НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кваліфікаційна наукова
праця на правах рукопису

СИМЕЦЬ ІВАН ІГОРОВИЧ

УДК 004.052

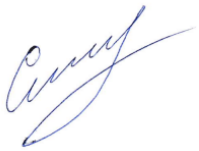
ДИСЕРТАЦІЯ

МОДЕЛІ І МЕТОДИ ПРОГНОЗУВАННЯ ТА АНАЛІЗУ НАДІЙНОСТІ ТЕХНІЧНИХ СИСТЕМ З УРАХУВАННЯМ ПРОЦЕСУ РОЗРОБКИ ПЗ

121 – «Інженерія програмного забезпечення»
12 – «Інформаційні технології»

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.



І.І. Симець

Науковий керівник:
Яковина Віталій Степанович
доктор технічних наук, професор

Львів – 2022

АНОТАЦІЯ

Симець І. І. **Моделі і методи прогнозування та аналізу надійності технічних систем з урахуванням процесу розробки ПЗ.** – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття ступеня доктора філософії за спеціальністю 121 «Інженерія програмного забезпечення» (12 – «Інформаційні технології»). – Національний університет «Львівська політехніка», Львів, 2022.

У дисертаційній роботі розв'язано актуальну науково-прикладну задачу у галузі інженерії програмного забезпечення – підвищення достовірності прогнозування та оцінювання показників надійності програмно-апаратних систем шляхом удосконалення відповідних моделей надійності та створення методів і засобів автоматизації їх побудови.

Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку літературних джерел та додатків.

В першому розділі розглянуто основні поняття, критерії та показники надійності програмних систем. Представлено детальний опис моделей надійності ПЗ та їх особливостей. Аналіз показав, що зростання складності програмно-апаратних систем вимагає розроблення і вдосконалення моделей надійності з підвищенням ступеня їх адекватності. Практичне використання певних методів і моделей є дещо ускладнене і пов’язане з розв’язанням задач великої розмірності та складними розрахунками, які унеможливлюють виконання даних робіт вручну, оскільки зростає імовірність помилки при побудові та аналізі таких моделей, відповідно цей процес потребує спеціальних методів автоматизації.

Розглянуто використання моделі на основі ланцюгів Маркова вищого порядку, як засобу для підвищення достовірності оцінки показників надійності ПЗ, оскільки дана модель враховує передісторію виконання попередніх модулів програмної системи.

Також, розглянуто методи прогнозування дефектності програмного забезпечення на основі метрик коду (під дефектністю розуміємо наявність чи відсутність програмних дефектів у відповідному модулі ПЗ). Ці методи

прогнозування дефектності програмного забезпечення дозволяють розробникам виявити дефекти на основі наявних програмних показників і таким чином покращити якість програмного забезпечення. Як показав аналіз літератури, наявні підходи мають певні недоліки: недостатньо високу точність прогнозу; немає єдиної думки щодо впливу метрик коду ПЗ на показники його якості, і, зокрема, надійності; питання визначення набору метрик програмного коду, що найбільше корелюють з показниками його якості.

У другому розділі описано розроблені методи для автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами та для визначення функції працездатності для Марковських моделей надійності ПЗ. Використання розроблених методів дозволяє автоматизувати процес використання відповідних моделей надійності, зменшує вплив людського фактора, і, відповідно, знижує ймовірність внесення помилки при використанні даних методів і моделей та підвищує достовірність оцінки надійності.

Для демонстрації практичного використання та перевірки розробленого методу подання Марковського процесу вищого порядку його було реалізовано та апробовано на прикладі оцінки надійності програмного забезпечення польотів наносупутників CubeSat. Дане дослідження спрямоване на визначення того, як зміниться інтенсивність відмови програмної системи під час оцінки за допомогою традиційних ланцюгів Маркова безперервного часу, а також ланцюгів Маркова другого і третього порядку. Значення розраховані за допомогою моделей другого та третього порядку, не є сумою чи усередненням кривих першого порядку. За результатами дослідження стало очевидно, що ланцюги Маркова вищого порядку дають змогу значно підвищити точність оцінки надійності складних програмних систем як за рівнем відмов, так і за критерієм залежності від часу.

У розділі, також, описано визначені комбінаторні формули, які дозволяють визначити максимальну і мінімальну кількість станів системи із загального простору станів, а також окремі формули для визначення максимальної і мінімальної кількості працездатних станів і станів простою із загальної множини

для системи із *n* елементів і *r* відновлень. Дані формули дозволяють швидко оцінити можливий простір станів системи із відомими значеннями n і r без використання спеціальних технічних засобів.

Третій розділ спрямовано на удосконалення моделей прогнозування дефектності ПЗ шляхом використання методів машинного навчання для вибору метрик, які найбільше впливають на дефектність модулів ПЗ, та розроблення методу класифікації модулів ПЗ за дефектністю на основі використання стекового ансамблю нейронних мереж. Усі дослідження було виконано за допомогою мови програмування Python, бібліотеки Scikit-learn і нейромережної бібліотеки Keras. На основі результатів дослідження побудовано модель дефектності ПЗ з використанням обмеженої множини метрик коду, отриманих як найважливіші метрики (кількість рядків коду за МакКейбом (loc), загальна кількість операндів і операторів (N) , функціональність модуля (I), обсяг при мінімальному виконанні (V), зусилля для написання модуля (E), цикломатична складність за МакКейбом (v(g)), кількість гілок в репозиторії, в яких змінювався модуль (branchCount)) та регресійного методу. Розроблено метод класифікації модулів ПЗ за дефектністю на основі метрик коду з використанням стекового ансамблю нейронної мережі на основі радіально-базисних функцій, рекурентної нейронної мережі та мережі довгої короткочасної пам’яті.

У четвертому розділі для автоматизованого використання, тестування і верифікації розроблених методів було спроєктовано та розроблено відповідне програмне забезпечення. Програмне забезпечення для автоматизації подання Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами реалізовано за допомогою бібліотеки ReactJS (для візуалізації графа було використано бібліотеку react-d3-graph) і мови програмування JavaScript у вигляді Web застосунку. Дане програмне забезпечення дозволяє ввести інформацію про вхідний граф першого порядку і на його основі побудувати граф вищого порядку, використовуючи різні значення змінної, що відповідають порядку процесу. Програмний засіб для розрахунку надійнісних характеристик складних технічних систем на основі функції

працездатності дозволяє візуально представити функціонування і взаємодію модулів системи та на їх основі визначити функцію працездатності системи, побудувати граф станів і переходів, визначити та розв’язати систему диференціальних рівнянь і визначити певні показники надійності. Якщо аналіз проводиться для відновлюваної системи відбувається побудова функції готовності і функції простою, а для невідновлювальної системи визначається ймовірність безвідмовної роботи протягом часу *t.* Тестування розробленого ПЗ продемонструвало, що дане ПЗ дає достатньо високий рівень швидкодії при моделюванні.

Ключові слова: надійність ПЗ, дефект, моделі надійності, прогнозування дефектів, машинне навчання, нейронні мережі, умова працездатності, граф станів і переходів, ланцюги Маркова вищого порядку.

ABSTRACT

Symets I.I. **Models and methods of prediction and analysis of technical systems reliability considering the software development process.** – Qualifying scientific work on the rights of the manuscript.

Thesis paper for achievement of the scientific degree Doctor of Philosophy in the specialty 121 «Software Engineering» (12 – «Information technologies»). – Lviv Polytechnic National University, Lviv, 2022.

The thesis paper solved the relevant scientific and applied task in the sphere of software engineering - increase prediction reliability and estimation of reliability indicators of software and hardware systems by improving related models of reliability and creating methods and tools of automation their producing.

The thesis paper consists of introduction, four chapters, conclusions, list of references and appendixes.

The first chapter: The software systems' basic concepts, criteria, and reliability indicators are examined in the first chapter. A detailed description of software reliability models and their features is presented. The analysis showed that the growing complexity of software and hardware systems requires developing and improving reliability models with an increasing degree of their sufficiency. The practical use of certain methods and models is somewhat complicated and is related to large-scale problems solving and complex calculations that make it impossible to perform these tasks manually since the probability of error in producing and analyzing such models increases, so this process requires special automation methods.

Using the model based on high-order Markov chains was considered a means to increase the authenticity of the indicators evaluation reliability software, as this model considers the background of previous software modules.

Also, methods for predicting software defects based on code metrics were considered (defect means the presence or absence of software defects in the proper software module). These predicting methods of software defects allow developers to identify defects based on available software indicators and thus improve the quality of software. As the literature analysis has shown, the available approaches have certain

imperfections: insufficiently high prediction accuracy; there is no consensus on the impact of software code metrics on its quality indicators, and in particular, reliability; the question of defining the set of metrics of program code that most correlate with indicators of its quality.

The second chapter described the developed methods for automation the representation of a higher-order Markov process by an equivalent first-order process with additional virtual states and determining the working function (condition) for Markov software reliability models. The use of developed methods allows to automate the process of using appropriate reliability models, reduces the impact of the human factor, and, accordingly, reduces the probability of errors entering in the use of these methods and models and increases the authenticity of reliability evaluation.

The CubeSat nanosatellite's flights on the software reliability example evaluation were realized and tested to demonstrate the practical use and verification of the developed method of presenting a higher-order Markov process. This study aims to determine how the failure rate of the software system will change during the evaluation using traditional Markov chains of continuous-time, as well as Markov chains of the second and third-order. Values are calculated using second-, and third-order models and are not the sum or average of first-order curves. As a result of the study, it was demonstrated that higher-order Markov chains could significantly increase the accuracy reliability evaluation of complex software systems both in failure rate and time dependence.

The chapter also describes certain combinatorial formulas that allow to determine the maximum and the minimum number of system states from the total state space and individual formulas for determining the maximum and the minimum number of operational states and idle states from the full set for a system of n elements and r renewal. These formulas allow you to quickly estimate the possible state space of the system with known values of n and r without the use of special technical means.

The third chapter aims to improve software defect prediction models by using machine learning methods to select the metrics that most affect software modules' defects and develop a method of classifying software modules by defect based on the

use of a stacking ensemble of neural networks. All studies were performed using the Python programming language, the Scikit-learn library, and the Keras neural network library. Based on the results of the study, a software defect model was produced using a set of code metrics obtained as the most important features (number of code's lines according to McCabe (loc), the total number of operands and operators (N), module functionality (I), minimum execution volume (V), effort to write a module (E), McCabe cyclomatic complexity (v (g)), number of branches in the repository in which the module changed (branchCount)) and regression method. A classifying method of software modules by defects based on code metrics using a neural network stack ensemble based on a radial basis function, a recurrent neural network, and a long short-term memory network has been developed.

In the fourth chapter, appropriate software was designed and developed for the automated use, testing, and verification of the developed methods. Software to automate the representation of the higher-order Markov process as an equivalent first-order process with additional virtual states was implemented using the ReactJS library (the react-d3-graph library was used to visualize the graph) and the JavaScript programming language as a Web application. This software allows to enter information about the input graph of the first order and, on its basis, build a graph of higher-order, using different values of the variable corresponding to the order of the process. Software for calculating the reliability characters of complex technical systems based on the efficiency functions allows to represent the operation and interaction of system modules and on their basis to determine the function of the system, build a graph of states and transitions, define and solve a system of differential equations and determine certain reliability indicators - if the analysis is performed for the recoverable system, the readiness and idle functions are constructed, and for the non-recoverable system, the probability of infallible operation during time t is determined. Testing of the developed software has shown that the developed software gives a high enough level of response speed in modeling.

Keywords: software reliability, defect, reliability models, defect prediction, machine learning, neural networks, efficiency condition, state and transition diagram, high-order Markov chains.

СПИСОК ПРАЦЬ ОПУБЛІКОВАНИХ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковано основні наукові результати дисертації

1. Yakovyna, V. S., Seniv, M. M., Symets, I. I., & Sambir, N. B. (2020). Algorithms and software suite for reliability assessment of complex technical systems. Radio Electronics, Computer Science, Control, (4), 163–177. https://doi.org /10.15588/1607-3274-2020-4-16.
2. Yakovyna, V. S., Seniv, M. M., Lytvyn, V. V., & Symets I. I. (2019). Програмний модуль розв'язування систем диференціальних рівнянь Колмогорова-Чепмена для автоматизації надійнісного проектування. Науковий вісник НЛТУ України, 29(5), 141-146. https://doi.org/ 10.15421/40290528.
3. Яковина, В. С., Сенів, М. М., & Симець, І. І. (2019). Засоби автоматизованого формулювання умов працездатності складних технічних систем. Науковий вісник НЛТУ України, 29(9), 136-141. https://doi.org/ 10.36930/40290924.
4. Vitaliy Yakovyna, Ivan Symets. A method of high-order Markov chain representation through an equivalent first-order chain for software reliability assessment // Комп'ютерні системи та інформаційні технології. – 2021. – № 3. – С. 66–73. https://doi.org/10.31891/CSIT-2021-5-9.
5. Vitaliy Yakovyna, Ivan Symets. Towards a software defect proneness model: feature selection // Прикладні аспекти інформаційних технологій. – 2021. – Vol. 4, № 4. – P. 354–365. https://doi.org/10.15276/aait.
6. Яковина, В. С., & Симець, І. І. (2021). Прогнозування дефектів програмного забезпечення ансамблем нейронних мереж. Науковий вісник НЛТУ України, 31(6), 104-111. https://doi.org/10.36930/40310616.
7. Yuriy Bobalo, Maksym Seniv, Vitaliy Yakovyna, Ivan Symets Method of Reliability Block Diagram Visualization and Automated Construction of Technical System Operability Condition // Advances in Intelligent Systems and Computing III, vol 871. Springer, Cham. P. 599-610. https://doi.org/ 10.1007/978-3-030-01069-0_43.
8. Yakovyna, Vitaliy, Ivan Symets. 2021. "Reliability Assessment of CubeSat Nanosatellites Flight Software by High-Order Markov Chains." Procedia Computer Science 192: 447–56. https://doi.org/10.1016/j.procs.2021.08.046.

Праці, які засвідчують апробацію матеріалів дисертації

1. Yuriy Bobalo, Maksym Seniv, Ivan Symets Algorithms of automated formulation of the operability condition of complex technical systems // Perspective technologies and methods in MEMS design (MEMSTECH'2018) : pros. of XIV-th Intern. Conf., 18-22 april 2018, Lviv - Polyana, Ukraine. – P. 220-224.
2. Maksym Seniv, Vitaliy Yakovyna, Ivan Symets Software for visualization of reliability block diagram and automated formulation of operability conditions of technical systems// Perspective technologies and methods in MEMS design (MEMSTECH'2018) : pros. of XIV-th Intern. Conf., 18-22 april 2018, Lviv - Polyana, Ukraine. – P. 191-195.
3. Yuriy Bobalo, Vitaliy Yakovyna, Maksym Seniv, Ivan Symets. Technique of automated construction of states and transitions graph for the analysis of technical systems reliability. // Proceedings of the 13th International scientific and technnical conference CSIT-2018, 11-14 September 2018. – Lviv, Ukraine 2018. – P. 314.
4. Vitaliy Yakovyna, Maksym Seniv, Ivan Symets. Techniques of Automated Processing of Kolmogorov – Chapman Differential Equation System for Reliability Analysis of Technical Systems. // Proceedings of the 15th Intern. Conf. on The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM'2019, 26 February – 2 March, 2019. – Lviv–Polyana, Ukraine 2019.
5. Yakovyna, V., Seniv, M., Symets, I. The Relation between Software Development Methodologies and Factors Affecting Software Reliability // Proceedings of IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT), CSIT 2020, 23-26 Sept. 2020. - Zbarazh, Ukraine. - 377 - 381; https://ieeexplore.ieee.org/document/9321937

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 14
ВСТУП 15
РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМАТИКОЮ РОБОТИ . 21
1.1. Надійність програмних систем: основні поняття і характеристики 21
1.2. Аналіз і класифікація моделей надійності ПЗ 26
1.3. Аналіз надійності програмних систем із використанням моделі на основі ланцюгів Маркова вищого порядку 31
1.4. Методи прогнозування дефектності програмного забезпечення на основі метрик коду 34
1.5. Висновки до розділу 1 42
РОЗДІЛ 2. МЕТОДИ АВТОМАТИЗАЦІЇ ДЛЯ МАРКОВСЬКИХ МОДЕЛЕЙ АНАЛІЗУ НАДІЙНОСТІ ПЗ 44
2.1. Метод автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами 44
2.2. Оцінка надійності програмного забезпечення польотів наносупутників CubeSat із використанням методу автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку 49
2.3. Метод автоматизованого визначення функції працездатності для Марковських моделей надійності ПЗ 59
2.4. Визначення простору станів для графа станів і переходів із скінченною кількістю відновлень 68
2.5. Висновки до розділу 2 72
РОЗДІЛ 3. МОДЕЛІ ТА МЕТОДИ ПРОГНОЗУВАННЯ ДЕФЕКТНОСТІ ПЗ НА ОСНОВІ МАШИННОГО НАВЧАННЯ 74
3.1. Об’єднаний набір даних із сховища PROMISE Software Engineering 75

3.2. Автоматизований вибір метрик коду, що найбільше впливають на дефектність ПЗ, засобами машинного навчання 78

3.3. Регресійна модель дефектності ПЗ з редукованою множиною ознак 87

3.4. Метод класифікації модулів ПЗ за дефектністю на основі стекового ансамблю нейронних мереж 94

3.5. Висновки до розділу 3 106

РОЗДІЛ 4. ЗАСОБИ АВТОМАТИЗОВАНОГО АНАЛІЗУ НАДІЙНОСТІ СКЛАДНИХ ТЕХНІЧНИХ СИСТЕМ 109

4.1. Програмне забезпечення для автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами 109

4.2. Процес визначення показників надійності із використанням функції працездатності та графа станів і переходів 118

4.3. Програмний засіб для розрахунку надійнісних характеристик складних технічних систем на основі функції працездатності 122

4.4. Аналіз роботи та тестування засобу для розрахунку надійнісних характеристик складних технічних систем на основі функції працездатності 130

4.5. Висновки до розділу 4 136

ОСНОВНІ РЕЗУЛЬТАТИ ТА ВИСНОВКИ 139

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 142

ДОДАТКИ 155

Додаток А. Приклад роботи методу визначення функції працездатності 156

Додаток Б. Приклад роботи методу автоматизації подання процесу маркова вищого порядку еквівалентним процесом першого порядку 160

Додаток В. Опис метрик коду із набору даних PROMISE 164

Додаток Г. Програмний код пз для подання процесу маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами 166

Додаток Д. Код програмної реалізації методу визначення функції працездадності 178

Додаток Е. Список публікацій за темою дисертації 183

Додаток Ж. Акти про впровадження та дослідне випробовування результатів дисертації 185

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ГСП – граф станів і переходів

ЖЦ – життєвий цикл

ЛМВП (англ. High-order Markov chain, HOMC) – ланцюг Маркова вищого порядку

НМ – нейронна мережа

ООП – об'єктно-орієнтоване програмування

ПАС – програмно-апаратна система

ПЗ – програмне забезпечення

ПС – програмна система

LOC (англ. Lines of code) – кількість рядків програмного коду

UML (англ. Unified Modeling Language) – уніфікована мова моделювання

ВСТУП

Обґрунтування вибору теми дослідження. Застосування сучасних інтелектуальних інформаційних систем охоплює такі відповідальні сфери, як промислове управління, робототехніка, медичні та діагностичні системи, фінанси, екологія, машинобудування, космічна та авіаційна галузь, що безпосередньо впливають на життя та охорону здоров'я людей. Із зростанням відповідальності цих інформаційних та інженерних систем постійно зростає потреба в їх якості та безпеці.

Надзвичайно важливою складовою якості складних технічних систем є їхня надійність, тобто властивість системи виконувати задані функції, зберігаючи в часі значення експлуатаційних показників у заданих межах, що відповідають умовам використання та заданим режимам технічного обслуговування, збереження і транспортування.

Більшість із сучасних технічних систем є програмно-апаратними, багатокомпонентними, і відповідно, складними в проєктуванні та розробці. Ці системи стали складнішими, ніж будь-коли, як за структурою, так і за функціональною поведінкою, зростає складність як технічних систем загалом, так і програмного забезпечення зокрема. Таким чином, постійне зростання відповідальності функцій, які виконують програмне і апаратне забезпечення вимагає вищого ступеня надійності, і відповідно більш достовірного і точного оцінювання показників їх надійності.

Оцінку та прогнозування надійності ПЗ виконують, зазвичай, з використанням відповідних моделей надійності. Сучасний стан розвитку методів і моделей аналізу надійності складних технічних систем характеризується поєднанням аналітичних методів дослідження надійності з обчислювальними можливостями сучасних комп'ютерних засобів. Зростання складності програмно-апаратних систем вимагає розроблення та вдосконалення моделей надійності з підвищеним ступенем їх адекватності. Підвищення ступеня адекватності таких моделей надійності зазвичай супроводжується зростанням їх складності. Практичне використання таких методів і моделей є ускладнене і пов'язане з

розв’язанням задач великої розмірності та складними розрахунками, які практично унеможливлюють виконання даних робіт вручну, оскільки зростає ймовірність помилки при побудові та аналізі таких моделей. Тому практичне використання моделей і методів надійності програмних і програмно-апаратних систем з підвищеним ступенем адекватності потребує створення відповідних методів автоматизації.

Питанням забезпечення та оцінки показників надійності приділяють увагу вже на початкових етапах проєктування і розробки таких систем, де формується їх архітектура, тобто структура та алгоритми функціонування. З метою підвищення якості та надійності програмного забезпечення, також, застосовують методи прогнозування дефектності програмного забезпечення для виявлення потенційних помилок на основі характеристик системи та програмного коду. Методи прогнозування дефектності програмного забезпечення можуть допомогти розробникам виявити дефекти на основі наявних програмних метрик, використовуючи методи аналізу даних, і тим самим покращити якість програмного забезпечення, що в результаті призводить до зниження витрат на розробку програмного забезпечення на етапі розробки та обслуговування.

Тому актуальною є наукова задача підвищення точності прогнозування та оцінювання показників надійності програмно-апаратних систем шляхом удосконалення відповідних моделей надійності та розроблення методів і засобів автоматизації їх побудови.

Зв’язок роботи з науковими програмами, планами, темами. Дисертацію виконано на кафедрі Програмного забезпечення Національного університету «Львівська політехніка». Тема дисертації відповідає науковому напрямку кафедри – програмне та математичне забезпечення автоматизованих систем.

Дисертаційні дослідження виконувалися в межах держбюджетних науково-дослідних робіт:

- «Підвищення ефективності засобів бездротового зв’язку відповідального призначення та процедур моделювання і прогнозування їх характеристик» (номер держреєстрації 0118U000261);

- «Розроблення інформаційної технології оцінювання та прогнозування надійності програмного забезпечення методами машинного навчання» (номер держреєстрації 0121U109527);
- «Розроблення криптозахищеної системи високошвидкісного передавання даних у діапазонах УВЧ і НВЧ з підвищеними завадостійкістю та відмовостійкістю» (номер держреєстрації 0122U000960).

Мета і завдання дослідження. Метою дисертаційного дослідження є підвищення точності прогнозування та аналізу показників надійності програмних систем.

Відповідно до вказаної у роботі мети **потрібно вирішити такі основні завдання:**

- аналіз відомих підходів і моделей для оцінювання та прогнозування показників надійності ПЗ;
- підвищення ступеня адекватності Марковських моделей надійності ПЗ:
    - розроблення методу автоматизованого визначення функції працездатності;
    - створення методу автоматизації аналізу надійності функціонування ПЗ на основі ланцюгів Маркова вищого порядку;
- удосконалення методів та моделей прогнозування дефектності ПЗ засобами машинного навчання на основі метрик коду;
- розроблення програмних засобів для оцінювання та прогнозування показників надійності ПЗ на основі отриманих теоретичних результатів;
- верифікація розробленого математичного та програмного забезпечення.

Об’єктом дослідження дисертаційної роботи є процес аналізу та прогнозування надійності ПЗ.

Предметом дослідження є моделі, методи та алгоритми прогнозування надійності ПЗ та визначення її показників.

Методи дослідження. При проведенні досліджень використано такі методи: методи теорії надійності складних технічних систем; методи теорії графів – для створення методу автоматизації подання процесу Маркова вищого порядку

еквівалентним процесом першого порядку з додатковими віртуальними станами; методи машинного навчання – для визначення набору метрик коду, які найбільше впливають на його дефектність і для створення методу класифікації модулів ПЗ за дефектністю на основі метрик коду із використанням стекового ансамблю нейронних мереж; методи обчислювальної математики – для побудови і розв’язання системи диференціальних рівнянь Колмогорова-Чепмена та визначення показників надійності ПЗ; комбінаторні методи – для виведення формул визначення простору станів для графа станів і переходів; методи об'єктно-орієнтованої парадигми програмування та методи теорії алгоритмів – для розроблення програмного забезпечення.

Наукова новизна отриманих результатів:

1. Вперше розроблено метод автоматизованого визначення функції працездатності, який грунтується на аналізі топології системи і, на відміну від існуючих підходів, дає змогу в автоматизованому режимі визначати складну логічну функцію, що зменшує ймовірність внесення похибок і підвищує точність моделювання надійності;
2. Вперше розроблено метод автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами із використанням якого є змога формувати еквівалентний процес для довільного порядку, не використовуючи розширеної матриці ймовірностей, у зв’язку із чим підвищується точність оцінки надійності складних програмних систем;
3. Отримали подальший розвиток Марковські моделі надійності програмно-апаратних систем, які, на відміну від існуючих, дають змогу визначати максимальну і мінімальну кількість працездатних станів;
4. Отримали подальший розвиток моделі дефектності ПЗ, які відрізняються від існуючих використанням обмеженої кількості метрик коду ПЗ, які найбільше впливають на дефектність, що дає змогу підвищити точність визначення показників надійності ПЗ на ранніх етапах його життєвого циклу;

5. Отримав подальший розвиток метод класифікації модулів ПЗ за дефектністю, який відрізняється стекінговим ансамблюванням нейронної мережі на основі радіально-базисних функцій, рекурентної нейронної мережі та мережі довгої короткочасної пам’яті та дає змогу підвищити точність прогнозування дефектності ПЗ.

