen maps.

### Вступ

Автоматизовані системи керують різноманітними технічними пристроями, що виконують функції, які мають важливе значення для безпеки, починаючи від електростанцій і закінчуючи автономними транспортними засобами, що контролюються за допомогою штучного інтелекту (ШІ). Невід'ємною частиною всіх цих систем є програмне забезпечення, яке або виконує функцію управління, або реалізує алгоритми штучного інтелекту. Вартість відмови працездатності такого програмного забезпечення дуже висока, це може призвести до подій різного ступеня тяжкості, починаючи від економічних збитків до завдання шкоди життю та здоров'ю людей. Отже, із збільшенням відповідальності функціонування комп'ютерних систем вимоги до їх надійності та безпеки значно зростають. Щоб задовольнити ці вимоги, є дві основні проблеми: оцінка надійності складних технічних систем та проектування таких систем із заданим рівнем надійності та безпеки. Однак оцінки надійності лише апаратного компонента таких систем недостатньо, як це було зроблено раніше для суто апаратних електронних систем. Програмне забезпечення стало важливою частиною пристроїв та систем. Напевно, немає іншого створеного людиною матеріалу, який би був всепоширенішим, ніж програмне забезпечення, у нашому сучасному суспільстві [1]. Отже, потреба в оцінці та аналізі надійності програмного забезпечення сьогодні стає все більш актуальною.

З іншого боку, вартість виправлення помилок швидко зростає на пізніх стадіях життєвого циклу програмного забезпечення, що викликає інтерес до моделей передбачення надійності програмного забезпечення, які мають високу прогнозовану силу. Одним із підходів до побудови таких моделей є використання технологій штучного інтелекту. Методи та моделі ШІ стають перспективним інструментом прогнозування надійності програмного забезпечення протягом останнього десятиліття. Таким чином, ця стаття присвячена пошуку закономірностей та прогнозуванню дефектів програмних модулів за допомогою техніки машинного навчання.

#### Аналіз літературних джерел

Моделі надійності програмного забезпечення можна розділити на два широкі класи - детерміновані (статичні) та ймовірнісні (динамічні) [2]. Ймовірнісні моделі представляють випадки виникнення дефектів та виправлення помилок як випадкові події. Детерміновані моделі використовують результати аналізу вихідного коду програми як вхідні дані і не включають жодних випадкових подій або значень. Детермінований клас включає модель Голстеда [3], модель цикломатичної складності Мак-Кейба [4] та метричну модель складності [5].

Загалом, ці моделі представляють кількісний підхід до вимірювання комп'ютерного програмного забезпечення. Модель Холстеда використовується для оцінки кількості дефектів програми [2], тоді як модель цикломатичної складності Маккейба використовується для визначення верхньої межі тестів програми [6]. Обидві моделі мають статичний характер, тобто вважають процеси в програмній системі незмінними з плином часу, а її надійність є виключно функцією програмних показників. Суттєво іншим підходом до вирішення проблем сучасного етапу теорії надійності програмного забезпечення є теорія динаміки програмних систем [7]. Теорія динаміки програмних систем відрізняється від існуючої теорії надійності програмного забезпечення тим, що вона базується на загальній теорії системної динаміки, а не на теорії ймовірностей, і розглядає збої програмного забезпечення не як випадковий процес, а як результат впливу визначених потоків дефектів. Отже, ця теорія також породжує детерміновані моделі надійності програмного забезпечення, які, проте, не є статичними, на відміну від моделей Халстеда та Маккейба.