Практичне значення одержаних результатів. Практичне значення дисертаційного дослідження полягає у застосуванні розроблених методів і програмного забезпечення, які дають змогу в автоматизованому режимі аналізувати надійність програмних та програмно-апаратних систем і тим самим забезпечити підвищення точності прогнозування та визначення оцінок для показників надійності таких систем. Використання розроблених методів та програмного забезпечення дає можливість зменшити трудоємність надійнісного проектування програмних та програмно-апаратних систем, знизити ймовірність внесення помилки на цьому етапі та не вимагає високої кваліфікації проєктанта. Розроблена модель дефектності та метод класифікації програмних модулів надають рекомендації розробникам щодо підвищення якості програмних модулів на етапах проєктування та кодування програмного забезпечення.

Результати дисертаційної роботи використано: у 3-х держбюджетних науково-дослідних роботах.

Результати роботи впроваджені у навчальний процес кафедри Програмного забезпечення Національного університету «Львівська політехніка» для студентів спеціальності 121 «Інженерія програмного забезпечення» в лекційному курсі та практикумі дисципліни «Теорія надійності програмних систем».

Результати дисертаційного дослідження пройшли дослідницьке випробування на підприємствах ТзОВ «Едвантіс» і ПП «Лінк Ап Студіо».

Особистий внесок здобувача. Усі наукові положення, які є основним змістом дисертаційної роботи, розроблено та обґрунтовано здобувачем особисто. У друкованих працях, опублікованих у співавторстві, здобувачеві належать: метод автоматизованого визначення функції працездатності, який грунтується на попередньому повному аналізі топології системи [102, 103, 107]; метод

автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами [104]; модель дефектності ПЗ, яка використовує обмежену множину метрик коду ПЗ, що найбільше корелюють з його надійністю [105,113]; метод класифікації модулів ПЗ за дефектністю, який відрізняється стекінговим ансамблюванням нейронних мереж [106]; формули визначення простору станів графа станів і переходів [101]; програмні засоби для оцінювання та прогнозування показників надійності ПЗ на основі отриманих теоретичних результатів [108, 109,110, 111, 112].

Апробація матеріалів дисертації. Результати дисертаційного дослідження апробовано на міжнародних науково-практичних конференціях та семінарах:

- XIV Міжнародна конференція Perspective technologies and methods in MEMS design (MEMSTECH 2018), Поляна (Закарпаття), Україна, 2018 р.;
- XIII Міжнародна конференція Computer Science and Information Technologies (CSIT 2018), Львів, Україна, 2018 р.;
- XV Міжнародна конференція The Experience of Designing and Application of CAD Systems (CADSM 2019), Поляна (Закарпаття), Україна, 2019 р.;
- XV Міжнародна конференція Computer Science and Information Technologies (CSIT 2020), Збараж, Україна, 2020 р.;
- XXV Міжнародна конференція Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2021), Szczecin, Poland, 2021 р..

Публікації. Основні положення дисертації опубліковано у 13 наукових працях, з яких: 5 статей у наукових фахових виданнях України, 1 стаття у науковому фаховому виданні України, що входить міжнародної науковометричної бази Web of Science, 2 статті у наукових періодичних виданнях інших держав та 5 праць – у матеріалах і тезах конференцій.

Структура та обсяг роботи. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел (114 найменувань) і 7 додатків. Основний зміст викладено на 127 сторінках друкованого тексту, містить 73 рисунки, 29 таблиць. Загальний обсяг роботи – 188 сторінок.

РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМАТИКОЮ РОБОТИ

В даному розділі виконано аналіз та огляд основних понять, критеріїв та показників надійності технічних систем. Здійснено порівняльний огляд відомих моделей оцінювання та аналізу надійності. Наведено різні класифікації моделей надійності та визначено їхні слабкі та сильні сторони.

Розглянуто використання моделі на основі ланцюгів Маркова вищого порядку, як засобу для підвищення достовірності оцінювання надійності складних програмних систем.

Виконано огляд відомих методів прогнозування дефектності програмного забезпечення на основі метрик коду, як засобу запобігання виникнення і передбачення помилок. Розглянуто поетапно процес прогнозування дефектності, який включає: збір даних і метрик коду про дефектність модулів програмних систем, вилучення потрібних метрик коду для прогнозування, побудову моделі класифікації і оцінку передбачення модулі. Також було розглянуто відомі методи і підходи для прогнозування дефектності на основі машинного навчання і штучних нейронних мереж.

1.1. Надійність програмних систем: основні поняття і характеристики

Надійність програмного забезпечення є важливим фактором вимірювання якості програмного забезпечення і характеризується властивістю технічної системи виконувати визначені завдання, зберігаючи експлуатаційні показники в заданих межах протягом операційного періоду, що відповідають умовам використання та заданим режимам, технічного обслуговування, збереження і транспортування [1]. Ступінь надійності характеризується ймовірністю роботи програмного продукту без відмови протягом певного проміжку часу.

Відповідно до стандарту ISO/IEC 25010:2011 [2] (System and software quality model), якість програмної системи – це ступінь, в якому система задовольняє заявлені та неявні потреби різних зацікавлених сторін і, таким чином, забезпечує цінність. Потреби цих зацікавлених сторін (функціональність, продуктивність,

безпека, ремонтопридатність тощо) є саме тим, що представлено в моделі якості (відповідно до стандарту рис. 1.1), яка класифікує якість продукції на характеристики та підхарактеристики.

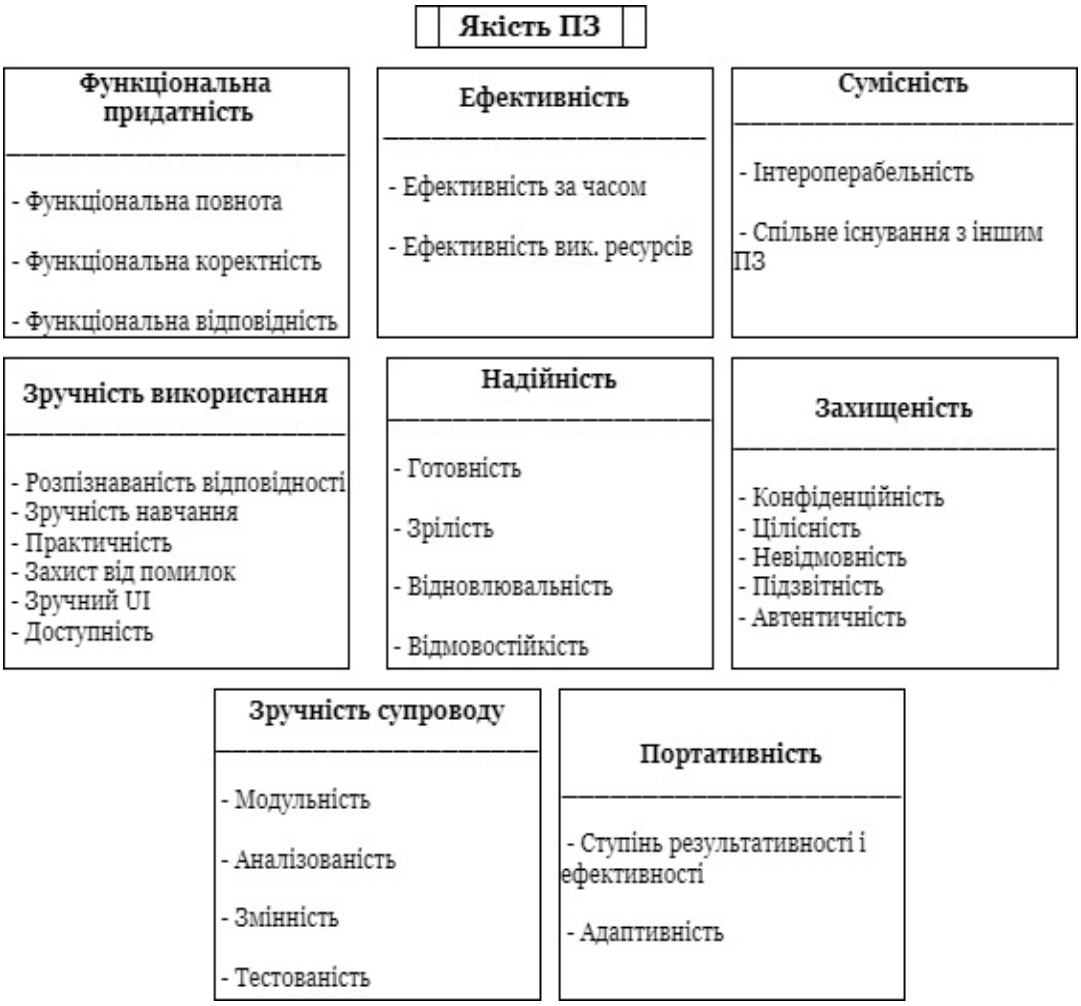


Рис. 1.1. Модель якості ПЗ відповідно до стандарту ISO/IEC 25010:2011

Однією із головних складових якості є надійність програмного забезпечення. Відповідно до стандарту ISO/IEC 25010:2011, надійність – це ступінь, до якого система, продукт або компонент виконують задані функції за певних умов протягом певного проміжку часу. Надійність іноді класифікують відповідно до того «як якість змінюється з часом». Різниця між якістю та надійністю полягає в тому, що якість показує, наскільки добре об’єкт виконує свою належну функцію, тоді як надійність показує, наскільки добре цей об’єкт зберігає свій початковий рівень якості протягом тривалого часу в різних умовах.

Відповідно до моделі якості ПЗ надійність складається із таких показників (рис. 1.2): готовність, зрілість, відмовостійкість, відновлюваність.

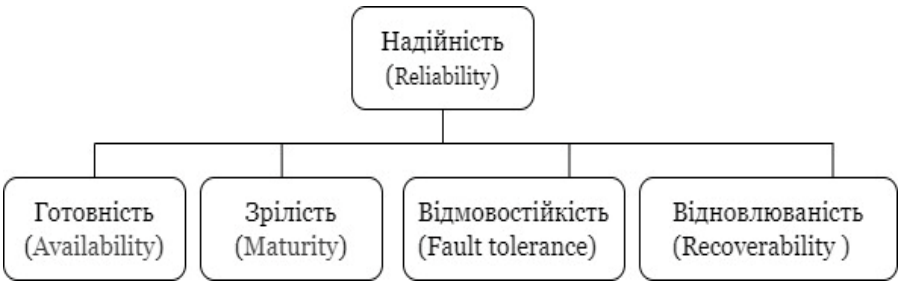


Рис. 1.2. Показники надійності ПЗ відповідно до моделі якості ISO/IEC 25010:2011

- готовність (availability) – ступінь, до якого система, продукт або компонент працюють і доступні, коли це необхідно для використання;
- зрілість (maturity) – ступінь, до якого система, продукт або компонент відповідають потребам у надійності при нормальній експлуатації;
- відмовостійкість (fault tolerance) – ступінь, до якого система, продукт або компонент працюють належним чином, попри наявні апаратні чи програмні збої;
- відновлюваність (recoverability) – ступінь, до якого у разі перерви або збою продукт або система можуть відновити дані, на які безпосередньо впливає, і відновити потрібний стан системи.

Також, визначають такі основні складові функціональної надійності програмних систем [3]:

- *безвідмовність* – властивість системи виконувати свої функції під час використання;
- *працездатність* – властивість системи виконувати коректно (відповідно до очікувань користувача) поставлені завдання протягом періоду використання;
- *безпека* – властивість програми бути безпечною для людей та навколишніх систем;
- *захищеність* – властивість програми протидіяти випадковим або навмисним вторгненням в її операційну діяльність.

Відмова – це подія, після появи якої характеристики технічного об’єкта виходять за допустимі рамки системи. Термін «відмова» є основним в теорії

надійності [3]. Відмова функціонування системи відповідає випадку, коли операційна поведінка програми відхиляється від вимог і може бути викликана низкою різних чинників [4]: помилки в програмній реалізації системи допущені на етапі розроблення; некоректність вхідних даних; дії користувача, які не відповідають умовам користування системою; несправність апаратних компонент системи.

Технічні системи можуть бути невідновлюваними та відновлювальними, тривалого і короткого часу роботи, резервованими і нерезервованими [4]. Невідновлювальні системи після відмови не можуть бути повернені в стан функціонування. Технічна система називається відновлюваною, якщо вона може продовжувати виконання своїх функцій після усунення відмови, що викликала зупинку її функціонування.

Кількісну оцінку надійності системи здійснюють за допомогою критеріїв і показників надійності. Критерій – ознака оцінки надійності, а показник надійності є числовим значенням критерію [5]. Не існує єдиного показника, що достатньо повно характеризує надійність складної системи.

Для забезпечення показників і характеристик якості та надійності програмного забезпечення під час проєктування архітектури ПЗ використовують, так звані, тактики забезпечення якості ПЗ [6]. Тактика забезпечення якості ПЗ – це набір рекомендацій або правил, які дозволяють забезпечувати визначені показники і характеристики якості ПЗ на високому рівні.

Одним із важливих показників надійності ПЗ є його готовність. На практиці системні вимоги щодо готовності розробляють відповідно до стійкої готовності (на відміну від миттєвої готовності). Стабільна готовність — це вимірювання часу безперебійної роботи системи протягом достатньо тривалого часу (90 днів, один рік, загальна місія тощо).

На рисунку 1.3 показано тактики забезпечення готовності ПЗ, які класифікують відповідно до того, чи стосуються вони виявлення несправностей, відновлення чи запобігання.

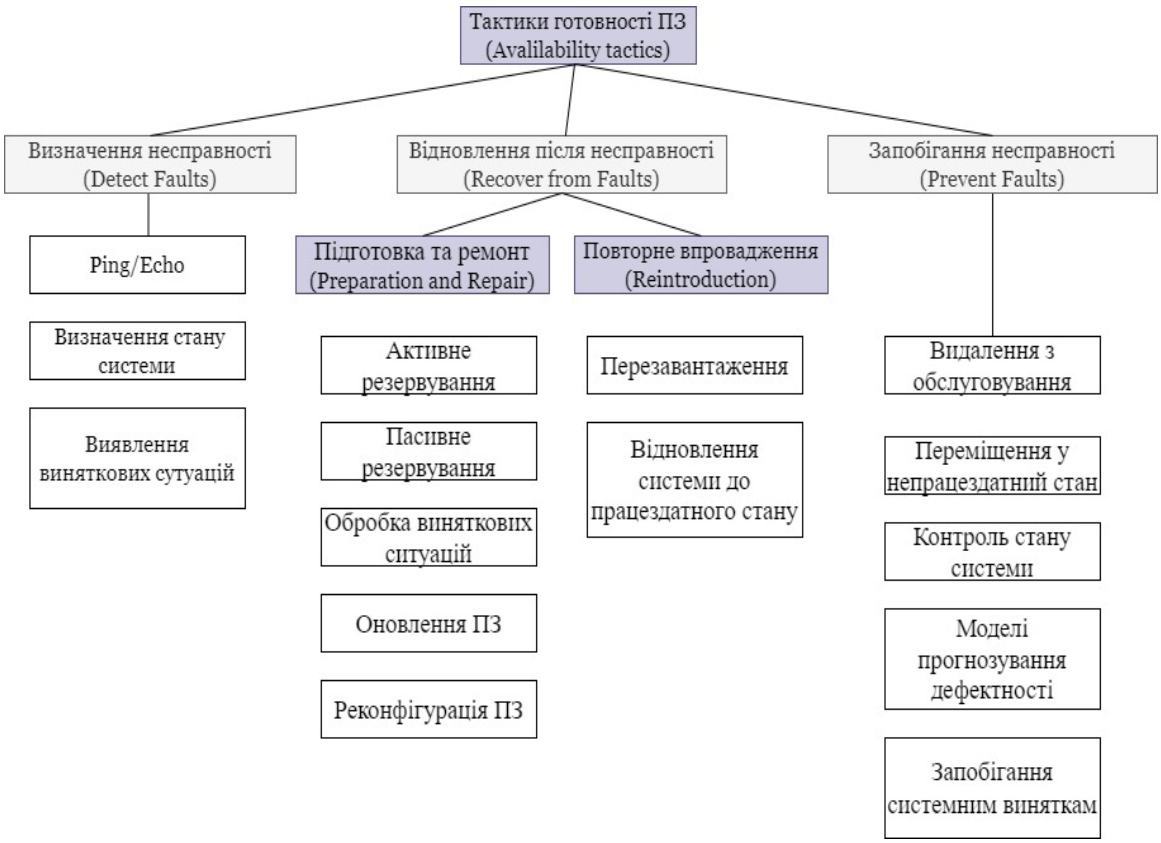


Рис. 1.3. Тактики забезпечення готовності ПЗ

До тактик виявлення несправностей належить тактика Ping/Echo, яку використовують для визначення доступності та затримки роботи системи. Визначення стану системи дозволяє виявити завислі або некоректні процеси. Тактика виявлення виняткових ситуацій дозволяє системі передбачити та надалі правильно опрацювати певні випадкові ситуації в роботі програмної системи.

Тактика відновлення несправностей поділяється на тактику підготовки і ремонту та тактику повторного впровадження. Тактики підготовки та відновлення включають активне резервування, пасивне резервування, обробку винятків та оновлення програмного забезпечення. Тактики повторного впровадження включають перезавантаження або відновлення системи до працездатного стану.

До тактик запобігання збоїв належать: тактика видалення з обслуговування передбачає переведення компонента системи у стан непрацездатності з метою пом’якшення потенційних збоїв системи; контроль стану системи дозволяє перевірити, що система працює в межах своїх номінальних робочих параметрів; прогнозування дефектності дозволяє виявити компоненти системи, які можуть містити дефекти і виправити їх.

1.2. Аналіз і класифікація моделей надійності ПЗ

Галузь досліджень надійності програмних систем бере свій початок з 70-х років минулого століття. Відтоді було розроблено понад 200 моделей надійності ПЗ, але питання, як кількісно оцінити надійність програмного забезпечення залишається відкритим [7]. Модель надійності ПЗ - це математична модель, створена для оцінки залежності надійності програмного забезпечення від деяких певних чинників (параметрів). Значення таких параметрів можуть бути відомим або отримані експериментальним способом під час дослідження процесу функціонування програмного забезпечення [8]. Але попри значну кількість моделей надійності, жодна модель повністю не відображає надійність програмного забезпечення.

Схема класифікацій моделей надійності постійно змінюється із постійним розвитком галузі розробки програмного забезпечення. Класифікація моделей надійності програмного забезпечення корисна для порівняння різних моделей надійності. Різні набори моделей полегшують отримання нових моделей [9].

Моделі надійності програмного забезпечення можна класифікувати за допомогою двох підходів: перший на основі історії відмов, а другий – на основі вимог до даних [10].

На основі історії відмов моделі надійності програмного забезпечення згруповані в такі класи [11]:

Моделі на основі часу між відмовами (Time Between Failure Models) – у цьому класі моделей досліджуваним процесом є час між відмовами. Передбачається, що час між *(i-1)-ю* і *i-ю* відмовами є випадковою величиною, що відповідає розподілу, параметри якого залежать від кількості помилок, які залишилися в програмі протягом цього проміжку часу. Оцінки цих параметрів отримують з експериментальних значень між відмовами, а після цього визначають параметри надійності ПЗ, які отримують за вибраними моделями [12].

Моделі на основі кількості помилок (Fault Count Models) – у цих моделях випадкова величина значущості - це кількість несправностей (збоїв), що виникають через задані інтервали часу.

Моделі на основі засівання помилок (Fault seeding Models) – у цьому класі моделей вважається, що програма має невідому кількість власних помилок. Крім цього, засівають відому кількість несправностей. Програму тестують і спостерігають кількість засіяних і нативних несправностей. Моделі цього класу отримують оцінку вмісту несправності програми перед додаванням помилок, а потім з цього значення розраховують параметри надійності програмного забезпечення [13].

На основі вимог до даних моделі надійності програмного забезпечення можна розділити на дві основні групи, тобто емпіричні моделі та аналітичні моделі [14].

Емпіричні моделі. Емпірична модель надійності програмного забезпечення створює зв'язок або набір зв'язків між показниками надійності програмного забезпечення та відповідними показниками програмного забезпечення, такими як складність програми. Ці відношення (відношення) використовуються для вимірювання надійності програмного забезпечення, для цього необхідно знайти відповідні показники надійності програмного забезпечення та розробити точний тип і форму взаємозв’язків серед метрик і показників надійності. Моделі цього типу – модель Moranda, модель Hallstead і модель Schneider [15].

Аналітичні моделі. Потребують певної форми даних, які отримують внаслідок відмов програмного забезпечення. Моделі ґрунтуються на підборі відповідного розподілу з необхідними припущеннями для полегшення набору даних, зібраних під час тестування програмного забезпечення, та передбачення 0параметрів надійності програмного забезпечення з відповідного розподілу. Аналітичні моделі можна розділити на статичні та динамічні моделі на основі поведінки зібраних даних, що залежить від часу. Залежно від типів даних, які використовують при розробці моделей, статичні моделі можна далі розділити на моделі області помилок (комбінаційна модель) і моделі області даних. А динамічні моделі на основі інтервалу часу, який використовується, далі групуються в безперервні моделі часу та моделі дискретного часу [16].

Також однією із відомих класифікацій моделей надійності є поділ на моделі типу «чорної скриньки» і «білої скриньки». Моделі типу «чорної скриньки» не враховують внутрішню структуру програмного забезпечення при оцінці надійності і називаються так, оскільки розглядають програмне забезпечення як монолітну сутність (чорний ящик) і враховують лише дані про помилки або показники, які збираються, якщо дані тестування недоступні [17]. До моделей такого типу належать, наприклад: модель Jelinski-Moranda [18, 19], модель Goel-Okumoto на основі неоднорідного пуассонового процесу [20], базова модель часу виконання Musa [21, 22], баєсівська модель Littlewood-Verrall [23].

Моделі надійності програмного забезпечення типу «білої скриньки» протягом останніх десятиліть стали популярними у дослідженнях надійності, оскільки розглядають внутрішню структуру програмного забезпечення при оцінці надійності на відміну від моделей «чорної скриньки», які моделюють лише взаємодію програмного забезпечення з системою, в якій воно працює. Їх популярність, також пов’язана зі зростанням складності програмного забезпечення, яке переважно має складну модульну/багатокомпонентну архітектуру. Існує твердження, що моделі «чорної скриньки» неадекватні для застосування до програмних систем у контексті програмного забезпечення на основі компонентів, що призводить до збільшення повторного використання компонентів та складної взаємодії між цими компонентами у великій програмній системі. Крім того, прихильники моделей білого ящика стверджують, що моделі надійності, які враховують надійність компонентів, при обчисленні загальної надійності програмного забезпечення, дають більш адекватну оцінку надійності [24].

У моделях «білої скриньки» спочатку визначають компоненти та модулі з припущенням, що модулі проектуються, реалізовуються та тестуються незалежно. Архітектура програмного забезпечення потім ідентифікується не в сенсі традиційної архітектури програмної інженерії, а скоріше у сенсі взаємодії між компонентами. Взаємодії визначаються як передачі керування між модулями, по суті, маючи на увазі, що архітектура системи є графом потоку керування, де

вузли графа представляють модулі, а його переходи представляють передачу керування між модулями. Поведінку відмов для цих модулів (і пов’язаних інтерфейсів) потім визначають з позиції частоти відмов або надійності. Поведінку відмови потім поєднують з архітектурою, щоб оцінити загальну надійність програмного забезпечення, як функцію надійності компонентів. Моделі «білої скриньки» поділяються на моделі на основі архітектурного підходу, моделі на основі шляхів виконання ПЗ і адитивні моделі [25].

Адитивні моделі – це клас моделей, де не враховується явно архітектура програмного забезпечення. Натомість вони зосереджені на оцінці загальної надійності програми за допомогою даних про збій компонента. Слід зазначити, що ці моделі враховують зростання надійності програмного забезпечення. Вони називаються адитивними моделями, оскільки за припущення, що надійність компонента може бути змодельована негомогенним пуассоновим процесом, інтенсивність відмов системи може бути виражена як сума інтенсивностей відмов компонентів системи [26]. До класу адитивних моделей належить модель Xie and Wohlin [27]. Ця модель розглядає програмну систему, що складається з n компонентів, які можна розробляти паралельно та тестувати незалежно. Припускаючи, що збій компонента є системним збоєм, з погляду надійності, систему можна розглядати як послідовну систему.

Моделі на основі шляхів виконання засновані на тих же загальних принципах, що й моделі на основі архітектурного підходу, за винятком того, що цей підхід, використовує поєднання архітектури програмного забезпечення з поведінкою збоїв, що описують як метод на основі шляху, оскільки надійність системи обчислюється з урахуванням можливих шляхів виконання програми, яка, переважно, отримується експериментально шляхом тестування [28]. До моделей даного типу належать: модель Shooman [29] – одна з найперших моделей, яка розглядає надійність модульних програм, вводячи підхід на основі шляху, використовуючи частоти, з якими виконуються різні шляхи; модель Krishnamurthy and Mathur [30] – модель для поєднання архітектури та поведінки відмов спочатку включає обчислення оцінок надійності шляху на основі

послідовності компонентів, що виконуються для кожного тестового запуску, а потім усереднення їх по всіх тестових запусках для отримання оцінки надійності системи; модель Yacoub [31], яка є специфічною для програмного забезпечення на основі компонентів, аналіз якого суворо базується на сценаріях виконання.

Значну частину серед вище перелічених типів моделей типу "білої скриньки" складають моделі на основі архітектурного підходу [32, 33, 34]. Моделі даного типу для опису функціонування системи використовують граф потоку керування. Передбачається, що передача керування між модулями має Марковську властивість, яка означає, що з огляду на знання модуля, який контролює в будь-який момент часу, майбутня поведінка системи умовно не залежить від поведінки в минулому. Даний підхід розглядає моделювання архітектури ПЗ із використанням ланцюга Маркова із дискретним часом, неперервним часом та напівмарковським процесом [35].

При описі архітектури із використанням ланцюга Маркова з дискретним часом взаємодії передача керування між модулями відбувається у встановлені проміжки часу. Дані моделі потребують, щоб були визначені такі параметри: ймовірність перебування системи в різних станах, матриця ймовірностей переходів між станами і інтенсивність відмов модулів системи. У роботі [36] модель Gokhale, яка будується на основі ланцюга Маркова із неперервним часом і використовує набір регресійних тестів для опису архітектури ПЗ.

Моделі з неперервним часом, виражають взаємодію модулів у вигляді ланцюга Маркова з неперервним часом, тобто, тривалість перебування в модулі експоненційно розподілений. До найбільш відомих моделей цього класу зараховують моделі Laprie [37] та Littlewood [38]. Модель Laprie вважають частковим випадком моделі Littlewood. В цій моделі визначено такі базові припущення: взаємодія модулів описується ланцюгом Маркова з неперервним часом; задано параметр середнього часу виконання модуля; відома матриця імовірностей переходів між модулями; для кожного модуля відома інтенсивність відмов.

Моделі, побудовані на основі напівмарківського ланцюга, описують взаємодію модулів програмної системи напівмарківським процесом, тобто, тривалість перебування в певному модулі має функцію розподілу, що залежить від стану, в якому перебуває система, а також, і від стану, в який вона перейде [39].

1.3. Аналіз надійності програмних систем із використанням моделі на основі ланцюгів Маркова вищого порядку

Для підвищення ступеня адекватності моделей надійності, і, відповідно, для підвищення достовірності оцінювання надійності складних програмних систем, необхідно враховувати те, що імовірність переходу в наступний стан (передача потоку управління до іншого модуля програми) може залежати не тільки від поточного стану, а й від попередніх станів при потраплянні в цей стан (тобто який модуль програми виконувався до того, як почав виконуватись поточний модуль).). У дослідженнях [40-42] запропоновано застосування ланцюгів Маркова вищого порядку (ЛМВП), які враховують послідовність і взаємозалежність виконання модулів програмних систем (ПС) і дають можливість більш точно оцінювати надійність цих систем.

В роботах [42-43] автор розробив і дослідив модель надійності ПЗ на основі ланцюгів Маркова вищого порядку з неперервним часом, яка містить такі складові:

- $N -$ кількість програмних модулів в системі;
- $\{C_i\}$ – граф, вершини якого відображають модулі системи, а ребра показуються, як модулі взаємодіють між собою і відбувається передача управління між модулями;
- $A = \{a_{ij}(t)\}$ – матриця, для відображення інтенсивності переходів між станами програмної системи;
- $P = \{p_i(t)\} -$ вектор ймовірності перебування системи в певному стані C_i в момент часу t;

- $\lambda_i\,(t)$ – інтенсивність відмов і-ого програмного модуля (для ЛМВП значення залежить від попередніх станів, з яких програмна система попала в стан *i*).

Дана модель інтенсивності відмов системи із N модулів виглядає так:

$$\lambda(t) = \sum_{i=1}^{N} p_i(t)\lambda_i(t), \tag{1.1}$$

де $\lambda_i\,(t)$ – інтенсивність відмов і-го модуля, $p_i(t)$ – ймовірність виконання і-ого модуля в момент часу t.

При моделюванні потоку управління між модулями системи, як процесу Маркова з неперервним часом, ймовірність перебування системи в і-му стані , $p_i(t)$ можна визначити за допомогою розв'язку системи диференціальних рівнянь Колмогорова-Чепмена:

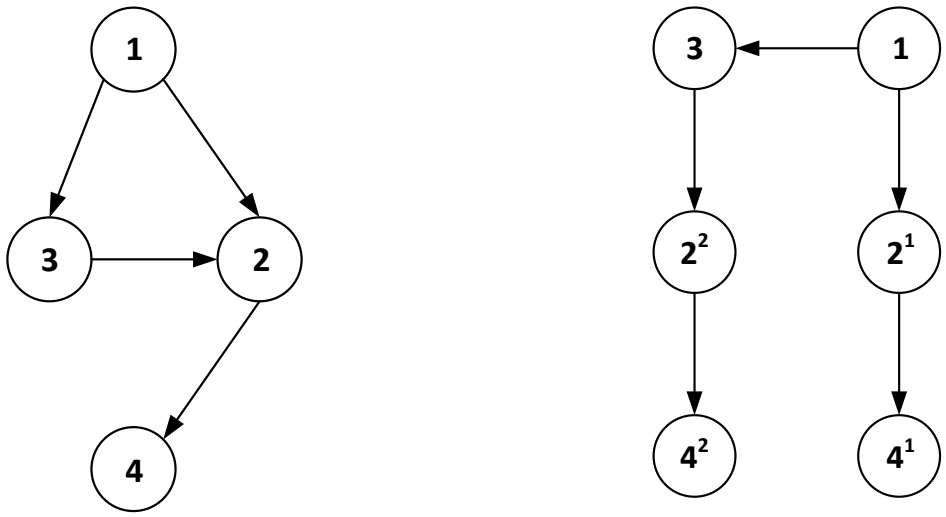
$$\frac{dp_i(t)}{dt} = -\sum_{i\in S} a_{ij}(t)p_i(t) + \sum_{j\in S} a_{ji}(t)p_j(t)\,, i \in S, \tag{1.2}$$

де $a_{ij}(t)$ – інтенсивність переходу зі стану і в стан j, а S – множина всіх станів системи.

В роботах [44, 45] вплив процесу вищого порядку визначається, як залежності інтенсивності переходу між станами від ланцюжка шляхів потрапляння в поточний стан. Відповідно для цього випадку система рівнянь (1.2) повинна бути модифікована, враховуючи ці залежності. Робота [43] демонструє дещо інший підхід для урахування процесу вищого порядку, де побудова еквівалентного процесу першого порядку, зводить задачу до обчислення імовірностей $p_i(t)$ на основі класичної системи рівнянь Колмогорова-Чепмена (1.2).

У роботах [42–43] запропоновано подання процесу Маркова вищого порядку еквівалентним процесом першого порядку із додатковими фіктивними станами, де кожен стан вхідного графа розділяється на відповідну кількість фіктивних станів, враховуючи всі можливі унікальні шляхи до цього стану. Тоді розв'язання цієї задачі зводиться до використання інструментів теорії графів.

Підхід запропонований у роботі [43] можна розглянути наступним чином на прикладі для системи із чотирьох елементів (рис. 1.4(а)). Для Марковського процесу другого порядку граф буде містити шість станів (рис 1.4(б)), оскільки в стан C_4 можна потрапити двома різними шляхами $C_1 \rightarrow C_2 \rightarrow C_4$ і $C_3 \rightarrow C_2 \rightarrow C_4$, тому стани C_2 і C_4 розщеплюються на додаткові віртуальні стани.



а) *б)*

Рис. 1.4. Приклад подання процесу Маркова другого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами (а – граф станів системи, б – граф станів системи з фіктивними станами).

Визначимо M^k, як матрицю, елементи якої m_{ij}^K відображають кількість можливих переходів (шляхів) порядку K зі стану C_i до C_j.

Тоді матрицю M^1, елементами якої є кількість переходів першого порядку між відповідними станами, зображеними на рис. 1(а) (m_{12}^1= 1 – в матриці M^1 є один можливий шлях першого порядку зі стану C_1 в C_2):

$$M^1 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Матриця M^2, яка міститиме кількість ланцюжків зі стану C_i в C_j до другого порядку включно, виглядає так (m_{12}^2= 2 зі стану C_1 в стан C_2 є можливі два шляхи – один довжиною 2 (ланцюг Маркова другого порядку) $C_1 \rightarrow C_3 \rightarrow C_2$ та один ланцюжок довжиною $C_1 \rightarrow C_2$.):

$$M^2 = \begin{pmatrix} 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Отже, кількість ланцюжків K-порядку зі стану C_i в стан C_j можна визначити за допомогою формули, що базується на методі Флойда:

$$m_{ij}^K = \sum_{j=1}^{N} \left(m_{il}^{K-1} + e_{il}\right) m_{lj}^1, \tag{1.3}$$

тут N – кількість всіх станів системи, e_{ij}– елементи M^k, матриці.

Базуючись на рівнянні (1.3), можна отримати еквівалентний граф станів системи $\{C_i\}$, який є еквівалентним представленням початкового графа у випадку процесу K-го порядку. Побудова і розв'язок системи рівнянь Колмогорова-Чепмена (1.2) для еквівалентного графа дає можливість отримати імовірності перебування системи в станах C_i, що відповідає імовірностям виконання і-го програмного модуля в момент часу t. У випадку Марковського процесу змінного порядку у виразі (1.3) для кожного стану використовують різне значення порядку процесу.

1.4. Методи прогнозування дефектності програмного забезпечення на основі метрик коду

Дефект програмного забезпечення – це недолік компонентів або модулів системи, який негативно впливає на зовнішній вигляд, роботу, функціональні можливості або продуктивність і може призвести до відмови певного функціоналу або неправильної роботи системи. Більшість програмних дефектів виникають на етапі розробки програмного забезпечення у вихідному коді програми і є спричинені низкою факторів, які виникають на різних етапах життєвого циклу продукту, а саме: помилки у програмному коді, проблеми комунікації, неточності вимог до ПЗ, слабка документація і дизайн, складність системи, терміни виконання проєкту, людський фактор, недостатнє тестування і т.д. [46].

З метою підвищення якості та надійності програмного забезпечення застосовують методи прогнозування дефектності програмного забезпечення для виявлення потенційних помилок. Традиційні методи виявлення помилок базуються на аналізі метрик програмного продукту і використовуються для класифікації потенційно дефектних модулів або передбачення приблизної кількості помилок у певному модулі системи [46]. В результаті метод прогнозування дефектності програмного забезпечення, може допомогти розробникам виявити дефекти на основі наявних програмних показників, використовуючи методи аналізу даних, і тим самим покращити якість, і відповідно надійність програмного забезпечення, що зрештою призводить до зниження витрат на розробку програмного забезпечення на етапі розробки та обслуговування.

Існує два основні класи методів класифікації в прогнозуванні дефектності програмного забезпечення: статистичний підхід (використовують традиційні статистичні моделі, такі як, наприклад, регресійні моделі) і підхід на основі машинного навчання [47].

Серед статистичних моделей прогнозування дефектності досить добре відомий підхід, при якому використовують метрики розміру та складності. Ця модель використовує програмний код як основу для передбачення дефектів. Точніше, рядки коду (LOC) використовуються разом з концепцією моделі складності, розробленою МакКейбом. Використовуючи рівняння регресії, можна отримати прості оцінки метрик прогнозу, врахувавши сумарну кількість дефектів під час тестування та після певного часу експлуатації програми. Значним недоліком даної моделі є те, що моделі розміру та складності припускають, що дефекти є прямою функцією розміру або дефекти виникають через складність програми, але ігнорують ефект людського фактора при створенні програмного забезпечення [48].

Ще одною статистичною є модель передбачення щільності дефектів. Щільність дефектів – це міра загальної кількості підтверджених дефектів, поділена на розмір об’єкта (модуля) програмного забезпечення, який

вимірюється. Кількість відомих дефектів – це кількість загальних дефектів, виявлених щодо конкретного програмного забезпечення за певний проміжок часу. Розмір — це як нормалізатор, який дозволяє порівнювати різні програмні сутності (тобто модулі, випуски, продукти). Розмір зазвичай вимірюється або в рядках коду, або в функціональних точках. Щільність дефектів корисна для порівняння дефектів у різних компонентах програмного забезпечення, щоб визначити компоненти високого ризику та пов'язані з ними ресурси. Крім того, його також можна використовувати для порівняння між різними програмними продуктами з позиції якості [49].

На сьогоднішній день підхід для прогнозування дефектності програмного забезпечення на основі машинного навчання є найбільш популярним через свої значні можливості і переваги при вирішенні даного завдання. Алгоритми машинного навчання продемонстрували велике практичне значення у вирішенні широкого кола інженерних проблем, що охоплюють прогнозування збоїв, помилок і дефектів, оскільки програмне забезпечення стає складнішим. Алгоритми машинного навчання дуже корисні, коли проблемні області не чітко визначені, людські знання обмежені та потрібна динамічна адаптація до умов, що змінюються, для розробки ефективних алгоритмів [50].

Загальний підхід для прогнозування дефектності програмного забезпечення на основі машинного навчання складається із таких етапів (рис. 1.5) [51]:

Маркування: дані про дефекти необхідно зібрати для навчання моделі прогнозування. У цьому процесі зазвичай виконується ідентифікація модулів системи на основі тестування на дефектні і без дефектів (маркування TRUE або FALSE).

Вилучення метрик і створення навчальних наборів: цей крок включає вилучення ознак для передбачення міток екземплярів. Загальними характеристиками для передбачення дефектів є показники складності, ключові слова, зміни, розмір програмного коду та структурні залежності. Комбінуючи мітки та функції екземплярів, ми можемо створити навчальний набір, який буде використовуватися машинним навчальним для побудови моделі передбачення.

Побудова моделі прогнозування: на даному етапі відбувається вибір моделі (моделей) машинного навчання, які можна використовувати для передбачення за допомогою навчального набору.

Оцінка: для оцінки моделі передбачення потрібен набір даних тестування, крім навчального набору.

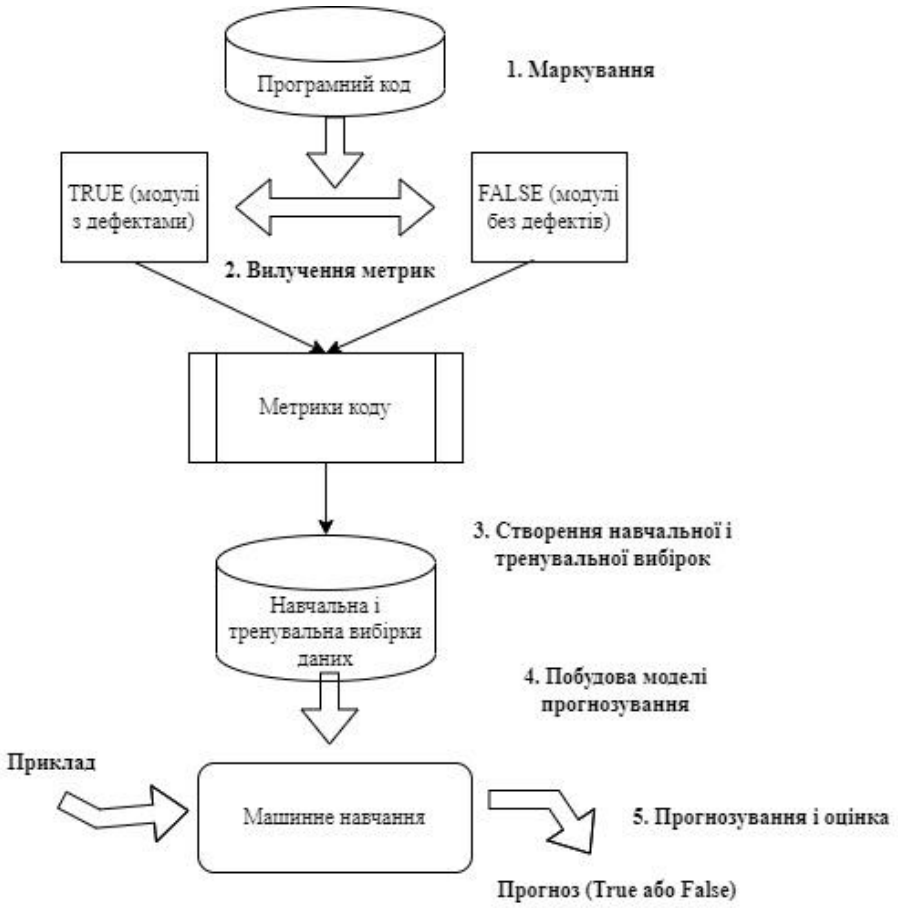


Рис. 1.5 Загальний процес прогнозування дефектів

Одним із перших кроків процесу прогнозування дефектів є вилучення ознак/метрик коду. Метрика коду - міра, що дозволяє отримати чисельне значення деякої якості програмного забезпечення або його специфікацій, до метрик коду належить: кількість рядків коду; цикломатична складність; кількість операторів і операндів, зв'язність коду; степінь покриття коду тестами; тощо. І важливим питанням в процесі прогнозування дефектів, яке є відкритим і актуальним сьогодні, є – які метрики коду найбільше впливають на його якість і надійність.

Широко використовують різноманітні методи машинного для вибору ознак, перед процесом прогнозування дефектів. Виділення або відбір ознак – це процес вибору найважливіших ознак вибірки даних для зменшення кількості вхідних

змінних шляхом усунення зайвих або невідповідних функцій і звуження набору функцій до тих, які найбільше стосуються моделі машинного навчання. [52]. Використання методів вибору ознак є хорошим розв'язанням проблеми високої розмірності вибірки даних і ці методи активно використовуються в контексті прогнозування дефектів програмного забезпечення. Існує велика кількість досліджень на тему вибору ознак в прогнозуванні дефектності, головною ціллю яких є вибрати найточнішу комбінацію метрик коду для прогнозування дефектів.

У роботі [53] було досліджено 46 методів вибору ознак із використанням класифікатора Decision Tree на 25-и наборах даних про дефекти програмного забезпечення з 4-х репозиторіїв програмного забезпечення. Експериментальні результати показали, що не існує єдиного найкращого методу для вибору ознак, оскільки їх відповідні показники залежать від вибору класифікаторів, показників оцінки ефективності та набору даних (проте були надані рекомендації для використання методів).

Дослідження [54] робить акцент на двох напрямках : вибір функції та передача екземплярів відстань-вага. При зменшенні відмінності між проєктами з погляду інженерії функцій, впроваджують технологію трансферного навчання для побудови міжпроєктної моделі передбачення дефектів WCM-WtrA і моделі з багатьма джерелами Multi-WCM-WTrA. Дані результати показують, що розроблений метод має середнє покращення на 23% порівняно з алгоритмом TCA + для наборів даних AEEEM і середнє покращення на 5% для наборів даних ReLink.

У роботі [55] було досліджено 12 автоматизованих методів вибору ознак щодо узгодженості, кореляції, продуктивності, обчислювальної вартості та впливу на розміри інтерпретації.

Також активно, поряд із наявними і добре відомими методами вибору ознак, розробляються нові методи. Запропоновано гібридний метод обгортки з багатьма фільтрами (hybrid multi-filter wrapper method for feature selection) для вибору ознак при передбаченні дефектності програмного забезпечення, який об'єднує переваги методів фільтрування і обгортки [56]. У дослідженні [57] запропоновано

виконувати вибір ознак, використовуючи алгоритму Firefly (FA). FA, – це нова техніка еволюційного обчислення, яка була натхненна процесом спалаху світлячків. Цей метод може швидко шукати в просторі функцій оптимальну або майже оптимальну підмножину ознак. Запропоновано новий метод обгортки, який складається з двох основних етапів відбору ознак і класифікації [58]. На першому етапі використовується Grey Wolf Optimization (GWO), щоб знайти найкращі характеристики в наборі даних, другий етап оцінює ознаки за допомогою машинного класифікатора опорних векторів (SVM).

Після того, як ідентифіковано і вибрано метрики на основі, яких буде відбуватись прогнозування, потрібно вибрати, який саме алгоритм машинного навчання використовувати для етапу побудови моделей прогнозування.

В роботі [59] описано порівняльне дослідження різних алгоритмів машинного навчання для класифікації програмних модулів на такі що схильні до дефектів і не схильні до дефектів. Було розглянуто ряд відомих алгоритмів машинного навчання Decision Trees, Artificial Neural Network (ANN), K nearest neighbour, SVM and Ensemble Learning, де найкраще показала себе техніка Stacking Ensemble technique із найкращим результатом для всіх наборів даних з точністю передбачення дефектів більше як 0,9. Праця [60] пропонує модель прогнозування розподілу дефектів (Deep belief network prediction model, DBNPM), систему для визначення того, чи містить програмний модуль дефекти. Ключовою ідеєю DBNPM є технологія мережі глибоких переконань (Deep belief network - DBN), яка є ефективною технікою глибинного навчання в обробці зображень та природної мови, характеристики якої подібні до дефектів вихідної програми.

У роботі [61] описано прогнозування дефектів на основі зменшення масштабу двох фокусів (характеристики дефектів і дефекти). Тут наведено глибокий аналіз різних ключових питань включаючи те, як створити набір порівняння характеристик дефектів і набір порівняння дефектів, теорію відштовхування для характеристик дефектів і дефектів, а також методологію і модель для прогнозування дефектності. Експериментальні результати

демонструють, що ця методологія прогнозування є дуже ефективною для прогнозування якості програмного забезпечення космічних проектів.

Моделі прогнозування дефектності можна будувати на основі даних про проєкт з попередніх версій або релізів, проте, у випадку коли це новий проєкт, таких даних немає для передбачення, тому використовують підхід, що відомий, як прогнозування дефектів між проєктами (CPDP – cross-project defect prediction). У роботі [62] описано застосування бандитського алгоритму (BA – bandit algorithm) до CPDP, щоб вибрати найбільш відповідний навчальний проєкт із набору проєктів. Експеримент, який було виконано на двох наборах даних (NASA і DAMB, загалом 12 проєктів) показує, що використання BA для прогнозування дефектів у CPDP є перспективним і може перевершити існуючі підходи. Дослідження [63] пропонує алгоритм на основі трансферного навчання TSboostDF. TSboostDF інтегрує метод вибірки BLS, який базується на вазі вибірки, з методом трансферного навчання, щоб подолати недоліки традиційних алгоритмів, що використовують в CPDP.

Також, окрім прогнозування дефектності, є важливим визначення рівня серйозності дефектів ПЗ, який вказує на вплив помилки на роботу програми та те, як швидко ці помилки потрібно усунути, оскільки встановлення пріоритетів цих дефектів вручну на основі досвіду може бути неточним прогнозом серйозності, яка затримує виправлення критичних помилок. У роботі [64] описано методи автоматизації призначення відповідного рівня серйозності на основі результатів звіту про помилку за допомогою різних методів вбудовування слів (word embedding techniques).

Прорив у технологіях машинного навчання, особливо розвиток методів на основі нейронних мереж та глибокого навчання, призвів до розв'язання багатьох проблем за допомогою цих методів і особливо активно дані методи почали використовувати у процесі прогнозування дефектності програмного забезпечення.

В роботі [65] для проблеми прогнозування дефектності програмного забезпечення дослідники запропонували алгоритми глибинного навчання, щоб

автоматично вивчати семантичні представлення програм і використовувати це уявлення для ідентифікації схильного до дефектів коду. Використання цих неявних функцій показує кращі результати, ніж попередні підходи, засновані на явних функціях, таких як метрика коду.

У дослідженні [66] автори пропонують підхід для прогнозування дефектів програмного забезпечення на рівні змін із використанням DBN (Deep Belief Network). Мережа навчається на основі традиційних метрик коду і генерує нові виразні функції та використовує їх у класичних класифікаторах машинного навчання. Вони витягують зв'язки з традиційних метрик коду, таких як кількість змінених модулів, каталогів і файлів, додані та видалені рядки, а також деякі функції, пов'язані з досвідом розробника. Пізніше автори запропонували підхід «TLEL» [67] на основі дерева рішень та ансамблевого навчання для класифікації.

В роботі [68] також використовують DBN, але в інший спосіб. Для прогнозування дефектів на основі семантики коду автори розробили aDBN для автоматичного вивчення семантичних ознак з вихідного коду. AST (Abstract Syntax Tree) програм і зміни вихідного коду використовуються для випадків передбачення рівня змін на рівні файлу, як вхідні дані для мережі. Потім автори використовують класичні класифікатори машинного навчання та вилучені функції, щоб класифікувати файли вихідного коду, чи є вони помилковими чи без помилок.

Модель на основі LSTM (long short-term memory) нейронної мережі було використано в роботі [69] для вивчення як семантичних, так і синтаксичних особливостей коду. Запропонований підхід представляє код як послідовність маркерів коду, яка подається в систему LSTM для перетворення коду у вектор ознак і станів ознак, що представляє семантичну інформацію маркера. Пізніше модель Tree-LSTM була розроблена авторами з використанням представлення AST (Abstract Syntax Tree) як вхідних даних [70].

Методика пошуку помилок на основі нейронної мережі запропоновано в роботі [71]. Автори тренують нейронну мережу на прикладах дефектного та правильного коду, а потім використовують отриманий двійковий класифікатор

для виявлення помилок. Щоб підготувати позначений набір даних, автори використовують програмне забезпечення для виявлення статичних помилок, щоб ідентифікувати певний тип помилок. Код представлений у вигляді послідовності токенів і перетворюється у вектор реального значення за допомогою одноразового кодування для кожного маркера. Тоді як модель використовує нейронну мережу з LSTM.

Інша модель, заснована на глибинному навчанні для прогнозування дефектності запропонована в [72]. Навчання нейронної мережі використовує техніку втрат триплетів і техніку зваженої перехресної ентропії.

У роботі [73] представлено методику прогнозування дефектності програмного забезпечення, засновану на моделі автоенкодерів із зниженими шумами. Автоенкодер із накопиченням шумів використовують для вилучення особливостей вищого рівня із традиційних показників. Для класифікації використовують двоетапне ансамблеве навчання на основі нейронних мереж як класифікаторів.

Модель прогнозування дефектності програмного забезпечення була побудована в роботі [74] на основі сіамських паралельних повнозв'язаних нейронних мереж. Ця модель використовує архітектуру парних паралельних сіамських мереж і підхід глибинного навчання. Мережа створює функції високого рівня, які використовують для класифікації.

1.5. Висновки до розділу 1

В даному розділі було проведено огляд та аналіз літературних джерел відповідно до теми дисертаційного дослідження. Виконано аналіз і опис основних тверджень і критеріїв із теорії надійності.

Представлено детальний опис моделей надійності ПЗ і їх особливостей. Із аналізу можна зробити висновки, що існує достатньо велика кількість моделей надійності, які мають свої переваги і недоліки. Одним із суттєвих недоліків відомих в літературі моделей класу «білої скриньки» є складність їх практичного використання в індустрії програмного забезпечення. Багато параметрів у цих

моделях вважають наперед відомими, але не описані способи отримання їх значень для реальних програмних продуктів, що для багатьох параметрів є нетривіальною задачею. Разом з тим, моделі оцінювання надійності ПЗ на основі архітектурного підходу у більшості випадків використовують теорію класичних Марковських процесів, припускаючи незалежність виконання модулів програмної системи, що є спрощеним описом реального процесу виконання ПЗ.

Розглянуто використання моделі на основі ланцюгів Маркова вищого порядку. Дана модель враховує те, що імовірність переходу в наступний стан (наприклад, передачі потоку управління до іншого модуля програми) може залежати не тільки від поточного стану, а й від передісторії потрапляння в цей стан, що може підвищити достовірності оцінювання надійності складних програмних систем.

Зроблено огляд методів прогнозування дефектності ПЗ на основі метрик коду. Підсумовуючи можна зробити висновок, що за останні роки зріс інтерес до використання моделей дефектності ПЗ та класифікації програмних дефектів на основі метрик коду та характеристик проєкту. Як показав аналіз літератури, наявні підходи мають недостатньо високу точність прогнозу; немає єдиної думки щодо впливу метрик коду ПЗ на показники його якості, і, зокрема, надійності; питання переносимості результатів, отриманих на основі даних про одні проекти на характеристики іншого ПЗ є відкритим, а, відповідно, залишається актуальним питання визначення набору метрик програмного коду, які найбільше впливають на його дефектність.