Зростання емпіричних технологій програмної інженерії призвів до збільшення інтересів до алгоритмів прогнозування помилок [8]. Ці алгоритми передбачають області програмних проєктів, які, ймовірно, схильні до помилок: області, де спостерігається висока частота появи помилок. Загальний підхід статистичний: алгоритми прогнозування помилок базуються на аспектах того, як розроблявся код, та різних метриках, які демонструє код, а не на традиційних підходах статичного або динамічного аналізу. Однією із загальноприйнятих стратегій є використання метрик коду [9], тоді як інша зосереджена на вилученні функцій з історії змін вихідного файлу, спираючись на інтуїцію, згідно з якою помилки скупчуються навколо складних фрагментів коду [10, 11]. Іншим важливим фактором, який впливає на надійність програмного забезпечення, є механізм проєктування, який має значний вплив на загальну якість програмного забезпечення. Для забезпечення кращої надійності необхідна добре розроблена внутрішня структура програмного забезпечення. У літературі описано лише кілька підходів, які стосуються цього питання. Один з них [12] вивчає вплив проєктних метрик на один із зовнішніх факторів якості, надійність програмного забезпечення, використовуючи багатофакторний регресійний аналіз.

Чому деякі програмні модулі більш схильні до дефектів, ніж інші? Відповідь на це питання допомагає зрозуміти природу дефектів і допоможе уникнути дефектів у майбутньому. На жаль, зараз немає універсальної відповіді на це питання. Однак багато досліджень [8, 13], включаючи галузеві, були присвячені висвітленню цих властивостей проєкту [14]. Комплексне порівняльне дослідження Лессмана використовували класифікатори машинного навчання для прогнозування несправних модулів програмного забезпечення [15, 16]. У роботі [9] було обговорено деякі властивості проєкту (інваріанти в їх історії), і було показано, що ймовірність дефекту якогось модуля залежить від його історії.

#### Розроблення алгоритму прогнозування алгоритму прогнозування дефектів програмного забезпечення

У цьому дослідженні було використано загальнодоступний набір даних зі сховища програмної інженерії PROMISE [17]. Вибрано набір даних JM1 щодо прогнозування дефектів програмного забезпечення. Джерелом цього набору даних є NASA та NASA Metrics Data Program. JM1 є прогнозованою системою реального часу, дані надходять від McCabe, а Halstead має екстрактори вихідного коду. Виміри Маккейба та Халстеда базуються на "модулі", де "модуль" є найменшою одиницею функціональності. Набір даних містить 10 885 записів (модулів), а також 21 метрику коду, що використовуються як функції. Серед усіх модулів, перелічених у наборі даних, 2 107 помічені як дефекти. Отже, ми вирішили проблему дисбалансу даних шляхом вибірки, випадковим чином вибравши підмножину, що містить 50/50 записів із повідомленнями про дефекти та без них. Всі розрахунки були зроблені за допомогою RStudio. Рандомізовані 90% набору даних використовувались як навчальний набір, а решта 10% - як набір для перевірки.

Метою є побудова моделі для класифікації дефектів програмного забезпечення. Завдання класифікації полягає у віднесенні об'єкта до одного із заздалегідь визначених класів на основі його формалізованих ознак. Кожен із класифікованих об'єктів представлений у вигляді N-мірного вектора, кожен вимір якого відповідає одній із особливостей об'єкта. Бінарна модель класифікації повинна оцінювати вплив параметра або групи параметрів на класифікацію дефектів. Процес аналізу складається з двох фаз. На першій фазі розглядали всі параметри. На другій фазі вивчався вплив кожного параметра.

Спочатку ми спробували використовувати традиційні методи класифікації, а саме: k найближчих сусідів (kNN), дерево рішень, логістичну регресію, SVM та класифікатор Наївого Байєса. Точність застосовуваних на цій фазі методів класифікації коливається від 77% для kNN до 81% у випадку логістичної регресії та SVM. Як бачимо, результати класифікації не дуже хороші для будь-якого із використовуваних методів.