РОЗДІЛ 2. МЕТОДИ АВТОМАТИЗАЦІЇ ДЛЯ МАРКОВСЬКИХ МОДЕЛЕЙ АНАЛІЗУ НАДІЙНОСТІ ПЗ

В даному розділі представлено опис розроблених методів для автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами та визначення функції працездатності для Марковських моделей надійності ПЗ.

Практичне використання методу автоматизації подання Марковського процесу вищого порядку продемонстровано на прикладі оцінки показників надійності програмного забезпечення польотів наносупутників CubeSat. Дане дослідження спрямоване на визначення того, як зміниться інтенсивність відмови програмної системи під час оцінки за допомогою традиційних ланцюгів Маркова безперервного часу, а також ланцюгів Маркова другого і третього порядку.

Розроблено метод визначення функції працездатності для Марковських моделей надійності ПЗ, який дає змогу в автоматизованому режимі визначати складну логічну функцію, що зменшує ймовірність внесення похибок і підвищує точність моделювання надійності.

На основі тестування розробленого методу автоматизованої побудови функції працездатності та побудови графа станів і переходів було виведено формули, які дозволяють обчислити мінімальну і максимальну кількості станів системи (також можна визначити кількість працездатних або станів простою) для системи із n елементів і r відновлень.

2.1. Метод автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами

Постійне зростання складності технічних систем, в яких важливу роль відіграє програмне забезпечення і їх широке використання в критичних з точки зору відмов сфер діяльності, підвищує вимоги надійності та безпечності таких систем загалом. Підвищення точності і достовірності аналізу надійнісних характеристик сучасних технічних систем вимагає використання моделей

надійності з високим ступенем адекватності. До таких моделей можна зарахувати моделі засновані на безперервних ланцюгах Маркова вищого порядку, які належать до класу архітектурних моделей. Використання таких моделей на практиці є дещо ускладнене через відсутність методів і алгоритмів, які дозволяли б розраховувати характеристики таких систем.

Тому, в даній роботі, було вирішено розробити метод подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами. Розроблений метод можна застосувати до Марковських процесів змінного порядку, використовуючи різні значення змінної, що відповідають порядку процесу для кожного стану. Цей метод за дією схожий на алгоритм Дейкстри, але його мета полягає не в тому, щоб знайти найкоротший шлях із заданої вершини, а в тому, щоб знайти всі можливі шляхи довжиною N - 1 для кожної вершини вхідного графа.

Основне завдання методу – це визначення станів системи і переходів між ними (у вигляді матриці суміжності, яка показує, як стани взаємодіють один з одним.), які враховують N-1 попередніх станів перебування для системи, що досліджується.

Вхідними даними для роботи методу є початковий граф, що відображає модулі та потоки взаємодії між ними (рис. 2.1) і N – порядок графа, який потрібно визначити за допомогою методу.

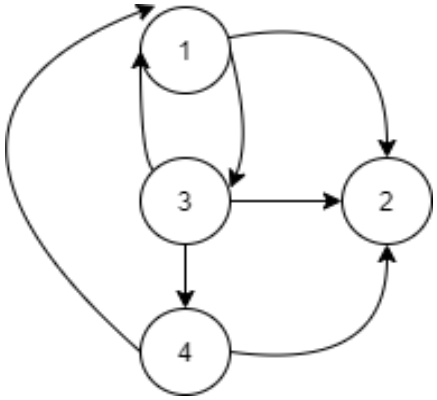


Рис. 2.1. Приклад зображення графа станів і переходів

Для подальшої роботи граф можна представити у вигляді матриці суміжності, яка є одним із зручних способів представлення графа у вигляді матриці.

$$M^1 = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \dots & \dots & \dots & \dots \\ r_{n1} & r_{n2} & \dots & r_{nn} \end{pmatrix},$$

де r_{ij} – може набувати значень 1 або 0 означає можливість переходу із стану *i* в стан *j* (1 – існує ребро, яке з'єднує вершину M_i і M, 0 – не існує); n – кількість модулів в системі.

Основні поняття, які потрібно розрізняти при аналізі роботи даного методу із теорії графів: *корінь* – верхній вузол в дереві ; *дитина* – вузол, безпосередньо приєднаний до іншого на шляху від кореня до зовнішнього вузла; *батько* – зворотне поняття до дитини; *предок* – вузол, досяжний послідовними переходами від дитини до батька; *глибина* – число ребер від кореневого вузла дерева до заданого; *лист (також зовнішній вузол)* - вузол, який не має дітей.

Робота розробленого методу складається із двох частин (кроків).

Крок 1. Спочатку метод послідовно опрацьовує всі вершини вхідного графа і визначає список вершин, які ведуть до цієї вершини за допомогою матриці суміжності. При наступній ітерації розглядаються вже кожна вершина зі списку вершин із попереднього кроку і визначається список вершин, які ведуть до цієї вершини. Даний процес продовжується N-1 кількість разів, поки не досягнемо потрібного порядку ланцюга Маркова.

В результаті роботи даного кроку буде отримано список із *k* (кількість вузлів у вхідному графі) дерев, кожне із яких в якості кореневого вузла міститиме вузол із початкового графа. Для кожного із вузлів дерева предками будуть вузли, які є попередніми станами відповідно до матриці суміжності і глибина кожного дерева становитиме N-1.

Крок 2. Наступним кроком для даного методу буде опрацювання і обхід списку дерев, який було отримано на кроці 1. Суть даного кроку полягає у визначенні станів еквівалентного графа для Марковського ланцюга N-го порядку. Для визначення станів потрібно пройтись по всіх гілках дерев (шлях від зовнішнього вузла дерева до кореневого вузла). Сукупність відповідних

унікальних шляхів і буду множиною станів еквівалентного графа для Марковського ланцюга N-го порядку.

На рисунку 2.2 наведено блок-схему методу подання процесу Маркова вищого порядку еквівалентним процесом першого порядку.

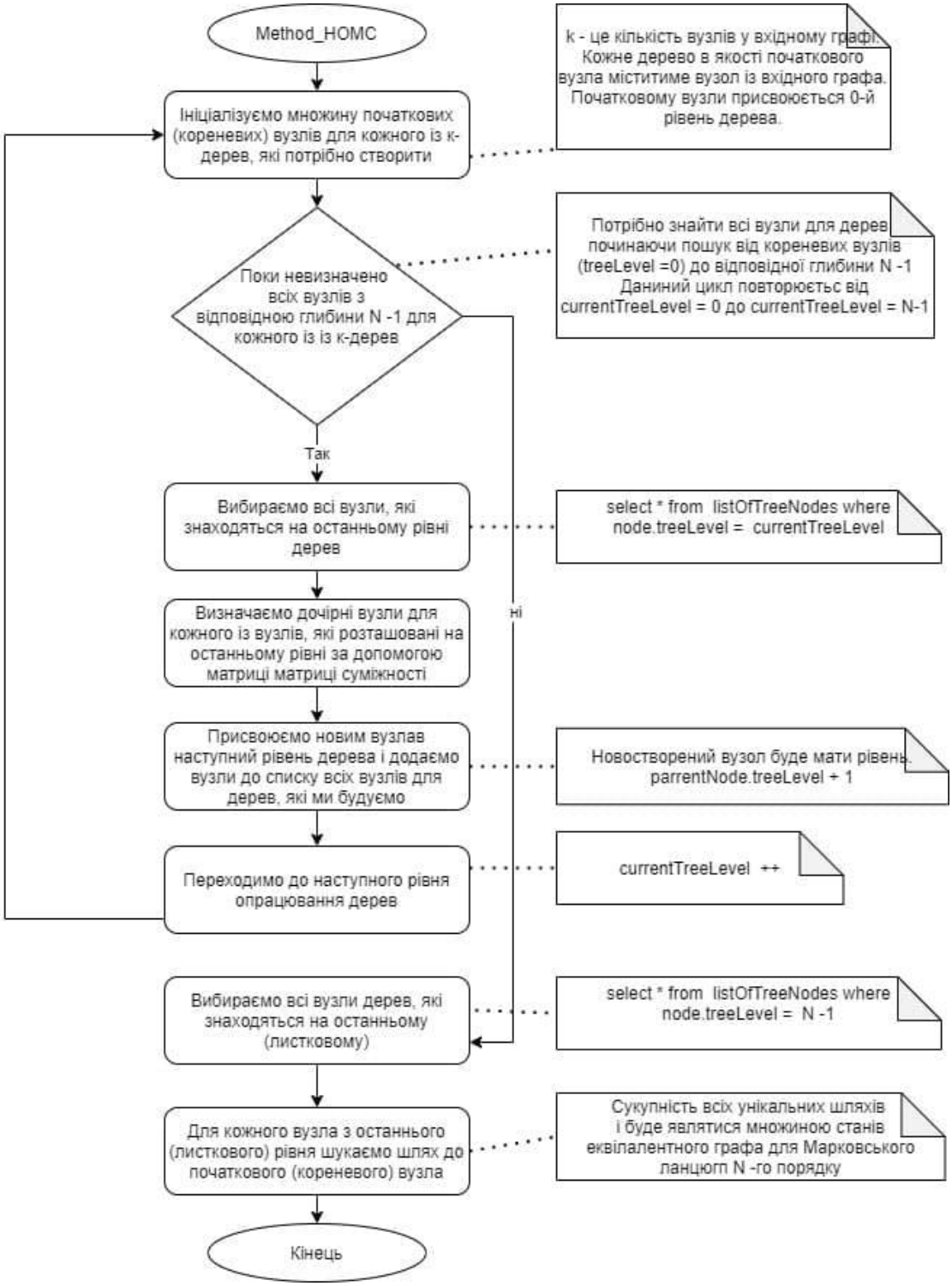


Рис. 2.2. Блок-схема розробленого методу автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами

Позначення:

- *N* – порядок графа, який визначається за допомогою методу.
- *k* – кількість вузлів вхідного графа.
- *inputGraphNodes* – це список вершин вхідного графа.
- *inputNodeId* – ідентифікатор вузла з вхідного графа.
- *adjacencyMatrix[k][k]* – матриця суміжності для вхідного графа.
- *listOfTreeNodes* – це список усіх вузлів для k (кількість вузлів у вхідному графі) дерев, кожна з яких міститиме вузол з початкового графа, як кореневий вузол.
- *treeLevel* – рівень дерева.
- *nodeChildrenList* – це список дочірніх вузлів для вузла дерева в структурі *TreeNode*.
- *HOMC_GraphNodes* – список вузлів еквівалентного графа для ланцюга Маркова N-порядку.

Відповідно до блок-схеми, роботу методу було описано у вигляді псевдокоду. Псевдокод для запропонованого методу подання процесу Маркова вищого порядку еквівалентним процесом першого порядку наведено нижче.

```
function Method_HOMC(inputGraphNodes,adjacencyMatrix, N):
foreach inputGraphNode in inputGraphNodes:
newTreeNode ← inputGraphNode
newTreeNode.treeLevel ← 0
  add newTreeNode to listOfTreeNodes
end foreach
// filling the listOfTreeNodes
currentTreeLevel  ←  0
while currentTreeLevel is not equal N-1:
  foreach node in (select * from listOfTreeNodes where treeLevel =
  currentTreeLevel):
  // find nodeChildrenList for node
  for indexI  ←  0 to indexI < k
       if adjacencyMatrix[indexI][node.inputNodeId]=1
```

```
        then
                    newTreeNode ← inputGraphNodes[node.initialNodeId]
         newTreeNode.treeLevel <- currentTreeLevel + 1
                  add newTreeNode to treeNode. nodeChildrenList
                    add newTreeNode to listOfTreeNodes
        end if
      end for
     end foreach
     currentTreeLevel ++
  end while
  foreach leafNode in (select * from listOfTreeNodes where treeLevel = N -
  1):
     HOMC_GraphNodes ← add the path from the leafNode node to the tree (root)
  node // this path is one of the states of the equivalent graph for the
  Markov chain of the nth order
  return HOMC_GraphNodes
```

Приклад використання розробленого методу наведено в додатку Б. Також розроблено програмне забезпечення для автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку (розділ 4.1).

2.2. Оцінка надійності програмного забезпечення польотів наносупутників CubeSat із використанням методу автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку

Для демонстрації практичного використання та перевірки розробленого методу автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку (у подальшому «метод») його було реалізовано та апробовано на прикладі оцінки надійності програмного забезпечення польотів наносупутників CubeSat [75].

Досліджувана система складається з таких компонентів [75]:

- Клієнтські модулі, що призначенні для виконання певних функціональних команд. Кожен модуль керує певною підсистемою, такою як пристрій або корисне навантаження;

- Репозиторії, які містять безпечні методи доступу (читання, запис) до всіх даних у системі;
- Модулі виклику команд та їх реалізація (Invoker і Receiver).

UML діаграма послідовності для програмного забезпечення CubeSat зображена на рисунку 2.3.

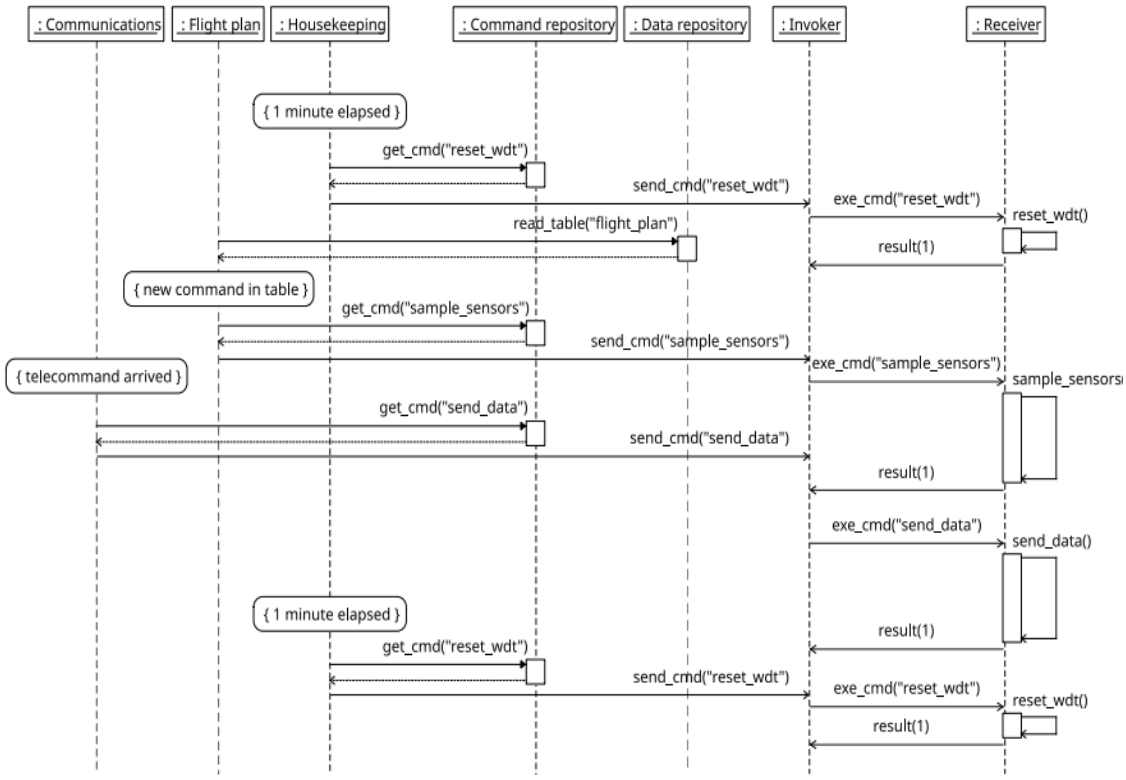


Рис. 2.3. Архітектура програмного забезпечення для польотів наносупутників CubeSat: діаграма послідовності UML.

Досліджувана система складається з таких компонентів [75]:

- Клієнтські модулі, що призначенні для виконання певних функціональних команд. Кожен модуль керує певною підсистемою, такою як пристрій або корисне навантаження;
- Репозиторії, які містять безпечні методи доступу (читання, запис) до всіх даних у системі;
- Модулі виклику команд та їх реалізація (Invoker і Receiver).

Кожен клієнтський модуль реалізує стратегію управління і може вимагати виконання команд за певних обставин. Для виконання команди клієнтський

модуль повинен створити її за допомогою сховища команд (command repository), а потім надіслати асинхронне повідомлення до модуля виклику команд (invoker). Модуль виклику команд (invoker) отримує всі повідомлення клієнта та організовує виконання, надсилаючи запит одержувачу. Приймач (receiver) фактично виконує команду, викликаючи відповідну функцію. Після того, як команда виконана, одержувач знову відправляє повідомлення із результатом виконання. Слід звернути увагу, що клієнти надсилають запити на виконання асинхронно і не мають прямого відгуку про результат виконання.

У досліджуваному програмному забезпеченні є три клієнтські модулі:

- Communications: отримує і аналізує команди із наземної станції для генерації відповідних системних команд;
- Flight Plan: планує розклад команд для польоту, план місії може динамічно змінюватись, коли супутник перебуває на орбіті;
- Housekeeping: сюди включені всі заходи, пов’язані з контролем стану супутника та здоров’я.

Існує два репозиторії у програмного забезпечення для польотів: репозиторій виконання команд (Command repository:) і сховище даних (Data repository). Репозиторій команд використовують для зберігання та надання доступу до команд, можливих у системі. Клієнти використовують цей репозиторій для створення нової команди з певними параметрами. Репозиторій даних надає доступ до загальних даних, таких як результати корисного навантаження, реєстрація системи, розклад рейсів.

Для даного програмного забезпечення необхідно описати потік управління програмною системою та встановити інтенсивність відмов кожного компонента, щоб оцінити його надійність за моделлю (1.1). Відповідно до (1.1) інтенсивність відмов системи визначається сумою інтенсивності відмов її модулів, зваженої на ймовірність виконання кожного модуля. Потік керування програмною системою зручно проілюструвати графом відповідних станів системи та переходів між ними. На основі такого графа можна побудувати систему рівнянь Колмогорова –

Чепмена (1.2). Імовірність виконання кожного модуля отримуємо , розв’язавши систему рівнянь (1.2).

Діаграма послідовності UML, представлена на рисунку 2.3, показує, як кожен модуль системи взаємодіє з іншими, тому цю діаграму можна легко перетворити, як граф станів і переходів. Відповідний граф потоку управління для програмного забезпечення польотів наносупутників CubeSat наведено на рисунку 2.4. Граф станів і переходів є орієнтованим графом,, вершинами якого є стани системи (програмні модулі), а ребра описують потік керування програмою. Назви програмних модулів на рис. 2.4. відповідають таким скороченням: Репозиторій команд (CR); Репозиторій даних (DR); Комунікації (C); План польоту (FP); ведення домашнього господарства (H); Інвокер (I); Приймач (R).

Дане дослідження спрямоване на визначення того, як зміниться інтенсивність відмови програмної системи під час оцінки за допомогою традиційних ланцюгів Маркова безперервного часу, а також ланцюгів Маркова другого і третього порядку. Інтенсивність відмов модулів зберігається близькою до зникнення їх впливу. Загальна стратегія моделювання була такою: якщо стан можна розбити на кілька віртуальних станів, ми встановлюємо дещо різну інтенсивність відмов для кожного віртуального стану, щоб побудувати моделі другого та третього порядку. Результати, отримані за допомогою моделей вищого порядку, порівнюються з результатами, отриманими за допомогою двох моделей першого порядку: одна з меншим значенням інтенсивності відмови, а інша з більшим. Дві моделі першого порядку використовуються для перевірки, чи знаходиться результат моделей вищого порядку між цими двома моделями першого порядку. Розглянемо, наприклад, стан FP (рис. 2.7.) розбитий на два віртуальні стани FPcr і FPdr, що походять із станів CR і DR відповідно. Потім ми встановлюємо $\lambda\_FPcr = 1{,}0 \cdot 10\text{-}6$ год-1 і $\lambda\_FPdr = 8{,}5 \cdot 10\text{-}5$ год-1 для моделі другого порядку і використовуємо ці значення інтенсивності відмови як значення λ_FP для двох відповідних моделей першого порядку.

Початковий граф станів і переходів, який був побудований на основі UML діаграми послідовності, та матриця суміжності для графа продемонстровані на рисунку 2.4 і в таблиці 2.1 відповідно.

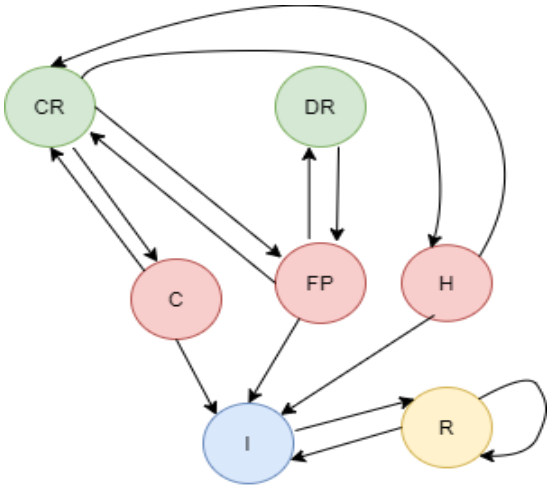


Рис. 2.4. Граф станів і переходів для програмного забезпечення польотів наносупутників.

Таблиця 2.1 – Матриця суміжності для графа рис. 2.4.

i/j	C	FP	H	CR	DR	I	R
C	0	0	0	1	0	1	0
FP	0	0	0	1	1	1	0
H	0	0	0	1	0	1	0
CR	1	1	1	0	0	0	0
DR	0	1	0	0	0	0	0
I	0	0	0	0	0	0	1
R	0	0	0	0	0	1	1

Програмна реалізація методу була використана для побудови еквівалентних ланцюгів Маркова другого та третього порядку. Значення порядку ланцюга Маркова (N) та матриця суміжності для початкового графа програмної системи є вхідними для методу. Досліджувана програмна система складається з 7 модулів (вершин графу) та 14 ребер, що представляють потік керування програмою.

На рисунках 2.5 і 2.6 показано візуалізацію першого етапу алгоритму для ланцюгів Маркова другого і третього порядку відповідно. Кожна фігура містить N дерев (N – кількість початкових вершин графу, тобто програмних модулів). Корінь кожного дерева є однією з початкових вершин графа. Гілки дерев – це єдині шляхи глибин, що відповідають порядку ланцюга Маркова на початковому

графі, що ведуть від гілок до відповідної кореневої вершини. Ці гілки визначають набір станів для еквівалентного графа першого порядку.

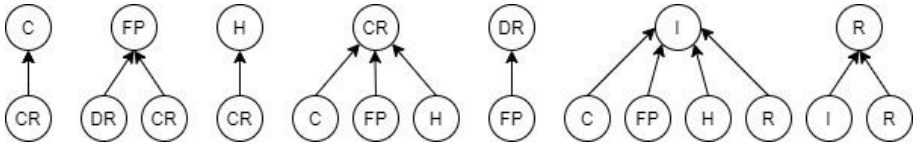


Рис. 2.5. Візуалізація визначення множини станів еквівалентного графа для ланцюга Маркова другого порядку.

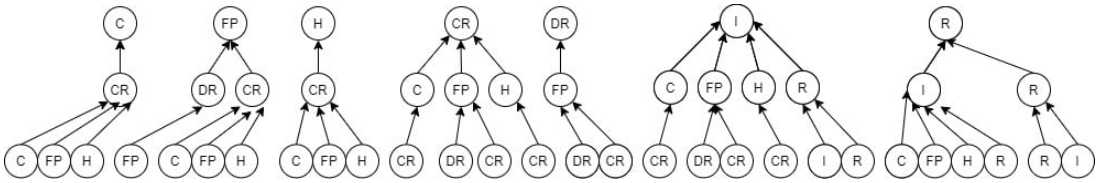


Рис. 2.6. Візуалізація визначення множини станів еквівалентного графа для ланцюга Маркова третього порядку.

Еквівалентний граф для ланцюга Маркова другого порядку складається з 14 станів і 28 переходів між станами (рис. 2.7 і табл. 2.2.)

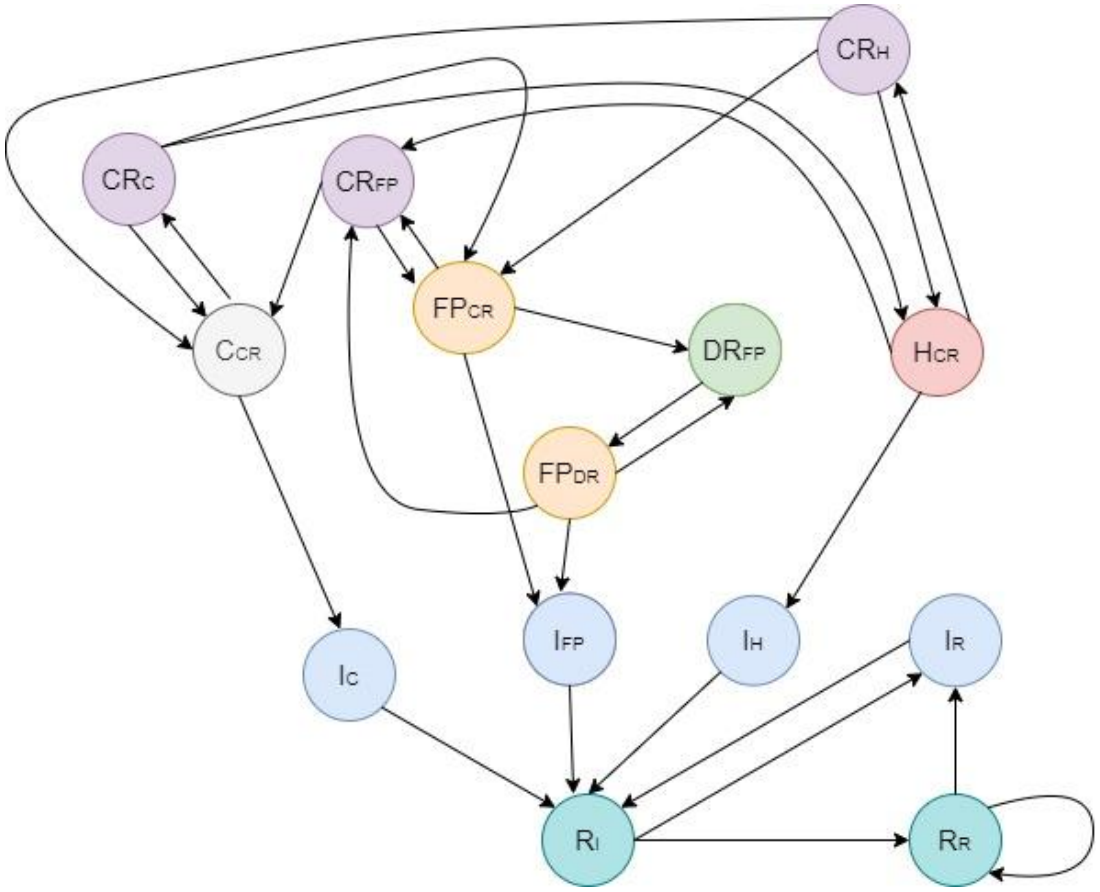


Рис. 2.7. Еквівалентний граф станів і переходів Марковського ланцюга 2-го порядку досліджуваної системи

Таблиця 2.2 – Матриця суміжності для графа рис. 2.7.

i/j	C_{CR}	FP_{CR}	FP_{DR}	H_{CR}	CR_C	CR_{FP}	CR_H	DR_{FP}	I_C	I_{FP}	I_H	I_R	R_I	R_R
C_{CR}	0	0	0	0	1	0	0	0	1	0	0	0	0	0
FP_{CR}	0	0	0	0	0	1	0	1	0	1	0	0	0	0
FP_{DR}	0	0	0	0	0	1	0	1	0	1	0	0	0	0
H_{CR}	0	0	0	0	0	0	1	0	0	0	1	0	0	0
CR_C	1	1	0	1	0	0	0	0	0	0	0	0	0	0
CR_{FP}	1	1	0	1	0	0	0	0	0	0	0	0	0	0
CR_H	1	1	0	1	0	0	0	0	0	0	0	0	0	0
DR_{FP}	0	0	1	0	0	0	0	0	0	0	0	0	0	0
I_C	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I_{FP}	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I_H	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I_R	0	0	0	0	0	0	0	0	0	0	0	0	1	0
R_I	0	0	0	0	0	0	0	0	0	0	0	1	0	1
R_R	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Еквівалентний граф для ланцюга Маркова третього порядку складається з 28 станів і 128 переходів. Матриця суміжності продемонстрована у таблиці 2.3.

Таблиця 2.3 – Матриця суміжності для еквівалентного графа Марковського ланцюга 3-го порядку досліджуваної системи

i/j	$C_{CR/C}$	$C_{CR/FP}$	$C_{CR/H}$	$FP_{DR/FP}$	$FP_{CR/C}$	$FP_{CR/FP}$	$FP_{CR/H}$	$H_{CR/C}$	$H_{CR/FP}$	$H_{CR/H}$	$CR_{C/CR}$	$CR_{FP/CR}$	$CR_{FP/DR}$	$CR_{H/CR}$	$DR_{FP/CR}$	$DR_{FP/DR}$	$I_{C/CR}$	$I_{FP/CR}$	$I_{FP/DR}$	$I_{H/CR}$	$I_{R/I}$	$I_{R/R}$	$R_{I/R}$	$R_{I/C}$	$R_{I/FP}$	$R_{I/H}$	$R_{R/R}$	$R_{R/I}$
$C_{CR/C}$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$C_{CR/FP}$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$C_{CR/H}$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
$FP_{DR/FP}$	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
$FP_{CR/C}$	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
$FP_{CR/FP}$	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
$FP_{CR/H}$	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
$H_{CR/C}$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
$H_{CR/FP}$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
$H_{CR/H}$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
$CR_{C/CR}$	1	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$CR_{FP/CR}$	1	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$CR_{FP/DR}$	1	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$CR_{H/CR}$	1	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$DR_{FP/CR}$	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$DR_{FP/DR}$	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$I_{C/CR}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
$I_{FP/CR}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
$I_{FP/DR}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
$I_{H/CR}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
$I_{R/I}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
$I_{R/R}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
$R_{I/R}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
$R_{I/C}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
$R_{I/FP}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
$R_{I/H}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
$R_{R/R}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
$R_{R/I}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1

Використовуючи еквівалентну матрицю суміжності ланцюга Маркова першого порядку, ймовірності $p_i(t)$ перебування в i-му стані можна обчислити за допомогою рівняння Колмогорова-Чепмена (1.2).

У цьому дослідженні, щоб виключити вплив інтенсивності переходів і виявити лише вплив ланцюгів Маркова вищого порядку, було використано однакові значення для всіх переходів потоку керування програмним

забезпеченням для польоту наносупутників. Було проведено три раунди експериментів з різними значеннями інтенсивності переходів.

Далі, використовуючи отримані залежності $p_i(t)$ та значення інтенсивності відмов для кожного модуля, за формулою (1.1) розраховували інтенсивність відмови усього програмного забезпечення. Як було описано раніше, інтенсивність відмови програмного забезпечення для польоту наносупутників була розрахована для двох традиційних ланцюгів Маркова першого порядку та моделей другого і третього порядку. Інтенсивність відмов для програмних модулів встановлювалася в діапазоні 10^{-5}–10^{-6} год$^{-1}$, що відповідає фактичним значенням, необхідним для промислового програмного забезпечення. Для кожної використаної інтенсивності переходу ми провели набір експериментів з різними значеннями інтенсивності відмов модулів, щоб побудувати набір кривих готовності системи та дослідити вплив подальшої надійності програмного модуля на надійність усієї програмної системи.

Вхідні дані для типового моделювання з набору наведені в таблицях 2.4. і 2.5. Це моделювання було проведено з a_{ij} = 0,0012 год$^{-1}$. Для побудови традиційного ланцюга Маркова першого порядку була використана таблиця 2.4, а таблиця 2.5 – для ланцюгів Маркова другого і третього порядку відповідно.

Таблиця 2.4 – Інтенсивність відмов для програмних модулів.

Модуль	Інтенсивність відмов, год$^{-1}$ (Перший запуск)	Інтенсивність відмов, год$^{-1}$ (Другий запуск)
C	$1.50 \cdot 10^{-5}$	$1.50 \cdot 10^{-5}$
FP	$1.00 \cdot 10^{-6}$	$8.50 \cdot 10^{-5}$
H	$5.00 \cdot 10^{-5}$	$5.00 \cdot 10^{-5}$
CR	$1.00 \cdot 10^{-5}$	$1.25 \cdot 10^{-5}$
DR	$2.00 \cdot 10^{-5}$	$2.00 \cdot 10^{-5}$
I	$2.00 \cdot 10^{-6}$	$1.00 \cdot 10^{-5}$
R	$5.00 \cdot 10^{-6}$	$6.50 \cdot 10^{-6}$

Таблиця 2.5 – Інтенсивність відмов програмних станів ПЗ для польоту наносупутників та віртуальних станів, що впроваджені моделями другого та третього порядку.

Модуль (модель другого порядку)	Інтенсивність відмов, год$^{-1}$	Модуль (модель третього порядку)	Інтенсивність відмов, год$^{-1}$
C_{CR}	$1.50 \cdot 10^{-5}$	$C_{CR\text{-}C}$	$1.50 \cdot 10^{-5}$
FP_{CR}	$1.00 \cdot 10^{-6}$	$C_{CR\text{-}FP}$	$2.00 \cdot 10^{-5}$
FP_{DR}	$8.50 \cdot 10^{-5}$	$C_{CR\text{-}H}$	$2.50 \cdot 10^{-5}$
H_{CR}	$5.00 \cdot 10^{-5}$	$FP_{CR\text{-}C}$	$1.00 \cdot 10^{-6}$
CR_C	$1.00 \cdot 10^{-5}$	$FP_{CR\text{-}FP}$	$3.00 \cdot 10^{-6}$
CR_{FP}	$1.20 \cdot 10^{-5}$	$FP_{CR\text{-}H}$	$5.00 \cdot 10^{-6}$
CR_H	$1.25 \cdot 10^{-5}$	$FP_{DR\text{-}FP}$	$8.50 \cdot 10^{-5}$
DR_{FP}	$2.00 \cdot 10^{-5}$	$H_{CR\text{-}C}$	$5.00 \cdot 10^{-5}$
I_C	$2.00 \cdot 10^{-6}$	$H_{CR\text{-}FP}$	$5.50 \cdot 10^{-5}$
I_{FP}	$1.00 \cdot 10^{-6}$	$H_{CR\text{-}H}$	$6.00 \cdot 10^{-5}$
I_H	$9.00 \cdot 10^{-5}$	$CR_{C\text{-}CR}$	$1.00 \cdot 10^{-5}$
I_R	$9.50 \cdot 10^{-5}$	$CR_{FP\text{-}CR}$	$1.20 \cdot 10^{-5}$
R_I	$5.00 \cdot 10^{-6}$	$CR_{FP\text{-}DR}$	$1.25 \cdot 10^{-5}$
R_R	$6.50 \cdot 10^{-6}$	$CR_{H\text{-}CR}$	$1.30 \cdot 10^{-5}$
–	–	$DR_{FP\text{-}CR}$	$2.00 \cdot 10^{-5}$
–	–	$DR_{FP\text{-}DR}$	$3.00 \cdot 10^{-5}$
–	–	$I_{C\text{-}CR}$	$2.00 \cdot 10^{-6}$
–	–	$I_{FP\text{-}CR}$	$3.00 \cdot 10^{-6}$
–	–	$I_{FP\text{-}DR}$	$4.00 \cdot 10^{-6}$
–	–	$I_{H\text{-}CR}$	$5.00 \cdot 10^{-6}$
–	–	$I_{R\text{-}I}$	$7.00 \cdot 10^{-6}$

–	–	$I_{R\text{-}R}$	$9.00 \cdot 10^{-5}$
–	–	$R_{I\text{-}C}$	$5.00 \cdot 10^{-6}$
–	–	$R_{I\text{-}FP}$	$5.50 \cdot 10^{-6}$
–	–	$R_{I\text{-}H}$	$6.00 \cdot 10^{-6}$
–	–	$R_{I\text{-}R}$	$6.20 \cdot 10^{-6}$
–	–	$R_{R\text{-}I}$	$6.30 \cdot 10^{-6}$
–	–	$R_{R\text{-}R}$	$6.50 \cdot 10^{-6}$

Часові залежності інтенсивності відмов програмного забезпечення, отримані в результаті цього моделювання, наведені на рисунку 2.8. Як показано на цьому рисунку, криві, розраховані за допомогою моделей другого та третього порядку, не є сумою чи усередненням кривих першого порядку (найменші та найбільші значення частоти відмов, див. табл. 2.4). Більше того, поведінка надійності з часом змінюється. Крива частоти відмов має мінімум, що може вплинути на продуктивність системи. Значення частоти відмов при t = 1000 годин відрізняється до 50% для моделей вищого та першого порядку. При цьому різниця між результатами, отриманими з моделями другого і третього порядку, становить близько 10%.

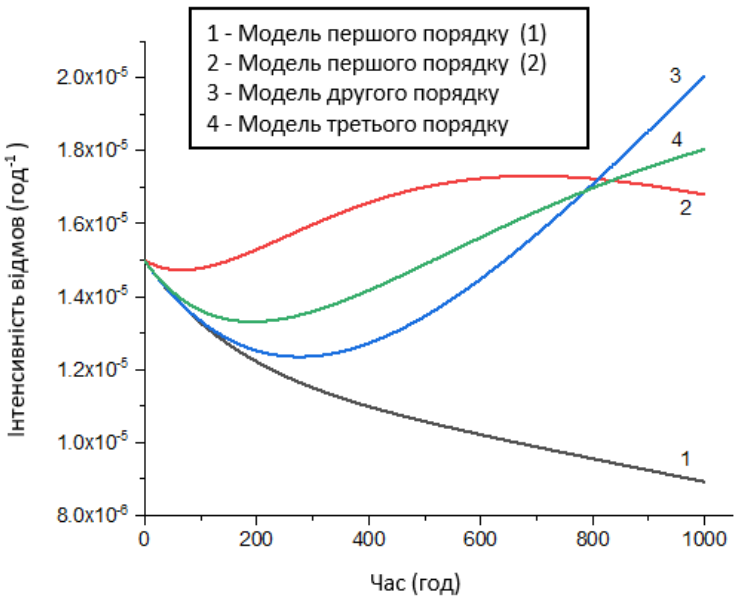


Рис. 2.8. Часові залежності частоти відмов програмного забезпечення CubeSat , змодельовані різними моделями.

Використання традиційних ланцюгів Маркова першого порядку може призвести до значної похибки в оцінці рівня відмови програмної системи – до 50% за інших рівних умов. Тоді як використання моделі другого порядку у порівнянні з моделлю третього порядку дає похибку оцінки не більше 10%. Крім того, різниця в результатах, отриманих за допомогою ланцюгів Маркова вищого порядку, не змінює форму кривої відмов з часом, а лише її значення. Таким чином, можна зробити висновок, що використання ЛМВП дає змогу значно підвищити точність оцінки надійності складних програмних систем як за рівнем відмов, так і за критерієм залежності від часу.

2.3. Метод автоматизованого визначення функції працездатності для Марковських моделей надійності ПЗ

Загальний підхід до формування Марковських моделей надійності систем полягає у тому, що ці моделі у формалізованому вигляді описують з позиції надійності взаємодію елементів системи у процесі функціонування і показують ступінь впливу надійності окремих елементів на надійність системи загалом.

Для опису функціонування системи можна використовувати схему, яка дозволяє візуалізувати взаємодії між елементами технічної системи і може складатися із трьох наступних компонент:

- модуль (елемент) системи – представляє одну із робочих частин системи, яка виконує певні функції і взаємодіє для функціонування системи (це може бути апаратна або програмна частина);
- лінія – дозволяє встановлювати зв'язок між модулями (представляють потік передачі взаємодії між модулями) або вузлами;
- вузол – це проміжний елемент схеми (схеми можна створювати і без використання таких проміжних елементів, цей елемент було внесено у схему візуалізації для коректної роботи і реалізації методу, розробленого в даному розділі), який потрібен для виділення розгалужень в системі, зображення паралельних підсистем схеми.

На рисунку 2.9 показані приклади простих схем візуалізації взаємодії елементів технічних систем для певних систем.

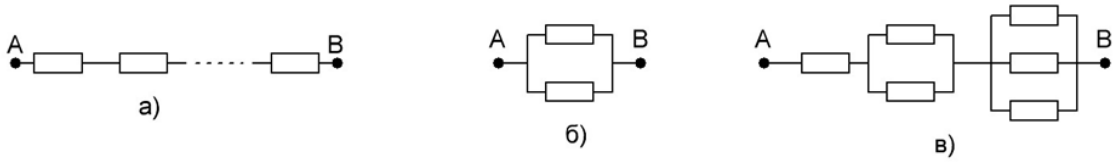


Рис. 2.9. Приклади простих схем візуалізації взаємодії елементів технічних систем

Схеми на рисунку 2.9 є простими і складаються із декількох модулів, тому функцію працездатності можна визначити наочно: система а) працює, якщо усі елементи є в працездатному стані (система із послідовним з'єднанням елементів); система б) є в працездатному стані, якщо хоча б один елемент системи є у працездатному стані (система із паралельним з'єднанням елементів); система в) працездатна, якщо перший елемент робочий, один із двох елементів в другій групі розгалуження є в працездатному стані і будь-який із трьох у третій групі розгалуження елементів.

Функцію працездатності системи можна описувати за допомогою функцій алгебри логіки під час створення графічного зображення взаємодії її елементів. Функція працездатності – це логічний вираз, який дозволяє визначити працездатність системи на основі інформації про працездатність певних елементів системи. Суть цього способу полягає у тому, що потрібно побудувати логічну функцію/умову, де при послідовному з'єднанні елементи системи об'єднуються логічною операцією AND, а при паралельному з'єднанні сегменти об'єднуються логічною операцією OR. У випадку складної системи це завдання суттєво ускладняється.

Побудову такої функції виконують переважно вручну і це вимагає високої кваліфікації розробника та глибокого розуміння системи, що проектується. Відповідно це збільшує імовірність внесення помилок в модель. Автоматизована побудова функції працездатності зменшує імовірність внесення помилок і підвищує достовірність моделі.

У роботі [76] було запропоновано рекурсивний алгоритм визначення функції відмови і працездатності. Він базується на визначенні послідовних і паралельних з’єднань із чітко визначеними межами розгалужених підсистем. Через свою рекурсивну природу даний метод накладає певні обмеження при визначенні функції працездатності і не придатний для визначення функції працездатності для схем певного типу системи (обмеження – всі вихідні лінії із вузла повинні входити в один кінцевий вузол), оскільки не володіє інформацією про повну топологію схеми, а лише будує функцію працездатності в процесі обходу схеми.

Відповідно, враховуючи обмеження алгоритму описаного у роботі [76], було розроблено метод для автоматизованого визначення функції працездатності для моделей надійності (рис. 2.10).

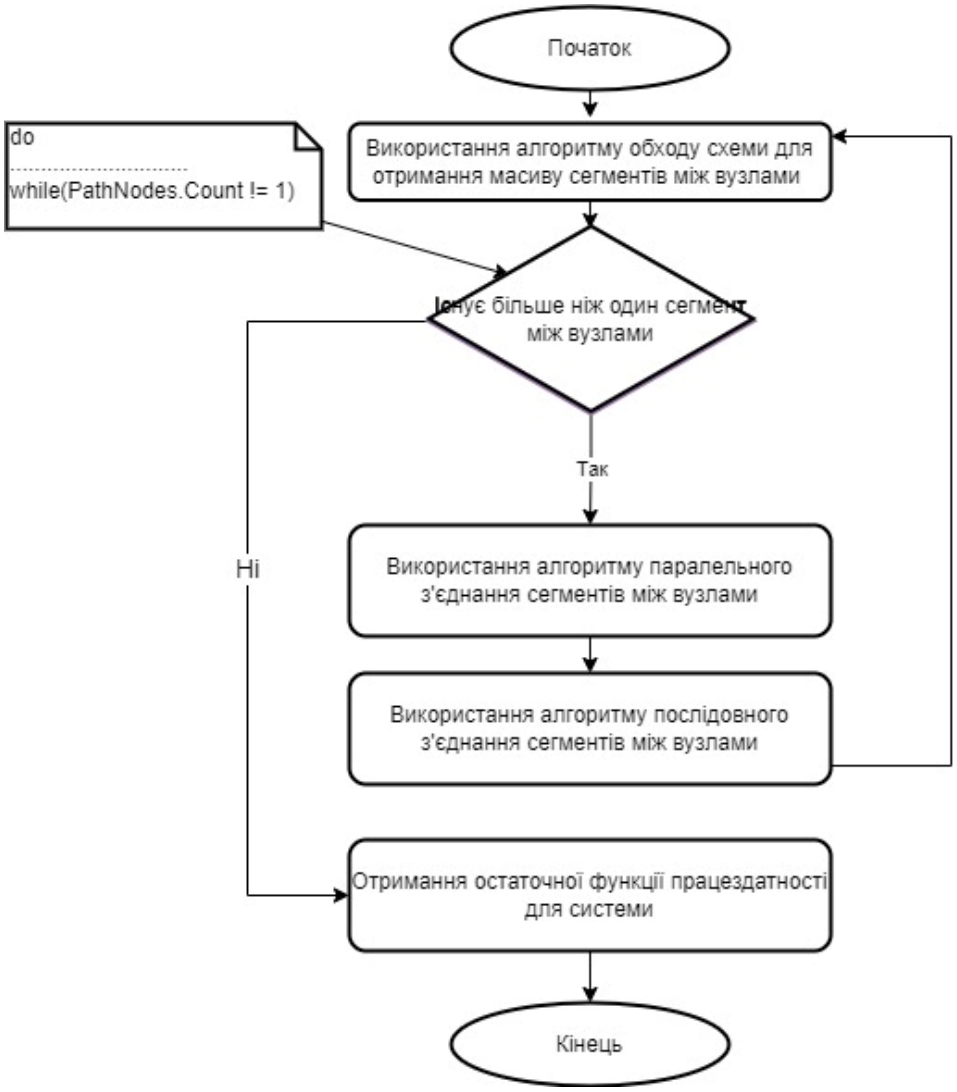


Рис. 2.10. Блок-схема методу визначення функції працездатності

Розроблений метод базується на попередньому повному аналізі схеми і складається із трьох складових частин :

- Алгоритму обходу схеми взаємодії елементів системи, який попередньо аналізує топологію схеми і виявляє масив сегментів схеми, кожен з яких є послідовним з'єднанням елементів (модулів) або одним елементом (сегмент – це певна частина структурної схеми, яка розташована між двома вузлами і містить послідовно з'єднані елементи між собою із відповідною частковою функцією працездатності для даного сегмента);
- Алгоритму послідовного з'єднання, який виявляє сегменти, котрі з'єднані між собою послідовно;
- Алгоритму паралельного з'єднання, який виявляє сегменти, котрі з'єднані між собою паралельно.

Даний метод передбачає виконання подальших кроків:

Крок 1. Попередній аналіз схеми і визначення масиву сегментів.

Першим етапом після опису схеми взаємодії елементів системи перед остаточним визначенням функції працездатності є загальний аналіз схеми і отримання інформації про елементи схеми і як вони взаємодіють між собою, визначення масиву сегментів схеми.

Структура сегмента складається із інформації про початковий і кінцевий вузли для даного сегмента, модулі з яких він складається, і відповідно, часткова функція працездатності для сегмента (рис. 2.11).

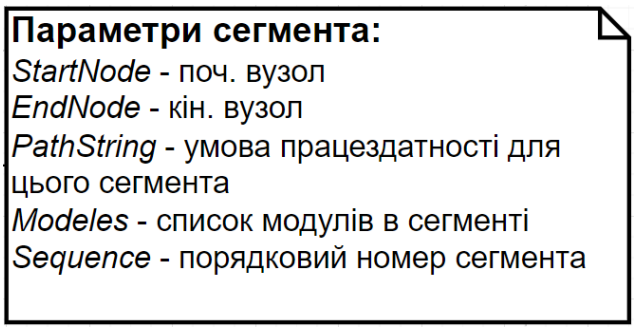


Рис. 2.11. Структура сегмента схеми взаємодії модулів системи

На рисунку 2.12 зображено блок-схему алгоритму обходу схеми взаємодії елементів системи і визначення масиву сегментів для схеми. Даний алгоритм

виконує аналіз схеми, починаючи від початкового вузла схеми до кінцевого, визначаючи типи елементів схеми і відповідно сегменти до яких вони належать.

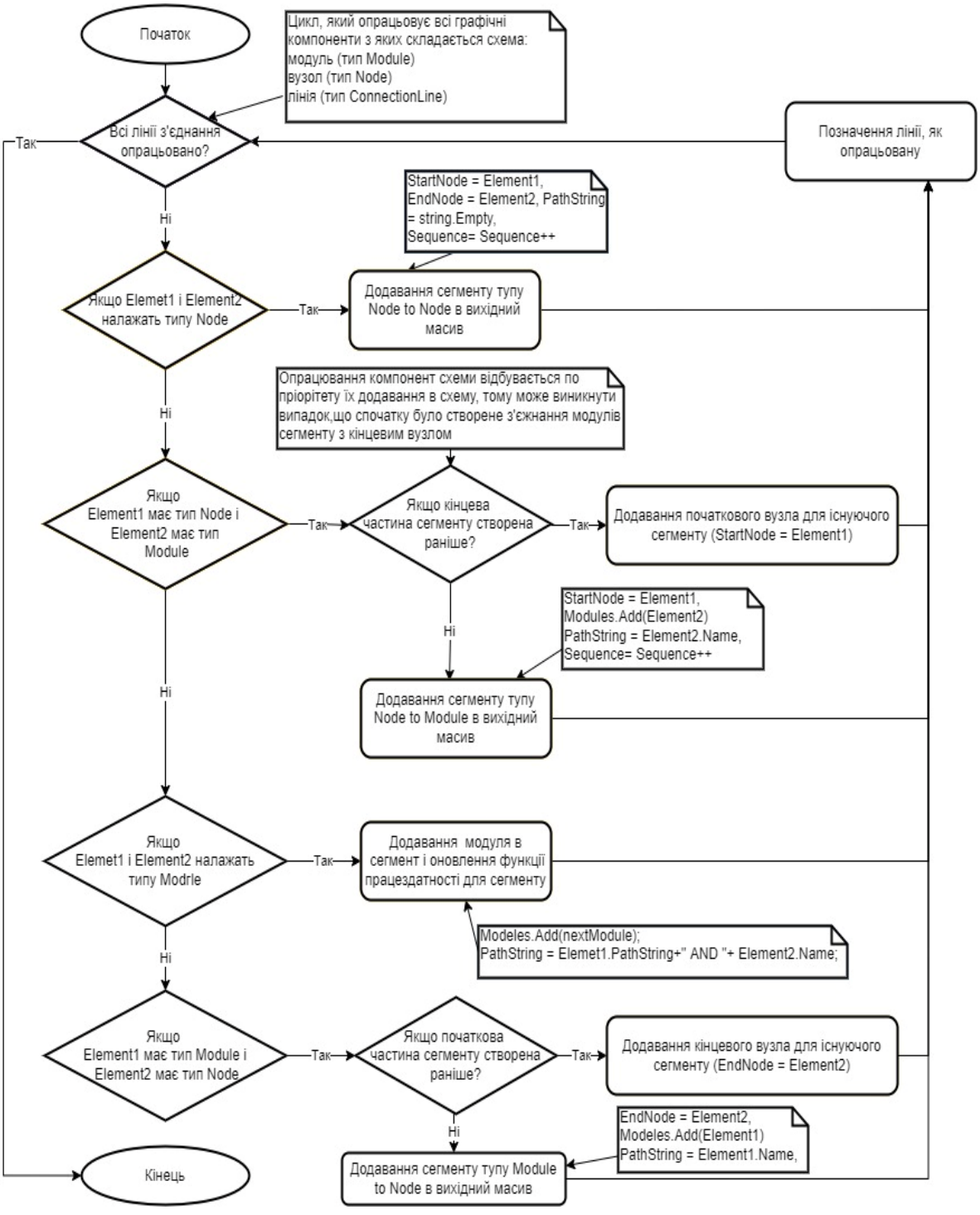


Рис. 2.12. Блок-схема алгоритму обходу схеми взаємодії елементів системи.

На рисунку 2.13 зображено візуалізацію взаємодії елементів для певної системи із 7-и елементів, для якої було наведено приклад масиву сегментів у таблиці 2.6.