Порівняння різних алгоритмів класифікації подано нижче:

Accuracy							
	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
CART	0.6753247	0.7272727	0.7532468	0.7469697	0.7662338	0.7922078	0
LDA	0.1666667	0.5178571	0.6666667	0.6465079	0.8000000	0.8571429	0
SVM	0.5000000	0.6666667	0.7142857	0.7326984	0.8000000	1.0000000	0
KNN	0.3333333	0.6166667	0.6666667	0.6634921	0.7142857	0.8333333	0
RF	0.3333333	0.6666667	0.7142857	0.7330159	0.8333333	1.0000000	0

Kapra							
	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
CART	0.2762566	0.3620724	0.4241878	0.41518671	0.4861107	0.5250000	0
LDA	-0.6666667	-0.2264706	0.0000000	0.05297494	0.2500000	0.6666667	0
SVM	-0.2857143	0.0000000	0.0000000	0.08151261	0.0000000	1.0000000	0
KNN	-0.5000000	-0.1875000	0.0000000	-0.08095238	0.0000000	0.0000000	0
RF	-0.5000000	0.0000000	0.0000000	0.20330322	0.5714286	1.0000000	0

Тому на наступному кроці ми намагалися зменшити розмір завдання, знаходячи найважливіші особливості. Так, з використанням алгоритму Boruta обрано 7 характеристик (Рис. 1).

Далі до обраних характеристик ми використовували ансамбль моделей - карту Кохонена та ієрархічну кластеризацію. Теплова карта показує вагу атрибутів та їх групування, а також кластеризує дані. Карта Кохонена використовує навчання без нагляду, а навчальний набір складається лише зі значень вхідних змінних. Карта Кохонена навчається шляхом послідовного наближення. Починаючи з випадково вибраного початкового розташування центрів, алгоритм поступово вдосконалюється для збору даних про навчання. Основний ітераційний алгоритм Кохонена послідовно проходить через ряд епох; для кожної епохи обробляється один навчальний приклад. Вхідні сигнали подаються послідовно в мережу. Бажані вихідні сигнали не визначаються. Після обробки достатньої кількості вхідних векторів синаптичні ваги мережі визначаються кластерами. Крім того, шкали організовані таким чином, що топологічно близькі вузли чутливі до подібних вхідних сигналів.

Для реалізації запропонованого алгоритму необхідно визначити ступінь сусідства нейронів (нейрон переможця). Для цього ми використовували ієрархічну кластеризацію. Були виявлені деякі відхилення, і відповідні зразки були виключені з подальшого аналізу. Відстані  $d_j$  від вхідного сигналу до кожного нейрона  $j$  визначаються як:

$$d_j = \text{SUM}(x_i(t) \cdot w_{ij}(t))^2,$$

тут SUM - це сума для всіх  $j$ ;  $x_i(t)$  -  $i$ -й елемент вхідного сигналу в момент часу  $t$ ,  $w_{ij}(t)$  - вага зв'язку від  $i$ -го елемента вхідного сигналу до  $j$ -го нейрона в момент часу  $t$ .

Відкориговані значення ваги для нейрона  $j^*$  та всіх нейронів у його найближчому сусіді розраховуються як:

$$w_{ij}(t+1) = w_{ij}(t) + r(t)(x_i(t) - w_{ij}(t)),$$

де  $r(t)$  - швидкість навчання, яка з часом зменшується (позитивна, менше одиниці).

Алгоритм використовує поступово зменшується темп навчання для точної настройки нової епохи. В результаті центр встановлюється в такому положенні, що задовільно кластеризує приклади, для яких даний нейрон є переможцем.

Властивість топологічного впорядкування досягається в алгоритмі за допомогою концепції сусідства. Околиці - це ряд нейронів, що оточують нейрон-переможця. Відповідно до швидкості тренувань, розмір району поступово зменшується, так що спочатку він охоплює досить велику кількість нейронів (можливо, цілу карту), на найновіших етапах він складається лише з виграшу. В алгоритмі навчання корекція застосовується не тільки до нейрону-виграшу, але й до всіх нейронів з його поточного сусідства.