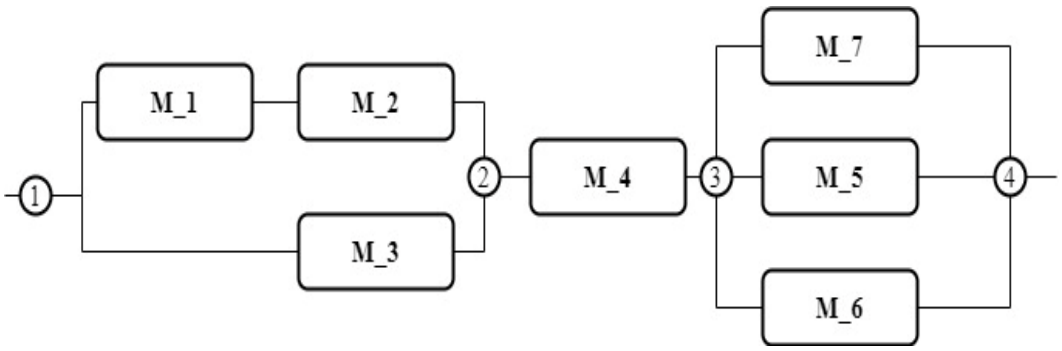


Рис. 2.13. Приклад схеми взаємодії модулів системи із 7-ти елементів

Таблиця 2.6 – Масив сегментів для схеми із 7-и елементів (рис. 2.13)

Sequence	StartNode	EndNodel	PathString
1	1	2	M_1 *AND* M_2
2	1	2	M_3
3	2	3	M_4
4	3	4	M_5
5	3	4	M_6
6	3	4	M_7

Крок 2. Опрацювання масиву сегментів і з'єднання сегментів із частковими функціями працездатності, які розташовані паралельно або послідовно в схемі. Даний крок передбачає опрацювання (прохід) всіх сегментів, які були визначені на попередньому кроці та об'єднання їх паралельно або послідовно (разом із частковими функціями працездатності), якщо вони маються залежності між початковими і кінцевими вузлами сегментів.

Крок 3. Визначення та об'єднання паралельно розташованих сегментів схеми.

Даний алгоритм перевіряє всі сегменти схеми і якщо виконується умова, що збігаються початковий (StartNode) і кінцевий вузол (EndNode) для декількох сегментів із загальної множини, ми об'єднуємо ці сегменти паралельно і відповідно перевизначаємо часткову функцію працездатності (функція працездатності сегментів об'єднується операцією OR (АБО)), в результаті з

декількох сегментів ми отримуємо один із новою частковою умовою працездатності.

На рисунку 2.14 зображено блок-схему алгоритму виявлення і з'єднання сегментів, які розташовані паралельно.

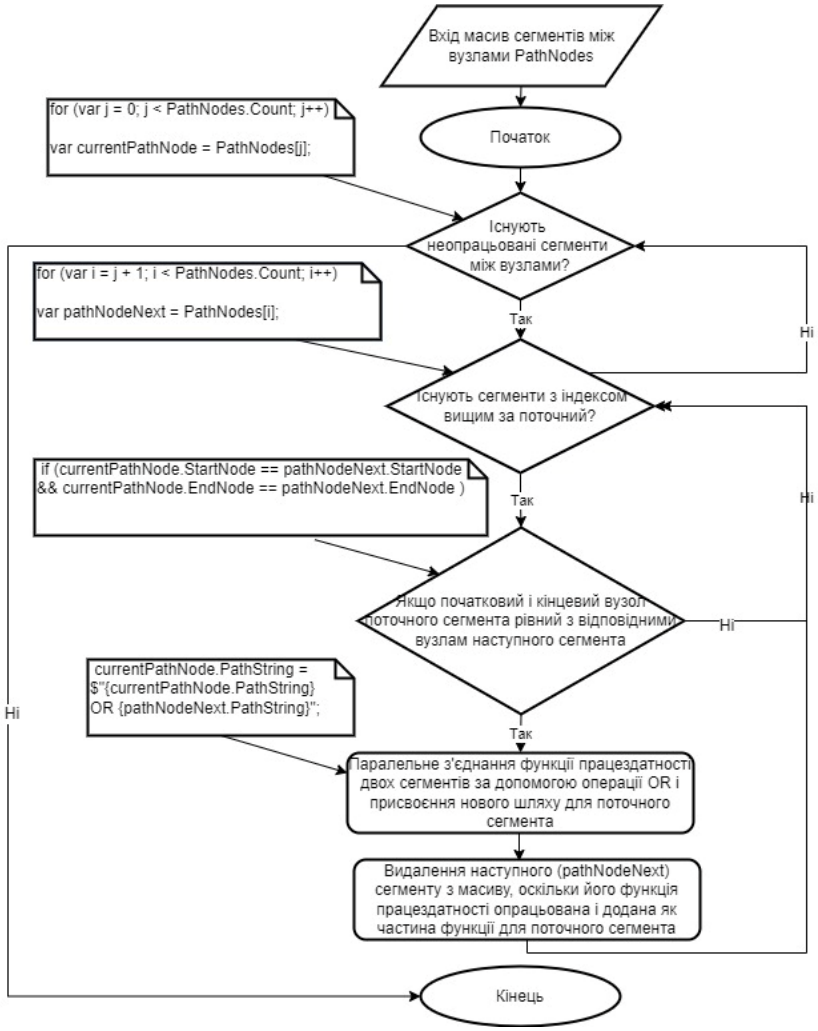


Рис. 2.14. Блок-схема алгоритму виявлення і з'єднання паралельних сегментів.

На рисунку 2.15 зображено частину схеми із 3-х елементів, яка складається із двох сегментів:

- StartNode: *1*; EndNodel: *2*; PathString: *M_1 AND M_2*
- StartNode: *1*; EndNodel: *2*; PathString: *M_3*

Для цих двох сегментів виконується умова, що збігаються початковий і кінцевий вузол, тому їх можна об'єднати паралельно, тоді в масиві сегментів залишиться один сегмент із перевизначеною частковою функцією працездатності:

- StartNode: *1*; EndNodel: *2*; PathString: (*M_1 AND M_2) OR M_3*

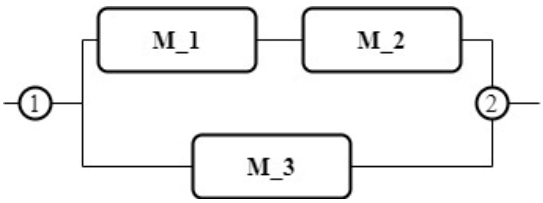


Рис. 2.15. Фрагмент схеми із 3-х елементів.

Крок 4. Визначення і об’єднання послідовно розташованих сегментів схеми.

На рисунку 2.16 зображено блок-схему алгоритму виявлення і з’єднання сегментів, які розташовані послідовно.

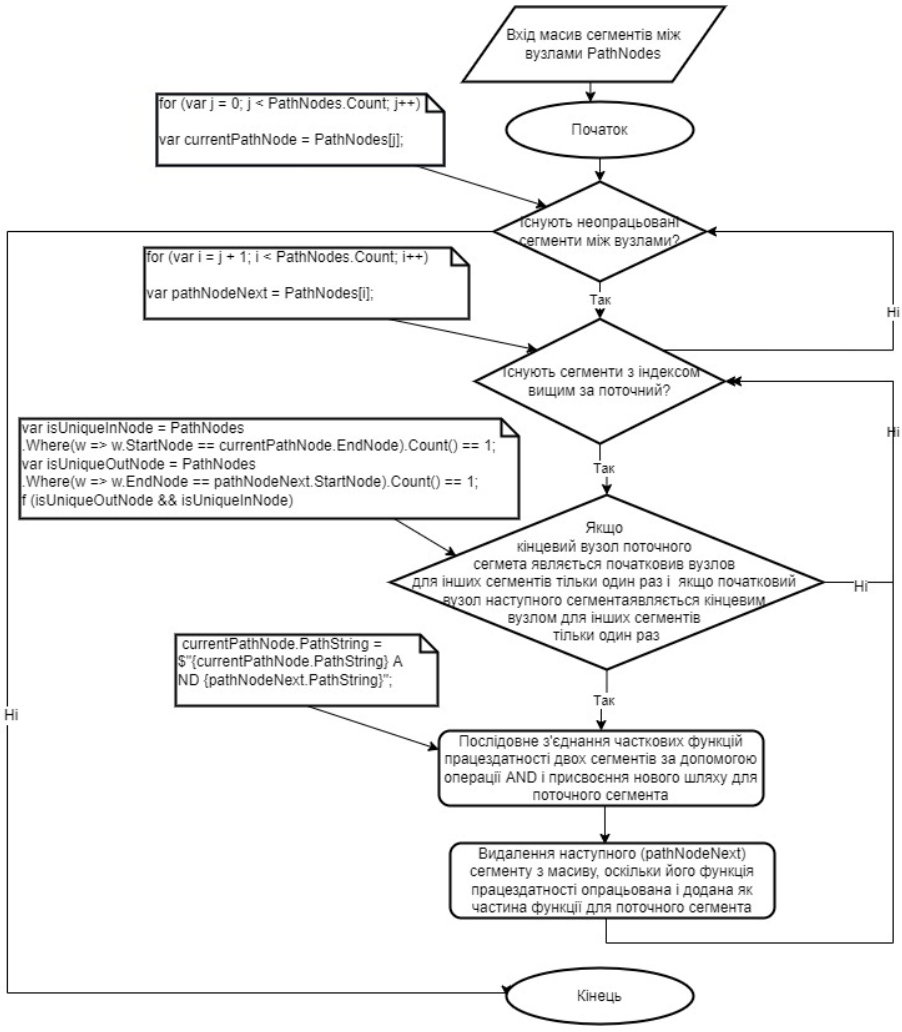


Рис. 2.16. Блок-схема алгоритму виявлення і з’єднання послідовних сегментів.

Після завершення кроку 3 і об’єднання всіх сегментів з масиву, які були розташовані паралельно, потрібно пройтись по всіх сегментах схеми і виявити та відповідно об’єднати сегменти з частковими функціями працездатності, які розташовані послідовно.

Якщо виконується умова, що в масиві сегментів є один єдиний сегмент у якого кінцевий вузол (EndNode) збігається із початковим вузлом (StartNode) іншого єдиного сегмента із множини всіх сегментів, то такі сегменти розташовані послідовно і їх потрібно з'єднати із використанням логічної операції AND (I).

На рисунку 2.17 зображено схему, яка є продовженням схеми, що на рисунку 2.15 і складається із 4-х елементів. Якщо враховувати попередній крок і паралельне об'єднання сегментів між вузлами 1 і 2, масив сегментів складається із двох сегментів:

- StartNode: *1*; EndNodel: *2*; PathString: (*M_1 AND M_2) OR M_3* – утворений після паралельного об'єднання сегментів.
- StartNode: *2*; EndNodel: *3*; PathString: *OR M_4*

Для цих двох сегментів виконується умова, що збігається кінцевий вузол першого сегмента із початковим другого, тому їх можна об'єднати послідовно, тоді в масиві сегментів залишиться один сегмент із перевизначеною частковою функцією працездатності і його значення StartNode (значення StartNode з першого сегмента) і EndNodel (значення EndNodel з другого сегмента) перевизначаться:

- StartNode: *1*; EndNodel: *3*; PathString: *((M_1 AND M_2) OR M_3)AND M_4*

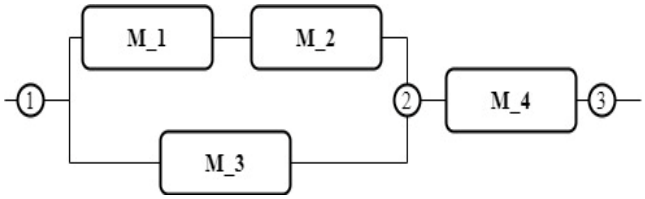


Рис. 2.17. Фрагмент схеми із 4-х елементів.

Крок 5. Перевірка отримання кінцевої функції працездатності

Відповідно кроки із паралельним і послідовним об'єднанням сегментів схеми можуть виконуватись декілька разів, оскільки спочатку виконується об'єднання паралельних сегментів. На цьому кроці утворюються сегменти, які задовольняють умову послідовного об'єднання на наступному кроці і так виконується циклічно допоки в масиві сегментів (початковий і кінцевий вузли сегмента будуть початковим і кінцевим вузлами всієї схеми) не залишиться один сегмент, функція працездатності якого буде повною функцією працездатності для всієї схеми, якщо існує більше ніж один сегмент в масиві повертаємося на Крок 2.

2.4. Визначення простору станів для графа станів і переходів із скінченною кількістю відновлень

В результаті роботи із функцією працездатності системи та графом станів і переходів, який будується на основі функції працездатності було проаналізовані і виведено формули, які базуються на загальних комбінаторних принципах і дозволяють обчислити мінімальну і максимальну кількості станів системи (також можна визначити кількість працездатних або станів простою) для системи із *n* елементів і *r* відновлень (кількість відновлень однакова для всіх модулів системи) і ці формули можна використовувати для швидкого аналізу простору станів при створенні ПЗ:

- **мінімальна кількість можливих станів системи із *n* елементів і *r* відновлень –** кількість станів при послідовному з'єднанні кожного із *n* елементів і *r* відновлень (формула 2.1.);
- **максимальна кількість можливих станів системи із *n* елементів і *r* відновлень –** кількість станів при паралельному з'єднанні кожного із *n* елементів і *r* відновлень (формула 2.2);
- **кількість можливих станів системи із *n* елементів і *r* відновлень для змішаного типу з'єднань елементів системи –** дане значення коливається між максимальним і мінімальним максимальним значенням, залежно від конфігурації з'єднань елементів в системі;

$$(2 * (r + 1) - 1)^n + 1 \quad (2.1)$$

$$(2 * (r + 1))^n , \quad (2.2)$$

де ***n*** **–** кількість елементів (модулів) в системі, ***r*** – кількість відновлень для кожного елемента в системі

Розглянемо детальніше виведення формул 2.1 і 2.2.

Нехай існує система із ***n*** елементів (модулів), які розташовані паралельно і кожен із елементів може відновитися ***r*** разів (***r*** **–** однакове для всіх модулів).

Стани функціонування даної системи можна показати за допомогою такого числа, де ***n*** – означає розрядність числа, а r_{index} **–** для кожного розряду, порядок відновлення модуля (скільки разів вже відновився модуль).

Також є важливою умовою, стани, які додаються у множину станів системи після відновлення і стани, які мають такі ж самі робочі елементи, як і стан після відновлення розрізняються, як два різні стани (наприклад стан $1_0\,1_0\,1_0\,1_0$ і стан $1_0\,1_1\,1_0\,1_0$ це різні стани в множині станів).

Отже, робочому стану модуля відповідає цифра 1, а стану відмови 0, але також потрібно зважати на індекс поточного відновлення ($1_0\,1_1\,1_2\;\;0_0\,0_1\,0_2\,...\,1_r\,0_r$), тому для визначення множини всіх можливих станів при паралельному з'єднанні (розглядаємо множину станів: робочі стани – один із елементів системи перебуває у робочому стані , стани простою – один або більше елементів у стані простою, а інші у стані відмови і один стан критичної відмови – всі елементи системи відмовили і не можуть бути більше відновлені) можна використати адаптовану під даний випадок комбінаторну формулу розміщення з повторенням:

$$A_n^m = n^m, \tag{2.3}$$

де

- ***n*** **–** кількість всіх можливих елементів для розміщення
- ***m*** – кількість елементів, що потрібно розмістити,

Для невідновлювальних систем використовуються значення 1 і 0, але для відновлювальних систем, значення можуть бути $1_0\,1_1\,1_2\;\;0_0\,0_1\,0_2\,...\,1_r\,0_r$, тобто при збільшенні числа відновлення для системи на 1 (r = r + 1) у нас множина елементів для розміщення збільшується на 2, оскільки додаються значення 1_{r+1} і $0_{r+1.}$ Тому формула (2.2) набуває такого вигляду:

$A_n^m = n^m = \;(2 * (r+1))^n$ – загальна кількість станів для системи із паралельно розташованими елементами, множина складається із працездатних станів, станів простою і стан критичної відмови.

Також із даної формули 2.2 можна вивести формули, для конкретних підмножин робочих станів і станів простою, критичної відмови (це завжди один стан із загальної множини).

Стан критичної відмови для системи із **n** паралельним розміщенням елементів з **r** відновленням виглядає так: $0_r\,0_r\,0_r\,...\;\;0_r$.

Як вже було вище сказано, стан простою – один або більше елементів у стані простою, а інші у стані відмови. Тому із множини $\{0_0\ 0_1\ 0_2\ \ldots\ \ 0_r\}$ми можемо виконати розміщення із повторенням, яке буде містити всі стани простою і один стан критичної відмови, тому формула для визначення кількості станів простою для системи із **n** паралельних елементів з **r** відновленням буде мати вигляд:

$$(r+1)^n - 1 \qquad (2.4)$$

Тоді для визначення кількості працездатних станів, потрібно від загальної кількості відняти стани простою і стан критичної відмови:

$$(2*(r+1))^n - ((r+1)^n - 1) - 1 = 2^n \cdot (r+1)^n - (r+1)^n + 1 - 1 = (2^n - 1)\cdot(r+1)^n \quad (2.5)$$

Наведемо приклад. Дано систему із 3-х (**n** = 3) паралельно розташованих елементів із значенням відновлення 1 (**r** = 1).

Тоді множина $\{1_0\ 1_1\ \ 0_0\ \ 0_1\}$ - всіх можливих елементів для розміщення, а оскільки в системи **n** = 3 – кількість елементів, що потрібно розмістити, тому кількість станів для даної системи дорівнює:

$$A_n^m = n^m = \ (2*(r+1))^n = \ (2*(1+1))^3 = 64\ ,$$

Це значення – максимальне значення можливих станів для паралельної системи із **n** = 3 і **r** = 1, для систем із такими ж значеннями **n** і **r**, але із послідовним або змішаним з'єднанням елементів – множина станів буде меншою, оскільки це значення максимальне при паралельному з'єднанні всіх елементів системи.

Робочі стани системи визначаються за формулою 2.5:

$(2*(r+1))^n - (r+1)^n - 2\ = (2*2)^3 - 2^n - 2 = 64 - 8 - 2 = 56$ – робочі стани із 64 станів,

$(r+1)^n - 1 = \ (1+1)^3 - 1 = 8 - 1 = 7$ – стани простою із 64 станів.

Для системи із послідовним розташуванням елементів, формула для визначення множити станів, можна визначити наступним чином.

Нехай існує система із ***n*** елементів (модулів), які розташовані послідовно і кожен із елементів може відновитися ***r*** разів (***r*** **–** однакове для кожного модуля).

Важлива умова для системи із послідовним розташуванням елементів – розглядаються такі елементи із множини всіх станів: робочі стани (всі елементи

системи перебувають у стані працездатності), стани простою (один або більше елементів у стані простою, а інші у стані працездатності), стан критичної відмови (один елемент системи відмовив і більше не може бути відновлений). Тобто для розміщення із повторенням, яке також буде використане для виведення формули визначення множини станів для послідовно з'єднаних елементів системи, множина всіх можливих елементів для розміщення $\{1_0\,1_1\,1_2\;\;0_0\,0_1\,0_2\;...\;1_r\,0_r\}$ буде на 1-ю менша, оскільки ми не розглядаємо значення $\{0_{r\ index\ =\ r\ max}\}$. $\{1_0\,1_1\,1_2\;\;0_0\,0_1\,0_2\;...\;1_r\}$ із даної множини ми зможемо визначити розміщення для всіх робочих станів і станів простою, і система матиме 1 стан критичної відмови, тому формула (2.1) виглядатиме так:

$$A_n^m = n^m = \;\; (2 * (r+1) - 1)^n + 1$$

Також із даної формули 2.1 можна вивести формули, для конкретних підмножин робочих станів і станів простою, критичної відмови (це завжди один стан із загальної множини).

Робочі стани для системи із **n** послідовним розміщенням елементів з **r** відновленням будуть складатись із розміщення з повторенням з множини значень $\{1_0\;1_1\;1_2\;...\;\;1_r\}$, тому формула для визначення кількості працездатних станів для системи із послідовним розміщенням елементів виглядатиме наступним чином:

$$(r+1)^n \qquad (2.6)$$

За формулою $(2 * (r+1) - 1)^n$ ми визначаємо сукупність робочих станів і станів простою для системи , тому формула для визначення кількості станів простою для системи із послідовним розміщенням елементів виглядатиме так:

$$(2 * (r+1) - 1)^n - \;\; (r+1)^n = \;\; (2r+1)^n - \;\; (r+1)^n \qquad (2.7)$$

Наведемо приклад. Дано системи із 3-х (**n** = 3) послідовно розташованих елементів із значенням відновлення 1 (**r** = 1).

Тоді множина $\{1_0\;1_1\;\;0_0\;\}$ - всіх можливих елементів для розміщення (оскільки значення 0_1 в розміщенні для послідовної системи ми не розглядаємо, бо при такому значенні система має критичну відмову і такі стани не розглядаються), а **n** = 3 – кількість елементів, що потрібно розмістити, тому кількість станів для даної системи дорівнює:

$A_n^m = n^m = \ (2 * (r + 1) - 1)^n + 1 = \ (2 * (1 + 1) - 1)^3 \ + 1 = 28$,

Це значення – мінімальне значення можливих станів для системи із **n** = 3 і **r** = 1, для систем із такими ж значеннями **n** і **r**, але із паралельним або змішаним з'єднанням елементів – множина станів буде більшою.

$(r + 1)^n \ = 2^3 = 8$ – робочі стани із 28 станів,

$(2r + 1)^n - \ (r + 1)^n = \ 3^3 - \ 2^3 \ = 19$ – стани простою із 28 станів.

Дані формули 2.1 і 2.2 (а також відповідні формули 2.3 - 2.7) зручно використовувати, для швидкої оцінки системи на максимальнj і мінімальнj можливу кількість станів системи із відомими значеннями **n** і **r,** оскільки непотрібно ніяких спеціальних технічних засобів, складних обрахунків і т.д.

2.5. Висновки до розділу 2

У даному розділі описано метод автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами. Запропонований підхід дає змогу інтегрувати моделі надійності вищого порядку в програмні засоби для аналізу показників надійності складних технічних систем. Цей метод заснований на методах обходу графів і схожий на алгоритм Дейкстри. Розглянутий метод можна застосувати до Марковських процесів змінного порядку, використовуючи різні значення змінної, що відповідають порядку процесу для кожного стану.

Для демонстрації практичного використання та перевірки розробленого методу його було реалізовано та апробовано на прикладі оцінки надійності програмного забезпечення польотів наносупутників CubeSat. Моделювання показує, що ланцюги Маркова вищого кого порядку можуть підвищити точність оцінки рівня відмов до 50% порівняно з традиційною моделлю ланцюга Маркова, тоді як різниця між результатами, отриманими за допомогою моделей другого та третього порядку, не є настільки значним (менше 10%). Таким чином, при оцінці надійності сучасних програмних систем слід враховувати процеси вищого порядку, тобто взаємозалежність програмних модулів. Для цього можна застосувати моделі надійності ЛМВП за допомогою описаного методу.

В даному розділі описано метод автоматизованого визначення функції працездатності для Марковських моделей надійності. Визначення такої функції виконують переважно вручну і це вимагає високої кваліфікації розробника та глибокого розуміння системи, що проєктується. Відповідно це збільшує імовірність внесення помилок в модель. Автоматизоване визначення функції працездатності зменшує імовірність внесення помилок і підвищує достовірність моделі.

На основі результатів тестування і випробовувань методу побудови функції працездатності та визначення графа станів і переходів на основі функції працездатності було виведено комбінаторні формули, які дозволяють визначити максимальну і мінімальну кількість станів системи, а також окремі формули для визначення максимальної і мінімальної кількості працездатних і станів простою із загальної множини для системи із n елементів і r відновлень. Дані формули дозволяють швидко оцінити можливий простір станів системи із відомими значеннями n і r без використання спеціальних технічних засобів.

РОЗДІЛ 3. МОДЕЛІ ТА МЕТОДИ ПРОГНОЗУВАННЯ ДЕФЕКТНОСТІ ПЗ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

Аналіз відомих методів і підходів для прогнозування дефектності програмного забезпечення (див. розділ 1.4.) показав, що із стрімким розвитком галузі машинного навчання, зросло зацікавлення до використання моделей і методів прогнозування дефектності на основі машинного навчання. Це зумовлено тим, що прогнозування дефектності дозволяє виявляти потенційно дефектні модулі системи на ранніх етапах життєвого циклу ПЗ, виправлення таких дефектів дозволяє розробляти більш якісне і надійне програмне забезпечення із меншими витратами (чим на швидше на ранньому ЖЦ ПЗ буде виявлено дефект, тим швидше його буде виправлено і менше ресурсів буде використано на етапі тестування і обслуговування).

Також аналіз літературних джерел показав, що даний напрямок (прогнозування дефектів ПЗ на основі машинного навчання) перебуває, ще на етапі розвитку і містить певні відкриті питання, які потребують дослідження.

Тому було вирішено виконати дослідження, яке спрямоване на удосконалення методів і моделей прогнозування дефектів ПЗ засобами машинного навчання на основі метрик коду, і складається з таких етапів і кроків:

Етап 1. Аналіз та удосконалення моделей дефектності ПЗ шляхом використання методів машинного навчання для вибору метрик коду ПЗ, що найсильніше корелюють із його надійністю.

1.1. Підготовка набору даних із сховища даних PROMISE Software Engineering Repository та об’єднання результатів про тестування 5-и проєктів NASA Metrics Data Program.

1.2. Аналіз методів вибору ознак, застосування різних методів до набору даних і визначення ознак, які є важливими для моделі в кожному із вибраних методів.

1.3. Порівняння різних моделей машинного навчання, таких як: методи RF – Random Forest, SVM – support vector machine, KNN – k-nearest neighbor, DT –

Decision tree класифікатор, AB – AdaBoost класифікатор, GB – Gradient Boosting для дослідження класифікації із вибраними метриками коду.

1.4. Вибір найкращої комбінації метрик програмного коду на основі точності кожного класифікатора, які можна буде використовувати для ефективного прогнозування дефектності модулів системи.

Етап 2. Розроблення методу класифікації модулів ПЗ за дефектністю, який використовує стекінгове ансамблюванням нейронних мереж і дає можливість підвищити точність прогнозування.

Усі дослідження виконано із використанням мови програмування *Python*, бібліотеки *Scikit-learn* (Scikit-learn - один з найбільш широко використовуваних пакетів Python для машинного навчання. Він дозволяє виконувати безліч операцій і надає безліч алгоритмів для роботи [77]) і бібліотеки *Keras* - відкритої нейромережної бібліотеки, що написана мовою Python, спроектована для швидких експериментів з нейронними мережами і мережами глибинного навчання, її можливості зосереджені на тому, щоб вона була зручною в користуванні, модульною та розширюваною [78].

3.1. Об'єднаний набір даних із сховища PROMISE Software Engineering

Для дослідження було використано дані із загальнодоступного сховища даних PROMISE Software Engineering Repository [79]. Для більшої ефективності дослідження і збільшення розміру вибірки було об'єднано декілька наборів даних із метриками коду. Дані були зібрані в NASA Metrics Data Program і містять інформацію про тестування таких проєктів:

- KC1 – це програмна система, написана на C++, яка реалізовує управління ресурсами про приймання і обробку наземних даних;
- KC2 – це програмна система, написана на C++, яка виконує обробку наукових даних (інша частина KC1, яка написана іншою командою, використовує певні спільні бібліотеки з KC1);
- PC1 – це модуль програмної системи, написаний на C, для керування польотом супутника на орбіті;

- CM1 – це модуль, написаний на C, для виконання певних функцій на космічному кораблі;
- JM1 – це система, написана на C, використовується для прогнозів на основі симуляцій для наземних систем.

Набір даних для дослідження складається з 15 123 записів, які складаються із 21-ї метрики коду (табл. 3.1), і цільової метрики *defects*, яка може містити значення *true* (модуль із зазначеними метриками з дефектом) або *false* (модуль із зазначеними метриками без дефекту). Набір даних є незбалансованим і містить 2 665 записів про дефектні модулі та 12 458 записів про модулі без дефектів.

Таблиця 3.1 – Метрики коду із вибраного набору даних PROMISE.

№	Позначення метрики	Визначення метрики
1	loc	кількість рядків коду за МакКейбом
2	v(g)	цикломатична складність за МакКейбом
3	ev(g)	суттєва складність за МакКейбом
4	iv(g)	складність дизайну за МакКейбом
5	n	загальна кількість операндів і операторів
6	v	обсяг на мінімальному виконанні
7	l	розмір програми
8	d	складність модуля
9	i	функціональність модуля
10	e	зусилля на написання програми/модуля
11	b	метрика Halstead
12	t	час на написання модуля
13	lOCode	кількість рядків коду за Halstead
14	lOComment	кількість рядків із коментарями за Halstead
15	lOBlank	кількість порожніх рядків за Halstead
16	lOCodeAndComment	загальна кількість рядків коду із коментарями
17	uniq_Op	кількість унікальних операторів

18	uniq_Opnd	кількість унікальних операндів
19	total_Op	загальна кількість операторів
20	total_Opnd	загальна кількість операндів
21	branchCount	кількість гілок в репозиторії, в яких змінювався модуль

Ці метрики, наведені в таблиці 3.1 (детальний опис метрик коду див. додаток В), є загальними для будь-якого програмного забезпечення і не залежать від технологій або мови програмування, які використовують при розробці. В основному вони описують складність, розміри, час роботи та зв'язки для певних програмних модулів.

Як було зазначено вище, набір даних є незбалансованим відносно значення метрики defects (2665/ 12458). У вибірці даних записів про модулі без дефектів є близько в п'ять раз більше ніж записів про модулі з дефектами, що і відображає реальний розподіл даних при тестуванні. Використання незбалансованої вибірки у машинному навчанні може призвести до неправильних результатів прогнозування. Якщо відношення дисбалансу є значним, то навчена модель може прогнозувати із загальною високою/низькою точністю, оскільки класифікатор, швидше за все, передбачить більшість значень, що належать до класу більшості [80]. Тому для подальшого дослідження будуть використовуватись збалансовані дані.

В таблиці 3.2 наведено розподіл даних по проєктах.

Таблиця 3.2 – Розподіл даних в проєктах

Проєкт	Модулів з дефектами	Модулів без дефектів	Кількість модулів
KC1	326	1783	2109
KC2	107	415	522
PC1	77	1032	1109
CM1	49	449	498
JM1	2106	8779	10885
	2 665	**12 458**	**15 123**

3.2. Автоматизований вибір метрик коду, що найбільше впливають на дефектність ПЗ, засобами машинного навчання

Вибір функцій або ознак (також відомий як вибір змінних, вибір атрибутів або вибір підмножини змінних) – це практика вибору підмножини відповідних ознак (предикторів та змінних) для використання у побудові моделі машинного навчання. Саме автоматичний вибір атрибутів, присутніх у даних (наприклад, стовпців у табличних даних), є найбільш значущими та доречними для проблеми прогнозного моделювання [81].

Метою вибору ознак у машинному навчанні є пошук найкращого набору функцій, що дозволяє будувати корисні моделі досліджуваних явищ. Використання методів вибору ознак, дає нам такі переваги на виході [82]:

- простіша модель – прості моделі легко пояснити та зрозуміти - занадто складна і незрозуміла модель не є цінною;
- коротший час навчання – більш точна підмножина функцій зменшує кількість часу, необхідного для навчання моделі;
- збільшення точності оціноки;
- менші затрати пам’яті при роботі.

Оскільки в нас існує цільова ознака *defects*, ми будемо в даному дослідженні розглядати методи вибору функцій під наглядом для виконання процесу відбору ознак, які поділяються на методи фільтрування (Filter methods), методи обгортання (Wrapper methods) і вбудовані методи (Embedded methods).

Методи фільтрування (Filter methods) – методи, основані на теорії ймовірності та статистичних підходах, і як правило, розглядають кожну ознаку незалежно. Основними із даного класу методів є такі методи: Хі-квадрат (chi-square), IG-індексування (обчислення information gain), Variance Threshold, Fisher Score, Anova F-value. Вони оцінюють важливість ознак лише на основі властивих їм характеристик, не включаючи жодного алгоритму навчання. Ефективно використовувати дані методи, якщо набір ознак дуже великий (сто і більше), оскільки методи фільтрації швидкі, вони можуть добре працювати як перший етап вибору, щоб виключити деякі змінні [83].

Оскільки наша вибірка складається із 21-ї метрики, даний клас методів, буде неефективно використовувати у нашому досліджені для методу вибору ознак.

Методи обгортки працюють шляхом оцінки підмножини функцій за допомогою алгоритму машинного навчання, який використовує стратегію пошуку для перегляду простору можливих підмножин функцій, оцінюючи кожну підмножину на основі якості (продуктивності) виконання вибраного алгоритму машинного навчання [84].

Вбудовані методи виконують процес вибору функцій у рамках побудови самого алгоритму машинного навчання. Іншими словами, вони виконують вибір функцій під час навчання моделі, тому ми називаємо їх вбудованими методами. Даний клас методів реалізує в собі вигідні аспекти двох попередніх класів методів. На відміну від методів обгортання, які послідовно розглядають неважливі ознаки на основі метрики оцінки, вбудовані методи паралельно виконують вибір функцій та навчання алгоритму [85].

Перед початком використання методів вибору ознак, було побудовано матрицю кореляції даних, оскільки це ефективний інструмент для узагальнення великого набору даних, а також для визначення та візуалізації закономірностей у наведених даних (рис. 3.1).

На рисунку 3.1 наведено матрицю кореляції для вибірки даних, де видно, що є значна кореляція між метриками *e* і *t*, а також між метриками *b* і *v*, коефіцієнт якої дорівнює 1. Кореляція між *n* (Operator and Operand total (Halstead)) і total_Op рівна 1, а між *n* (Operator and Operand total (Halstead)) і *total_Opnd* коефіцієнт кореляції становить 0.99. Також є значною кореляція 0.93 між *loc* і *lOCode*.

Значний коефіцієнт кореляції між переліченими змінними є достатньо зрозумілим і закономірним, оскільки дані метрики так чи інакше є подібними або означають одне і те саме, або включають одна одну. Із набору взаємопов’язаних метрик можна видалити всі, крім одного, без значних втрат інформації і впливу на якість моделі.

Зважаючи на значення кореляції, було прийнято рішення відкинути такі метрики з вибраного набору даних: t, b, total_Op. total_Opnd, lOCode.

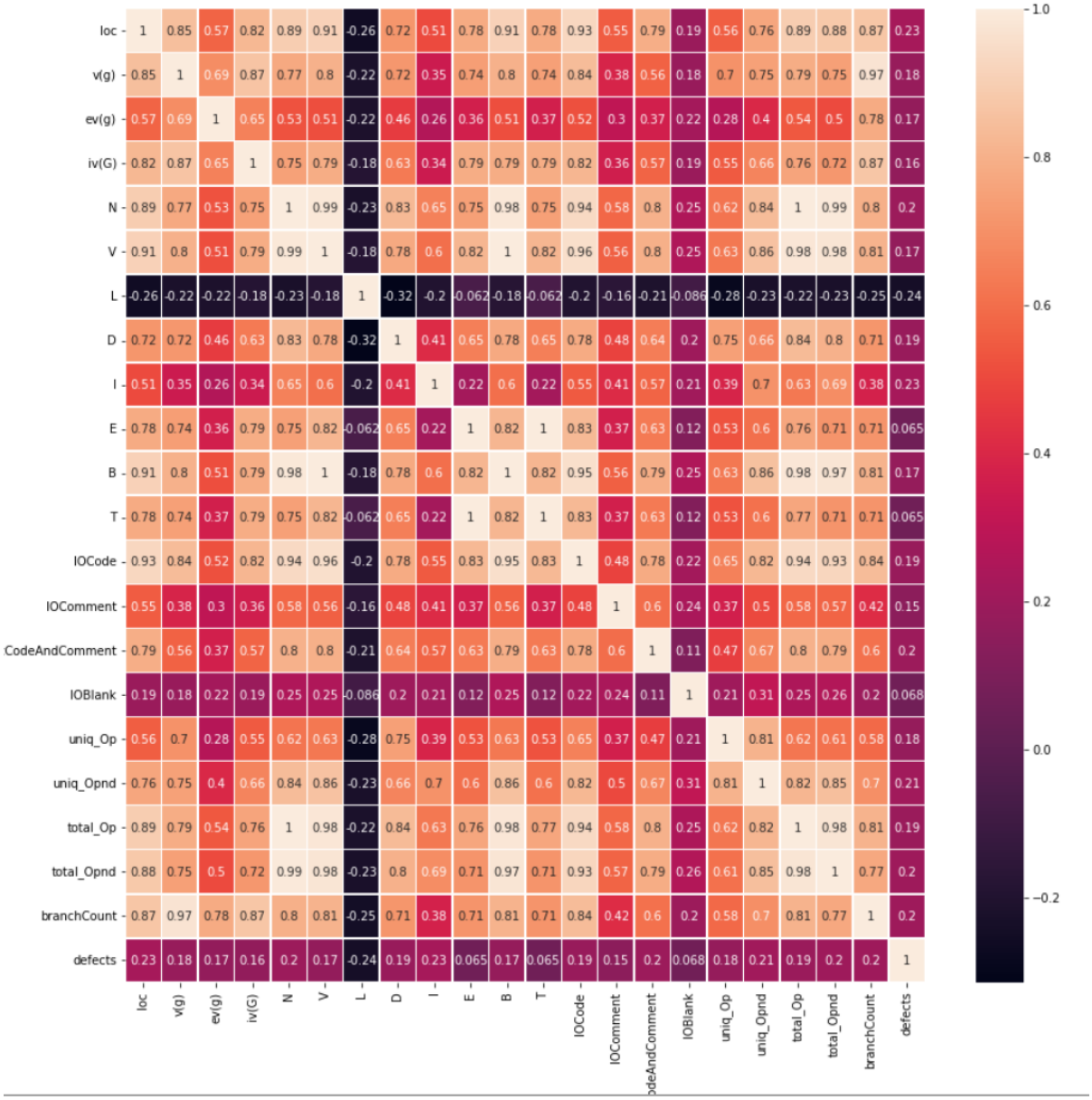


Рис. 3.1. Матриця кореляції для набору даних із сховища PROMISE

Для вибору ознак ми використали такі методи вибору ознак:

Boruta (Using RandomForestClassifier). Борута - це метод на основі випадкових лісів, тому він працює для моделей дерев, таких як Random Forest або XGBoost, але також діє з іншими моделями класифікації, такими як логістична регресія або SVM. Борута послідовно видаляє функції, які статистично менш актуальні, ніж випадковий зонд (змінні штучного шуму, введені алгоритмом Борути). У кожній ітерації відхилені змінні видаляються з розгляду в наступній ітерації. Як правило, це закінчується хорошою глобальною оптимізацією вибору функцій [86].

Застосувавши даний метод вибору ознак для об'єднаного набору даних із сховища PROMISE (рис. 3.2.), було визначено, що важливими для подальшого дослідження є наступні метрики : *loc, v(g), iv(G), N, V, I, E, locCodeAndComment, uniq_Opnd, branchCount.*

```
#!pip install Boruta
from boruta import BorutaPy
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_jobs=-1, class_weight='balanced', max_depth=8)
boru_selector = BorutaPy(rf, n_estimators=100, verbose=1, random_state=1)
boru_selector.fit(X.values, y.values)

#concat  dataframes for better visualization
xcolumns = pd.DataFrame(X.columns)
supports =  pd.DataFrame(boru_selector.support_)
rankings =  pd.DataFrame(boru_selector.ranking_)

featureScores = pd.concat([xcolumns,rankings,supports],axis=1)
featureScores.columns = ['Column','Ranking','Use']  #naming the dataframe columns
print(featureScores.nlargest(100,'Use'))
```

Рис. 3.2. Реалізація методу Boruta для вибору ознак

Exhaustive Feature Selection. При вичерпному виборі функцій (Exhaustive Feature Selection) ефективність алгоритму машинного навчання оцінюється на основі всіх можливих комбінацій функцій у наборі даних. Вибирається підмножина функцій, що забезпечує найкращу продуктивність. Вичерпний алгоритм пошуку – є найбільш затратним по ресурсах алгоритмом з усіх методів обгортки, оскільки він випробовує всі комбінації функцій і вибирає найкращу [87].

На рисунку 3.3 наведено реалізацію методу із вибором таких ознак для аналізу*: loc, v(g), N, V, I, branchCount.*

```
from mlxtend.feature_selection import ExhaustiveFeatureSelector
from sklearn.neighbors import KNeighborsClassifier


knn = KNeighborsClassifier(n_neighbors=3)

efs = ExhaustiveFeatureSelector(knn,
           min_features=4,
           max_features=10,
           scoring='accuracy',
           print_progress=True,
           cv=5)

efs = efs.fit(X, y)


print('Best accuracy score: %.2f' % efs.best_score_)
print('Best subset (indices):', efs.best_idx_)
print('Best subset (corresponding names):', efs.best_feature_names_)
```

Рис. 3.3. Реалізація методу Exhaustive Feature Selection для вибору ознак

Random Forest Importance. Вибір функцій за допомогою випадкового лісу належить до категорії вбудованих методів. Випадковий ліс – це свого роду алгоритм обробки даних, який об'єднує певну кількість дерев рішень. Стратегії на основі дерев, які використовують випадкові ліси, природно ранжуються за тим, наскільки добре вони покращують чистоту вузла, або іншими словами, зменшують домішки (домішки Джині) над усіма деревами. Вузли з найбільшим зменшенням домішок трапляються на початку дерев, тоді як ноти з найменшим зменшенням домішок трапляються в кінці дерев. Таким чином, обрізаючи дерева під певним вузлом, ми можемо створити підмножину найважливіших ознак [88].

Обчислення важливості ознак виконується із використанням домішків вузлів у кожному дереві рішень. У випадковому лісі остаточна важливість ознак – це середнє значення всіх властивостей дерева рішень.

Метод вибору ознак вказав, що важливими для подальшого дослідження є наступні метрики (рис. 3.4): *loc, N, V, D, I, E, locCodeAndComment*.

```
#!pip install scikit-learn
from sklearn.feature_selection import  SelectFromModel
from sklearn.ensemble import RandomForestClassifier

embeded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimators=500), max_features=16)
embeded_rf_selector.fit(X, y)

embeded_rf_support = embeded_rf_selector.get_support()

embeded_rf_feature = X.loc[:,embeded_rf_support].columns.tolist()
print(str(len(embeded_rf_feature)), 'selected features')
print(embeded_rf_feature)
```

```
7 selected features
['loc', 'N', 'V', 'D', 'I', 'E', 'locCodeAndComment']
```

Рис. 3.4. Реалізація і результати роботи методу Random Forest Importance

Light Gradient Boosting Machine Importance. Даних підхід подібний до методу описаного вище (Random Forest Importance), проте в основі використовує алгоритм Light GBM, оскільки він також має атрибут *feature_importance*. Light GBM - алгоритм навчання на основі дерев, Light GBM зростає дерево вертикально, тоді як інші алгоритми - горизонтально, що означає, що даний алгоритм зростає по дереву по листках, тоді як інший алгоритм-по рівню [89].

Застосувавши даний метод вибору ознак для нашої вибірки, було вибрано наступні метрики коду (рис. 3.5): *loc, N, V, L, D, I, E, uniq_Op*.

```
from sklearn.feature_selection import SelectFromModel
from lightgbm import LGBMClassifier


lgbc=LGBMClassifier(n_estimators=500, learning_rate=0.05, num_leaves=32, colsample_bytree=0.2,
            reg_alpha=3, reg_lambda=1, min_split_gain=0.01, min_child_weight=40)

embeded_lgb_selector = SelectFromModel(lgbc, max_features=16)
embeded_lgb_selector.fit(X, y)

embeded_lgb_support = embeded_lgb_selector.get_support()
embeded_lgb_feature = X_train.loc[:,embeded_lgb_support].columns.tolist()
print(str(len(embeded_lgb_feature)), 'selected features')
print(embeded_lgb_feature)
```

```
8 selected features
['loc', 'N', 'V', 'L', 'D', 'I', 'E', 'uniq_Op']
```

Рис. 3.5. Реалізація і результати роботи методу Light GBM

Step-wise selection. Поетапний вибір (Step-wise selection) двоспрямований, заснований на поєднанні прямого вибору (Forward selection) та усуненні назад (Backward elimination). Його вважають менш затратним по ресурсах, оскільки там передбачена можливість додавання предикторів назад до моделі, які були видалені (і навпаки). Однак, такі висновки все ще зроблені на основі локальної оптимізації на будь-якій ітерації.

Оскільки даний метод базується на послідовному додаванні/відніманні метрик з множини, потрібно вказати точну їх кількість для вибору. Тому використано бібліотеку *plot_sequential_feature_selection* для візуалізації оцінки при різних значеннях параметра *k_features* (значення параметра означає скільки метрик потрібно вибрати із набору даних), щоб полегшити прийняття рішення (рис. 3.6).

Рис. 3.6. Візуалізації точності при різних значеннях параметра k_features для Step-wise методу

З наведеного рисунка видно, що найкраще значення метрики *Accuracy* при виборі 13 метрик. Реалізація методу Step-wise для вибору ознак (рис. 3.7)

визначила наступні ознаки, які будуть надалі використані в дослідженні : *loc, ev(g), iv(G), N, V, L, I, E, lOComment, locCodeAndComment, branchCount, uniq_Op, uniq_Opnd.*

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.ensemble import RandomForestClassifier
import pandas as pd


#Define Sequential Forward Selection (sfs)
sffs = SFS(RandomForestClassifier(),
           k_features=13,
           forward=True,
           floating=True,
           scoring = 'accuracy',
           cv = 2)

#Use SFS to select the top 13 features
sffs.fit(X_train, y_train)
print(sffs.subsets_[13])
```

Рис. 3.7. Реалізація методу Step-wise selection для вибору ознак

Autoencoders method. Машинне навчання передбачає використання нейронних мереж для створення високоефективних моделей машинного навчання. Що особливо цікаво в нейронних мережах, то це їх здатність вивчати нелінійні зв'язки між ознаками. Більшість традиційних методів, які було використано і досліджено, цього не роблять – загалом, методи, які було розглянено, можуть досліджувати лише лінійні зв'язки між об'єктами. Даний метод використовує особливу архітектуру нейронної мережі, яка називається автокодерами. AutoEncoder використовуються для вибору ознак, щоб розкрити наявні нелінійні зв'язки між ознаками [93]. За допомогою даного методу було визначено такі важливі ознаки (рис. 3.8): *loc, v(g), N, branchCount, V, E.*

```
from keras.layers import Dense
from keras.models import Sequential
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)

# create the autoencoder.
model = Sequential()

# add the encoding layer.
model.add(Dense(16, activation='relu', input_shape=(X_train.shape[1],)))

# add the output layer.
model.add(Dense(X_train.shape[1], activation='linear'))

# compile the model, you can use whatever optimizer.
model.compile(optimizer='adadelta', loss='categorical_crossentropy')
model.fit(X_train, X_train, epochs=50, batch_size=64, shuffle=True, alidation_data=(X_test, X_test))

# get the first layer weights.
weights = model.layers[1].get_weights()[0]
print(weights.sum(axis = 1))
```

```
Index(['loc', 'v(g)', 'N', 'branchCount', 'V', 'E'], dtype='object')
```

Рис. 3.8. Реалізація і результати роботи методу Autoencoders

Xverse python for feature selection. Xverse – метод, розроблений на основі мови програмування Python, застосовує різноманітні методи для вибору функцій. Коли алгоритм вибирає функцію, він дає голос за цю функцію. Зрештою, Xverse обчислює загальну кількість голосів для кожної ознаки, а потім вибирає найкращі на основі голосів. Таким чином ми вибираємо найкращі змінні з мінімальними зусиллями в процесі вибору функцій. Xverse використовує такі методи для вибору важливих функцій [92]:

- Information Value using Weight of evidence;
- Variable Importance using Random Forest;
- Recursive Feature Elimination;
- Variable Importance using Extra trees classifier;
- Chi-Square best variables;
- L1 based feature selection.

Відповідно до результатів роботи даного методу (рис. 3.9) було вибрано метрики, за які проголосувала більшість методів, які реалізує Xverse (4/6): *loc*, *locCodeAndComment*, *uniq_Opnd*, *branchCount*, *N*, *I*, *L*.

Variable_Name	Information_Value	Random_Forest	Recursive_Feature_Elimination	Extra_Trees	Chi_Square	L_One	Votes
loc	1	1	1	1	1	1	6
locCodeAndComment	1	1	1	1	1	0	5
uniq_Opnd	1	1	1	1	0	1	5
branchCount	1	0	1	0	1	1	4
N	1	1	0	1	1	0	4
I	1	1	0	1	1	0	4
L	0	0	1	1	1	1	4
v(g)	1	0	1	0	0	1	3
V	0	1	0	1	1	0	3
D	0	1	1	0	0	1	3
iv(G)	1	0	0	0	0	1	2
uniq_Op	0	0	0	1	0	1	2
E	0	1	0	0	0	0	1
ev(g)	0	0	0	0	1	0	1
IOComment	0	0	1	0	0	0	1
IOBlank	0	0	0	0	0	0	0

Рис. 3.9. Результати голосування за вибір ознак методом Xverse

Genetic Algorithms. GA або генетичні алгоритми - це глобальні методи оптимізації для пошуку дуже великих просторів. Вони були натхнені

біологічними механізмами природного відбору та розмноження. Вони працюють у всіх популяціях можливих рішень (так званих поколінь), де кожне рішення в просторі пошуку представлено у вигляді рядка скінченної довжини (хромосоми) над деяким кінцевим набором символів, який потім використовує цільову (або придатну) функцію для оцінки відповідності кожної метрики [90]. З погляду вибору ознак, кожна хромосома буде позначати підмножину ознак, і вона буде представлена двійковим кодуванням: 1 – означає «вибрати» певну ознаку, а 0 – означає «не вибирати» ознаку.

Реалізація і результати використання даного методу продемонстровано на рисунку 3.10, і вибрано наступні ознаки для подальшого дослідження: *loc, iv(G), I, lOComment, locCodeAndComment, uniq_Op.*

```
] #!pip install sklearn-genetic
  from genetic_selection import GeneticSelectionCV
  from sklearn.ensemble import RandomForestClassifier

  rf = RandomForestClassifier(n_estimators=100)

  genetic_selection = GeneticSelectionCV(rf,
                                cv=5, scoring="accuracy",
                                max_features=6, n_population=10,
                                crossover_proba=0.5, mutation_proba=0.2,
                                n_generations=50, crossover_independent_proba=0.5,
                                mutation_independent_proba=0.05, n_gen_no_change=10,
                                n_jobs=-1)

  genetic_selection = selection.fit(X_train, y_train)

  selected_columns = X.columns[genetic_selection.support_]
  print(selected_columns)

['loc', 'iv(G)', 'I', 'lOComment', 'locCodeAndComment', 'uniq_Op']
```

Рис. 3.10. Реалізація і результати роботи методу Genetic Algorithms

Principal Component Analysis. Аналіз головних компонентів, або PCA, – це статистичний метод для перетворення даних великої розмірності в дані з низькими розмірами шляхом вибору найважливіших функцій, які охоплюють максимальну інформацію про набір даних. Характеристики вибирають на основі дисперсії, яку вони викликають у вихідних даних. Особливість, яка викликає найбільшу дисперсію, є першим головним компонентом. Функцію, яка відповідає за другу найбільшу дисперсію, вважають другим головним компонентом і т.д.. Важливо зазначити, що основні компоненти не мають жодної кореляції між собою [91].

Застосувавши даний метод вибору ознак для вибірки даних, отримали наступні метрики коду (рис. 3.11) : *loc, v(g), ev(g), iv(G), N ,V, L, D, I, E.*

```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)

pca = PCA(n_components=10)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_

#concat  dataframes for better visualization
xcolumns = pd.DataFrame(X.columns)
scores =  pd.DataFrame(explained_variance)

featureScores = pd.concat([xcolumns,scores],axis=1)
featureScores.columns = ['Column','Score']
print(featureScores.nlargest(100,'Score'))
```