Variable Importance

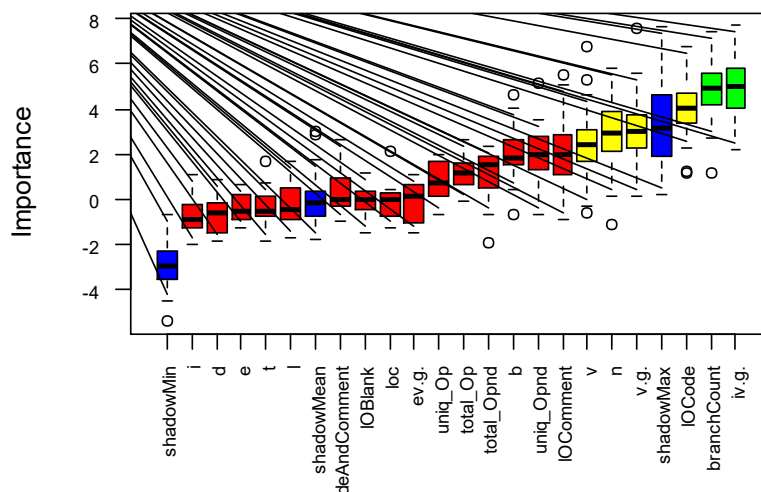


Рис 1. Обрані характеристики набору даних

Сусідство є результатом агломеративної ієрархічної кластеризації. Ідея агломеративних ієрархічних методів полягає в наступному. Спочатку кожен об'єкт розглядається як окремий кластер. Далі ідентифікують два найближчі кластери  $Q_i$  та  $Q_j$  та об'єднують їх в один кластер  $Q_i + j$ . Процес злиття триває, поки всі об'єкти не утворюють єдиний кластер. Під час злиття слід обчислити відстань від нового кластеру  $Q_i + j$  до всіх інших кластерів. Ця відстань обчислюється за допомогою Лейнса-Вільямса:

$$d(Q_{s+j}, Q_m) = \alpha_i \cdot d(Q_i, Q_m) + \alpha_j \cdot d(Q_j, Q_m) + \beta \cdot d(Q_i, Q_j) + \gamma \cdot |d(Q_i, Q_m) - d(Q_j, Q_m)|,$$

де  $d(\bullet, \bullet)$  - міра відстані;  $Q_m$  ( $m \neq i, j$ ) - поточний кластер (сусідство для SOM);  $\alpha_i, \alpha_j, \beta, \gamma$  деякі числові параметри.

У результаті цієї зміни в районах, початкові досить великі ділянки мережі мігрують до тематичних досліджень. Мережа утворює грубу структуру топологічного порядку, в якій подібні приклади активують групи нейронів, які знаходяться близько на топологічній карті. З кожною новою епохою швидкість тренувань та розміри районів зменшуються, отже, все менше знаходяться відмінності всередині розділів карти, що в кінцевому підсумку призводить до більш тонкої настройки кожного нейрона. Часто навчання навмисно розбивається на дві фази: більш коротку, з вищою швидкістю навчання та більшими кварталами, і довшу з повільною швидкістю навчання та нульовими або майже нульовими сусідами.

У Таблиці 1 представлені результати класифікації, отримані з використанням описаного алгоритму.

### Висновки

У цій роботі було розроблено вдосконалений алгоритм прогнозування дефектів програмного забезпечення, заснований на поєднанні карти Кохонена та ієрархічної кластеризації. Алгоритм використовує поступово зменшується темп навчання для точної настройки нової епохи. В результаті центр встановлюється в такому положенні, що задовільно кластеризує приклади, для яких даний нейрон є переможцем. Властивість топологічного впорядкування досягається в алгоритмі за допомогою концепції сусідства. Сусідство є результатом агломеративної ієрархічної кластеризації. Запропонований алгоритм показує вищу точність, ніж інші алгоритми класифікації. Попередня обробка даних дозволяє нам підвищити якість аналізу, розділивши всі дані на два кластери.

Стаття доводить висновок Хаттона [18], що кількість рядків програмного коду, LOC - у нашому випадку вона відповідає PC1, є такою ж хорошою, як і будь-яка інша метрика. Однак наше дослідження показує, що показник LOC не може бути використаний як виключно ознака для класифікації програмного забезпечення з точки зору його схильності до дефектів. З іншого боку, наші результати підтверджують пропозицію Лі та ін. [15] спробувати різні виміри при виборі найкращого класифікатора, а саме. використовувати ансамбль метрик - як видно з результатів роботи, щонайменше 13 метрик коду необхідні для отримання близько 99% достовірності.

Подальші дослідження включатимуть зменшення набору функцій, щоб виявити лише найбільш актуальні. Таке скорочення слід проводити з урахуванням можливих відмінностей між мовами програмування, методологією розробки та іншими процесами програмної інженерії..

### Література

1. Michael Lyu. Software Reliability Engineering: A Roadmap. In Future of Software Engineering (FoSE'07), pages 153–170, Minneapolis, MN, USA, May 2007. IEEE
2. Hoang Pham. System software reliability. Springer-Verlag London Limited, 2006.
3. Maurice H. Halstead. Elements of Software Science. Elsevier North-Holland Publishing, New York, 1977
4. T.J. McCabe. A Complexity Measure. IEEE transactions on Software Engineering, SE-2(4): 308–320, 1976
5. Ning Chen, Steven C. H. Hoi, Xiaokui Xiao. Software process evaluation: a machine learning framework with application to defect management process. Empirical Software Engineering, 19(6): 1531–1564, 2014
6. Cobra Rahmani, Azad H. Azadmanesh. Exploitation of Quantitative Approaches to Software Reliability. Survivable Networked Systems (CIST-9900) Report. University of Nebraska at Omaha, 2008
7. Dmitry Maevsky, Vyacheslav Kharchenko, Maryna Kolisnyk, Elena Maevskaya. Software reliability models and assessment techniques review: Classification issues. In Proceedings of 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2017), pages 894–899 (vol. 2), Bucharest, Romania, September 2017. IEEE
8. Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, E. James Whitehead. Does Bug Prediction Support Human Developers? Findings from a Google Case Study. In Proceedings of the 35th International Conference on Software Engineering ICSE'13, pages 372–381, San Francisco, CA, USA, September

2013. IEEE.

9. Nachiappan Nagappan, Thomas Ball, Andreas Zeller. Mining metrics to predict component fail-ures. In Proceedings of the 28th International Conference on Software Engineering ICSE 06, pages 452, Shanghai, China, May 2006. ACM
10. A. E. Hassan, R. C. Holt. The Top Ten List: Dy-namic Fault Prediction. In 21 IEEE Int. Conf. on Software Maintenance ICSM05, pages 263–272, Budapest, Hungary, September 2005. IEEE
11. Thomas J. Ostrand, Elaine J. Weyuker, Robert M. Bell. Predicting the location and number of faults in large software systems. IEEE Transactions on Software Engineering, 31(4): 340–355, 200
12. R. Selvarani, R. Bharathi. Early Detection of Soft-ware Reliability: A Design Analysis. In book: Strategic Engineering for Cloud Computing and Big Data Analytics, pp. 83–99. Springer, 2017
13. Foyzur Rahman, Daryl Posnett, Abram Hindle, Earl Barr, Premkumar Devanbu. BugCache for In-spections: Hit or Miss? In Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Soft-ware Engineering Conference (ESEC/FSE'11), Szeged, Hungary, September 2011. ACM.
14. Thomas Zimmermann, Nachiappan Nagappan, Andreas Zeller. Predicting Bugs from History. In Tom Mens, Serge Demeyer (Ed.), Software Evolu-tion, Chapter 4, pages 69–88, Springer, March 2008
15. Libo Li, Stefan Lessmann, Bart Baesens. Evaluat-ing software defect prediction performance: an updated benchmarking study. arXiv preprint arXiv:1901.01726 [cs.SE], 2019
16. Stefan Lessmann, Bart Baesens, Christophe Mues, Swantje Pietsch. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transac-tions on Software Engineering, 34(4): 485–496, 2008
17. Sayyad Shirabad, J. and Menzies, T.J. (2005) The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. Avail-able: <http://promise.site.uottawa.ca/SERepository>
18. Les Hatton. Invited Talk: The Role of Empiricism in Improving the Reliability of Fu-ture Software. In Testing: Academic and Industrial Conference – Practice and Research Techniques (TAICPART), Windsor, UK, August 2008. IEEE

#### References

1. Michael Lyu. Software Reliability Engineering: A Roadmap. In Future of Software Engineering (FoSE'07), pages 153–170, Minneapolis, MN, USA, May 2007. IEEE
2. Hoang Pham. System software reliability. Springer-Verlag London Limited, 2006.
3. Maurice H. Halstead. Elements of Software Sci-ence. Elsevier North-Holland Publishing, New York, 1977
4. T.J. McCabe. A Complexity Measure. IEEE trans-action on Software Engineering, SE-2(4): 308–320, 1976
5. Ning Chen, Steven C. H. Hoi, Xiaokui Xiao. Soft-ware process evaluation: a machine learning framework with application to defect management process. Empirical Software Engineering, 19(6): 1531–1564, 2014
6. Cobra Rahmani, Azad H. Azadmanesh. Exploita-tion of Quantitative Approaches to Software Reli-ability. Survivable Networked Systems (CIST-9900) Report. University of Nebraska at Omaha, 2008
7. Dmitry Maevsky, Vyacheslav Kharchenko, Maryna Kolisnyk, Elena Maevskaya. Software re-liability models and assessment techniques review: Classification issues. In Proceedings of 9th IEEE International Conference on Intelligent Data Ac-quisition and Advanced Computing Systems: Technology and Applications (IDAACS'2017), pages 894–899 (vol. 2), Bucharest, Romania, Sep-tember 2017. IEEE
8. Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, E. James Whitehead. Does Bug Prediction Support Human Developers? Find-ings from a Google Case Study. In Proceedings of the 35th International Conference on Software Engineering ICSE'13, pages 372–381, San Fran-cisco, CA, USA, September 2013. IEEE.
9. Nachiappan Nagappan, Thomas Ball, Andreas Zeller. Mining metrics to predict component fail-ures. In Proceedings of the 28th International Conference on Software Engineering ICSE 06, pages 452, Shanghai, China, May 2006. ACM
10. A. E. Hassan, R. C. Holt. The Top Ten List: Dy-namic Fault Prediction. In 21 IEEE Int. Conf. on Software Maintenance ICSM05, pages 263–272, Budapest, Hungary, September 2005. IEEE
11. Thomas J. Ostrand, Elaine J. Weyuker, Robert M. Bell. Predicting the location and number of faults in large software systems. IEEE Transactions on Software Engineering, 31(4): 340–355, 200
12. R. Selvarani, R. Bharathi. Early Detection of Soft-ware Reliability: A Design Analysis. In book: Stra-tegic Engineering for Cloud Computing and Big Data Analytics, pp. 83–99. Springer, 2017
13. Foyzur Rahman, Daryl Posnett, Abram Hindle, Earl Barr, Premkumar Devanbu. BugCache for In-spections: Hit or Miss? In Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Soft-ware Engineering Conference (ESEC/FSE'11), Szeged, Hungary, September 2011. ACM.
14. Thomas Zimmermann, Nachiappan Nagappan, Andreas Zeller. Predicting Bugs from History. In Tom Mens, Serge Demeyer (Ed.), Software Evolu-tion, Chapter 4, pages 69–88, Springer, March 2008
15. Libo Li, Stefan Lessmann, Bart Baesens. Evaluat-ing software defect prediction performance: an updated benchmarking study. arXiv preprint arXiv:1901.01726 [cs.SE], 2019
16. Stefan Lessmann, Bart Baesens, Christophe Mues, Swantje Pietsch. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transac-tions on Software Engineering, 34(4): 485–496, 2008
17. Sayyad Shirabad, J. and Menzies, T.J. (2005) The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. Avail-able: <http://promise.site.uottawa.ca/SERepository>
18. Les Hatton. Invited Talk: The Role of Empiricism in Improving the Reliability of Fu-ture Software. In Testing: Academic and Industrial Conference – Practice and Research Techniques (TAICPART), Windsor, UK, August 2008. IEEE

Рецензія/Peer review : 09.01.2021 р.

Надрукована/Printed : 10.03.2021 р.