```
  Column         Score
0    loc  9.999904e-01
1   v(g)  9.548435e-06
2  ev(g)  9.202478e-09
3  iv(G)  8.380486e-09
4      N  3.156633e-09
5      V  1.714500e-09
6      L  9.942540e-10
7      D  4.657042e-10
8      I  1.754475e-10
9      E  1.365505e-10
```

Рис. 3.11. Реалізація і результати роботи методу Principal Component Analysis

3.3. Регресійна модель дефектності ПЗ з редукованою множиною ознак

Для вибору ознак було використано 9 методів вибору ознак, які відрізняються своїми типами та підходами до вибору ознак:

- Boruta using RandomForestClassifier (Boruta)
- Step-wise selection (SWS)
- Exhaustive Feature Selection (EFS)
- Random Forest Importance (RFI)
- LightGBM Importance (LightGBM)
- Genetic Algorithms (GA)
- Principal Component Analysis (PCA)
- Xverse python for feature selection (Xverse)
- Autoencoders method (Autoencoder)

Результати роботи цих методів продемонстровано на рисунку 3.12, де описано інформацію про кількість голосів за певну ознаку, які були вибрані методами, що описані вище (1 – ознака була вибрана методом вибору ознак, 0 – не вибрана, для спрощення візуалізації метрики за які не було жодного голосу виключені з таблиці).

Ознака	Boruta	SWS	EFS	RFI	LGBM	GA	PCA	Xverse	Autoencoder	Загалом
loc	1	1	1	1	1	1	1	1	1	9
N	1	1	1	1	1	0	1	1	1	8
I	1	1	1	1	1	1	1	1	0	8
V	1	1	1	1	1	0	1	0	1	7
E	1	1	0	1	1	0	1	0	1	6
v(g)	1	1	1	0	0	0	1	0	1	5
branchCount	1	1	1	0	0	0	0	1	1	5
iv(G)	1	1		0	0	1	1	0	0	4
locCodeComm	1	0	0	1	0	1	0	1	0	4
uniq_Opnd	1	1	0	0	0	1	0	1	0	4
L	0	1	0	0	1	0	1	1	0	4
uniq_Op	0	1	0	0	1	1	0	0	0	3
D	0	0	0	1	1	0	1	0	0	3
lOComment	0	1	0	0	0	1	0	0	0	2
ev(g)	0	1	0	0	0	0	0	0	0	1

Рис. 3.12.. Результати роботи методів вибору ознак

Як важливі метрики, які мають значний вплив при прогнозуванні дефектності ПЗ, було вибрано ознаки (надалі дана множина ознак називатиметься Important features (Важливі метрики)), які отримали більш ніж половину голосів із всіх використаних методів вибору ознак (5/9): *loc, N, I, V, E, v(g), branchCount.*

Для того, щоб отримати прогноз чи модуль дефектний, чи без дефектів, було використано різні методи класифікації. Наступним етапом було порівняння різних моделей машинного навчання, таких як методи RF – Random Forest, SVM – support vector machine, KNN – nearest neighbor, DT – decision tree класифікатор, AB – AdaBoost класифікатор, GB – Gradient Boosting для класифікації. Було реалізовано і порівняно результати роботи класифікаторів з різними ознаками, які були вибрані на попередньому етапі, щоб вибрати найкращу комбінацію метрик програмного коду на основі точності кожного класифікатора, які можна буде використовувати для прогнозування дефектності системи.

Для дослідження було виконано три раунди експериментів із використанням трьох збалансованих вибірок. Оскільки співвідношення записів про модулі з дефектами та без дефектів є нерівномірне (2665/12458), було виконано балансування і створення вибірок для дослідження таким чином. Вибрано 2665 унікальних записів із множини записів без дефектів і об'єднано з 2665 записами про модулі з дефектами, таким чином утворилася одна вибірка даних для дослідження, яка складається із 50% даних про модулі без дефектів і 50 % даних про модулі з дефектам. Наступні дві вибірки утворювались подібним чином (якщо записи з множини записів без дефектів були вибрані на попередньому етапі, їх виключаємо із множини, щоб при наступному виборі вони не були залучені до вибірки).

Для тренування моделей було вибрано із кожної вибірки 80% даних, а для тестування використано 20% даних.

Продуктивність даних класифікаторів була оцінена за допомогою метрики точності (Accuracy). Точність – це метрика, наскільки часто навчена модель є правильною, результати класифікації часто подаються у вигляді матриці помилок. Матриця складається з 4-х різних комбінацій прогнозованих та фактичних значень. Прогнозовані значення описуються як позитивні та негативні, а фактичні – як справжні та хибні (TP – істинно позитивних, TN – істинно негативних; FP – хибно позитивних, FN – хибно негативних). Метрика точності (Accuracy) показує таке співвідношення (відношення правильних передбачень до їх загальної кількості):

$$Accuracy=(TP+TN)/(TP+TN+FP+FN) \quad (3.1)$$

В таблиці 3.3 наведено результати роботи класифікаторів із усередненою точністю оцінки для ознак, які були вибрані методами вибору ознак.

В таблицях 3.4-3.9. наведено результати точності класифікації (вибраними методами класифікації) із використанням метрик вибраних різними методами для трьох експериментальних вибірок даних (деякі назви методів вибору ознак були скорочені).

Таблиця 3.3 – Результати роботи класифікаторів для ознак, вибраних різними методами

Метод вибору ознак	**Середня точність**					
	RF	**SVM**	**KNN**	**DT**	**AB**	**GB**
Boruta (Bo)	0.812	0.827	0.783	0.777	0.792	0.812
SWS	0.777	0.796	0.763	0.755	0.783	0.781
EFS	0.840	0.851	0.803	0.791	0.820	0.834
RFI	0.801	0.820	0.817	0.752	0.813	0.809
LGBM	0.806	0.807	0.794	0.772	0.801	0.822
GA	0.795	0.805	0.782	0.767	0.805	0.798
PCA	0.822	0.830	0.820	0.799	0.842	0.822
Xverse (XV)	0.804	0.810	0.789	0.778	0.810	0.799
Encoder (EN)	0.845	0.843	0.817	0.765	0.829	0.821
Важливі ознаки (7) (IF)	0.857	0.835	0.827	0.801	0.815	0.854
Всі ознаки (21) (All)	0.705	0.725	0.632	0.617	0.720	0.695

Таблиця 3.4 – Результати роботи RF класифікатора із точністю оцінки для ознак, вибраних різними методами для трьох вибірок даних

№	Точність (метод RF)										
	Bo	SWS	EFS	RFI	LGBM	GB	PCA	XV	EN	IF	All
1	0.811	0.782	0.84	0.803	0.812	0.795	0.821	0.805	0.844	0.856	0.702
2	0.815	0.775	0.86	0.799	0.808	0.799	0.825	0.800	0.85	0.859	0.711
3	0.809	0.773	0.82	0.801	0.799	0.790	0.819	0.806	0.841	0.856	0.701

Таблиця 3.5 – Результати роботи SVM класифікатора із точністю оцінки для ознак, вибраних різними методами для трьох вибірок даних

№	Точність (метод SVM)										
	Bo	SWS	EFS	RFI	LGBM	GB	PCA	XV	EN	IF	All
1	0.827	0.797	0.852	0.820	0.807	0.801	0.829	0.810	0.847	0.838	0.725
2	0.832	0.790	0.854	0.800	0.819	0.815	0.840	0.812	0.842	0.831	0.720
3	0.822	0.800	0.848	0.840	0.795	0.799	0.831	0.809	0.84	0.835	0.730

Таблиця 3.6 – Результати роботи KNN класифікатора із точністю оцінки для ознак, вибраних різними методами для трьох вибірок даних

№	Точність (метод KNN)										
	Bo	SWS	EFS	RFI	LGBM	GB	PCA	XV	EN	IF	All
1	0.782	0.765	0.804	0.817	0.794	0.780	0.821	0.789	0.817	0.823	0.618
2	0.790	0.752	0.811	0.821	0.802	0.789	0.825	0.781	0.82	0.826	0.631
3	0.778	0.771	0.794	0.812	0.785	0.778	0.815	0.796	0.814	0.831	0.647

Таблиця 3.7 – Результати роботи DT класифікатора із точністю оцінки для ознак, вибраних різними методами для трьох вибірок даних

№	Точність (метод DT)										
	Bo	SWS	EFS	RFI	LGBM	GB	PCA	XV	EN	IF	All
1	0.778	0.754	0.792	0.752	0.771	0.769	0.799	0.778	0.766	0.798	0.617
2	0.77	0.746	0.791	0.763	0.762	0.753	0.8	0.782	0.772	0.799	0.624
3	0.782	0.766	0.789	0.742	0.782	0.778	0.798	0.774	0.758	0.805	0.610

Таблиця 3.8 – Результати роботи AB класифікатора із точністю оцінки для ознак вибраних різними методами для трьох вибірок даних

№	Точність (метод AB)										
	Bo	SWS	EFS	RFI	LGBM	GB	PCA	XV	EN	IF	All
1	0.793	0.786	0.821	0.816	0.803	0.803	0.842	0.808	0.823	0.815	0.715
2	0.798	0.779	0.815	0.808	0.791	0.809	0.838	0.809	0.831	0.821	0.719
3	0.786	0.785	0.824	0.815	0.81	0.804	0.845	0.814	0.832	0.809	0.727

Таблиця 3.9 – Результати роботи GB класифікатора із точністю оцінки для ознак, вибраних різними методами для трьох вибірок даних

№	Точність (метод GB)										
	Bo	SWS	EFS	RFI	LGBM	GB	PCA	XV	EN	IF	All
1	0.807	0.776	0.841	0.8	0.82	0.799	0.823	0.799	0.822	0.86	0.691
2	0.819	0.782	0.832	0.809	0.826	0.8	0.818	0.792	0.814	0.852	0.7
3	0.809	0.786	0.829	0.817	0.819	0.795	0.825	0.806	0.828	0.85	0.694

З аналізу таблиць можна побачити, що сім важливих ознак (Important features), обраних на основі голосування з усіх методів показують найвищу точність прогнозування для 4 з 6 використаних алгоритмів класифікації; у випадку SVM ці зважені ознаки хоча й не дали найбільшої точності, все ж входять в трійку найбільш точних методів вибору ознак, і тільки для класифікації методом AdaBoost використання цієї множини ознак показує дещо гірший результат прогнозування після методів PCA, Autoencoder та EFS. Крім того, в трійку лідерів за точністю прогнозування входять такі методи вибору ознак, як EFS – точність п’яти з шести алгоритмів класифікації при використанні цієї множини ознак є в трійці лідерів, а у випадку методу SVM ця точність є найвищою з усіх; Autoencoder – 5 з 6 алгоритмів класифікації є в трійці лідерів за точністю прогнозу;

PCA – 4 з 6 алгоритмів є в трійці лідерів, однак для двох з них (Decision tree та AdaBoost класифікатор) ця точність є найвищою.

Таким чином можна зробити висновок, що для даного дослідження і використаних алгоритмів класифікації найкращу для більшості класифікаторів точність прогнозу вдалось отримати з використанням набору ознак із семи метрик (Important features), отриманого методом голосування з усіх досліджених методів вибору ознак. Крім того, показано, що можливим, також, є використання окремих методів, таких як Autoencoder, EFS та PCA практично без втрати точності класифікації та прогнозування. Також, показано значне підвищення точності прогнозування дефектів ПЗ шляхом зменшення вибірки ознак. Приріст точності прогнозування в такому випадку (усі ознаки та вибрані ознаки) становив від 10% до 21%. При цьому використання будь-якого методу вибору ознак підвищує точність класифікації принаймні на 10% у порівнянні з вибіркою, яка складається із всіх метрик, що підтверджує важливість цієї процедури (вибір ознак) для прогнозування дефектності ПЗ на основі метричних датасетів, які містять значну кількість метрик коду ПЗ, виміряних за різними підходами, які однак, мають сильну кореляцію.

Як бачимо з описаного вище, вибрані нами ознаки підвищують точність прогнозування дефектності програмного забезпечення, а відповідні метрики коду, отже, пов’язані з його надійністю і дають можливість побудувати на їх основі модель дефектності програмного забезпечення. Для побудови моделі дефектності ПЗ було використано множину метрик коду, отриманих як найважливіші ознаки на попередньому етапі та регресійний метод. Оскільки цільова (залежна) змінна є бінарною – метрика *defects*, що має значення *true* або *false* – в якості регресійного методу було обрано логістичну регресію. Цей вид регресії добре підходить для задач бінарної класифікації, що у нашому дослідженні означає класифікацію програмних модулів на такі, що містять дефекти, та бездефектні.

Отримане рівняння логістичної регресії виглядає наступним чином:

defects = $\beta 1 * loc + \beta 2 * v(g) + \beta 3 * N + \beta 4 * I + \beta 5 * branchCount + \beta 6 * V + \beta 7 * E + \beta 0$ (3.2)

Модель для логістичної регресії була створена за допомогою компонента *linear_model.LogisticRegression* (частина бібліотеки *sklearn*), а значення коефіцієнтів для рівняння логістичної регресії отримано за допомогою атрибута *coef_* моделі *LogisticRegression.* Значення коефіцієнтів регресії наведено в таблиці 3.10. Точність прогнозу цього рівняння регресії для тестової вибірки становила 0.8288 (82,88%), що цілком узгоджується з даними таблиці 3.3.

Таблиця 3.10 – Значення коефіцієнтів моделі надійності ПЗ

Коефіцієнт	Значення
β0	-0.99489
β1	8.8409e-03
β2	-1.7126e-01
β3	-1.5715e-02
β4	-7.1093e-03
β5	8.8159e-02
β6	2.4233e-03
β7	7.9552e-07

Як видно з таблиці 3.10 найбільший внесок в ймовірність того, що програмний модуль містить один або більше дефектів вносить цикломатична складність МакКейба, кількість гілок в програмі та кількість операторів та операндів за Холстедом. Цей результат добре узгоджується з висновками, зробленими в роботі [93], що надійність ПЗ залежить від його складності, однак ця залежність є комплексною і включає декілька факторів (метрик складності).

3.4. Метод класифікації модулів ПЗ за дефектністю на основі стекового ансамблю нейронних мереж

Як було зазначено в огляді літературних джерел (дивитись розд. 1.4), традиційні методи машинного навчання у наш час не дають достатньо високої точності прогнозування дефектності програмного забезпечення внаслідок

достатньо великої множини ознак, багато з яких мають сильну взаємну кореляцію, а також внаслідок недослідженості питання переносимості виявлених ознак між різними проєктами при створенні програмного забезпечення, які мають різні характеристики розробки. В такому випадку перспективним є використання методів глибинного навчання та ансамблів нейронних мереж для виявлення ознак, підвищення точності прогнозування. Тому було виконано дослідження, спрямоване на застосування методів на основі нейронних мереж для бінарної класифікації програмних модулів на дефектні та бездефектні на основі множини метрик програмного коду.

Оптимізація гіперпараметрів є важливою частиною при навчанні нейронних мереж. Причина в тому, що нейронні мережі, як відомо, важко налаштувати, і існує багато параметрів, які потрібно встановити. Продуктивність моделей істотно залежить від значення гіперпараметрів. Неможливо заздалегідь дізнатися найкращі значення для гіперпараметрів, тому в ідеалі потрібно спробувати всі можливі значення, щоб дізнатися оптимальні значення. Виконання цього вручну може зайняти значну кількість часу та ресурсів, тому у даному дослідженні було використано функцію GridSearchCV для автоматизації налаштування гіперпараметрів.

GridSearchCV — це функція, яка постачається в пакеті model_selection бібліотеки Scikit-learn. Ця функція допомагає перебирати попередньо визначені гіперпараметри та вставляти їх в модель. В результаті ми отримуємо комбінацію гіперпараметрів, які дозволяють отримати найкращі результати обчислень [94].

У таблиці 3.11 наведено параметри, які були передані в функцію GridSearchCV і їхні визначення.

Таблиця 3.11 – Параметри і їх значення для функції GridSearchCV

Параметр	**Значення**	**Визначення**
epochs	1, 5, 10, 50, 100	кількість епох навчання
neurons	5, 10, 20, 50	кількість нейронів у шарі мережі

optimizer	'SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam'	параметр допомагає знайти оптимальні значення кожної ваги в нейронній мережі
batch_size	1, 64, 128, 256	вказує, скільки рядків буде передано в мережу за один раз, після чого розпочнеться обчислення, а нейронна мережа почне коригувати свої ваги на основі помилок
activation	'softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear'	визначає функцію активації для обчислень всередині кожного нейрона

На рисунках 3.13 і 3.14 наведено приклад використання і результати (фрагмент розрахунку) роботи функції GridSearchCV.

```
# Creating the classifier
classifierModel=KerasClassifier(make_classification_rnn_lstm, verbose=0)

# Creating the Grid search space
grid_search=GridSearchCV(estimator=classifierModel, param_grid=Parameter_Trials, scoring='accuracy', cv=5)

# Running Grid Search for different paramenters
grid_result=grid_search.fit(X_train,y_train, verbose=1)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Рис. 3.13. Використання функції GridSearchCV.

```
0.514286 (0.448125) with: {'Activation': 'relu', 'Neurons_Trial': 15, 'Optimizer_Trial': 'SGD', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'relu', 'Neurons_Trial': 15, 'Optimizer_Trial': 'Adam', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'relu', 'Neurons_Trial': 20, 'Optimizer_Trial': 'SGD', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'relu', 'Neurons_Trial': 20, 'Optimizer_Trial': 'Adam', 'batch_size': 128, 'epochs': 10}
0.285714 (0.393830) with: {'Activation': 'tanh', 'Neurons_Trial': 15, 'Optimizer_Trial': 'SGD', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'tanh', 'Neurons_Trial': 15, 'Optimizer_Trial': 'Adam', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'tanh', 'Neurons_Trial': 20, 'Optimizer_Trial': 'SGD', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'tanh', 'Neurons_Trial': 20, 'Optimizer_Trial': 'Adam', 'batch_size': 128, 'epochs': 10}
```

Рис. 3.14. Фрагмент розрахунку параметрів за допомогою функції GridSearchCV.

Балансування даних у дослідженні було виконано за допомогою методу *RandomOverSample* з бібліотеки *sklearn.* Цей метод використовує один із способів

балансування (Oversampling) при якому відбувається збільшення кількості записів із вибірки меншого класу до розміру вибірки більшого класу за допомогою дублювання записів. У нашій вибірці даних, до класу із меншою кількістю записів належать записи про модулі з дефектами (*2 665*), а записи про модулі без дефектів мають значну кількісну перевагу (*12 458*), після використання методу *RandomOverSample* розмір вибірки було збільшено до *24 916* записів (*12 458/12 458*).

Для оцінки класифікації за допомогою вибраних методів було використано метод перехресної перевірки (стратифікована k-кратна перехресна перевірка). Перехресна перевірка потрібна для того, щоб оцінити наскільки результати дослідження узагальнюватимуться на незалежний набір даних.

Стратифікована k-кратна перехресна перевірка працює так (було використано функцію *StratifiedKFold* з бібліотеки *sklearn,* яка реалізовує даний метод перехресної перевірки):

1) Вибрана вибірка даних перемішується випадковим чином;
2) Вибірка даних розділяється на k рівних частин. Було використано стратифіковану перехресну перевірку, оскільки при даному типі перевірки вибірка розбивається на k рівних частин, де кількість записів кожного класу є рівно розподілена (у нашому випадку 50% на 50%), цей тип затвердження добре підходить для роботи при бінарній класифікації;
3) Вибирається одна із k частин вибірок для тестування і k-1 для тренування, такий процес виконується циклічно, поки не буде виконано тестування і тренування моделей із використанням всіх частинок вибірки даних.
4) Узагальнення результатів оцінки – на даному етапі вибирається усереднені значення оцінки.

Для оцінки моделей були використані такі метрики оцінки моделей:

Accuracy (3.1) – точність є хорошим базовим показником для вимірювання продуктивності моделі. Недоліком простої точності є те, що точність добре працює у збалансованих наборах даних. Однак у незбалансованих наборах даних точність стає гіршим показником.

Recall (повнота) – справжній позитивний показник – це міра того, скільки істинних позитивів буде передбачено з усіх позитивів у наборі даних. Іноді її також називають чутливістю.

$$Recall = (TP)/(TP+FN) \quad (3.3)$$

Precision (влучність) – дана метрика є мірою правильності позитивного прогнозу. Іншими словами, це означає, що якщо результат прогнозується як позитивний, наскільки ви впевнені, що він насправді позитивний.

$$Precision = (TP)/(TP+FP) \quad (3.4)$$

F-score або *F1-score* – дана метрика є способом поєднання влучності (precision) і повноти (recall) моделі. Він визначається як середнє гармонійне значення влучності і повноти моделі. Поодинці ні повнота, ні влучність не дозволяє добре оцінити модель. Ми можемо мати високе значення влучності з низьким значенням повноти або, навпаки. F-міра дає можливість виразити обидві метрики за допомогою одного показника. *F-score* визначається за наступною формулою:

$$F\text{-}score = (2 * Precision * Recall) / (Precision + Recall) \quad (3.5)$$

Для класифікації було вибрано і реалізовано наступні популярні нейронні мережі в області прогнозування дефектності програмного забезпечення:

1. Багатошаровий перцептрон (MLP)

Багатошаровий перцептрон (MLP) є доповненням до нейронної мережі прямого зв’язку. Він складається з трьох типів шарів - вхідного шару, вихідного шару та прихованого шару. Вхідний шар отримує вхідний сигнал для обробки. Потрібне завдання, таке як передбачення та класифікація, виконується вихідним шаром. Довільна кількість прихованих шарів, які розміщуються між вхідним і вихідним шарами є справжньою обчислювальною машиною MLP. Подібно до мережі прямого зв'язку в MLP, дані переходять у прямому напрямку від рівня входу до вихідного рівня. Нейрони в MLP тренуються за допомогою алгоритму навчання зворотного поширення. MLP призначені для апроксимації будь-якої

безперервної функції і можуть розв'язувати задачі, які не є лінійно роздільними [95].

Використовуючи мову програмування Python і відкриту нейромережну бібліотеку Keras було реалізовано MLP мережу, яка складається із 1-го вхідного шару, 2-х прихованих і 1-го вихідного шару (рис. 3.15). За допомогою функції GridSearchCV було визначено гіперпараметри для моделі: *batch* = 64; *epochs* = 5; *optimizer* = RMSprop; *neurons* = 50; *activation* = relu.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau


mlnn_model = Sequential()
mlnn_model.add(Dense(units=50, input_dim= 21, kernel_initializer='normal', activation='relu'))
mlnn_model.add(Dense(units=50, kernel_initializer='normal', activation='relu'))
mlnn_model.add(Dense(units=1, kernel_initializer='normal', activation='relu'))
mlnn_model.compile(optimizer='RMSprop', loss='binary_crossentropy', metrics=['accuracy'])

print(mlnn_model.summary())

mlnn_model.fit(X_train,y_train,batch_size=64, epochs = 5)
scores = mlnn_model.evaluate(X_test, y_test)
print("All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'")
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Рис. 3.15. Реалізація MLP нейронної мережі за допомогою мови програмування Python і бібліотеки Keras

Значення метрик оцінки результатів класифікації для методу MLP наведено у таблиці 3.12.

Таблиця 3.12 – Значення метрик Accuracy, Recall, Precision, F-score для MLP класифікатора для всіх метрик і важливих метрик

Метрики	Accuracy	Recall	Precision	F-score
Важливі метрики (7 метрик)	0.829	0.797	0.883	0.837
Всі метрики (21-а метрика)	0.704	0.637	0.794	0.728

2. Radial Basis Function (RBF) Networks (Мережі радіальної базисної функції)

RBF подібні до MLP з трьома шарами (вхідний, середній або «прихований» шар і вихід). Також, як і MLP, RBF можуть легко моделювати будь-яку нелінійну функцію. Основна відмінність між двома мережами полягає в тому, що RBF не вводить необроблені вхідні дані, а передає міру відстані від вхідних даних до

прихованого шару. Ця відстань вимірюється від деякого центрального значення в діапазоні змінної (іноді середнього) до заданого вхідного значення [96].

Реалізація даної нейронної мережі представлена на рисунку 3.16. За допомогою GridSearchCV було визначено гіперпараметри для моделі: *batch* = 128; *epochs* = 5; *optimizer* = SGD; *neurons* = 50; *activation* = sigmoid.

```
from keras.layers import Dense, Flatten
from keras.models import Sequential
from keras.losses import binary_crossentropy

rbfnn_model = Sequential()
rbfnn_model.add(Flatten(input_shape=(7, 1)))
rbfnn_model.add(RBFLayer(50, 0.5))
rbfnn_model.add(Dense(1, activation='sigmoid'))

rbfnn_model.compile(optimizer='SGD', loss='binary_crossentropy', metrics=['accuracy'])

rbfnn_model.fit(X_test, y_test, batch_size=128, epochs=10)

rbfnn_model.fit(X_train,y_train,batch_size=64, epochs = 5)
scores = rbfnn_model.evaluate(X_test, y_test)
print("All features: ")
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Рис. 3.16. Реалізація RBF нейронної мережі за допомогою мови програмування Python і бібліотеки Keras

Значення метрик оцінки результатів класифікації для методу RBF наведено у таблиці 3.13.

Таблиця 3.13 – Значення метрик Accuracy, Recall, Precision, F-score для RBF класифікатора для всіх метрик і важливих метрик

Метрики	Accuracy	Recall	Precision	F-score
Важливі метрики (7 метрик)	0.877	0.915	0.852	0.882
Всі метрики (21-а метрика)	0.747	0.709	0.831	0.765

3. Simple Recurrent Neural Networks (Рекурентна нейронна мережа)

Рекурентна нейронна мережа (RNN) – це особливий тип штучної нейронної мережі, пристосованої для роботи з даними часових рядів або даними, які включають послідовності. Звичайні нейронні мережі прямого зв’язку призначені лише для точок даних, які не залежать одна від одної. RNN називаються рекурентними, оскільки виконують одну і ту ж функцію для кожного введення даних, тоді як вихід поточного входу залежить від одного минулого обчислення..

RNN мають концепцію «пам’яті», яка допомагає їм зберігати стани або інформацію попередніх входів для створення наступного виходу послідовності. В інших нейронних мережах всі входи незалежні один від одного. Але в RNN всі вхідні дані пов’язані один з одним [97].

RNN має низку переваг, таких як:

- здатність обробляти послідовності даних;
- здатність обробляти вхідні дані різної довжини;
- здатність зберігати або «запам’ятовувати» історичну інформацію.

Недоліками є:

- обчислення може бути дуже повільним;
- мережа не враховує майбутні вхідні дані для прийняття рішень;
- проблема зникаючого градієнта, коли градієнти, які використовуються для оновлення ваги, можуть наблизитися до нуля, заважаючи мережі вивчати нові ваги. Чим глибше мережа, тим більш виражена ця проблема.

Використовуючи мову програмування Python і бібліотеку Keras було реалізовано Simple RNN мережу, яка складається із 1-го вхідного шару, 1-го прихованого рекурентного і 1-го вихідного шару (рис. 3.17). За допомогою функції GridSearchCV було визначено гіперпараметри для моделі: *batch* = 256; *epochs* = 50; *optimizer* = Adam; *neurons* = 20; *activation* = tanh.

```
from pandas import read_csv
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN


rnn_model = Sequential()
rnn_model.add(SimpleRNN(20, input_shape=(7,1), activation = 'tanh' , use_bias=True,  kernel_initializer='glorot_uniform'))
rnn_model.add(Dense(units = 1, activation = 'tanh'))
rnn_model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
rnn_model.fit(X_train, y_train, epochs=50, batch_size=256, verbose=2)

rnn_model.fit(X_train,y_train,batch_size=256, epochs = 50)
scores = rnn_model.evaluate(X_test, y_test)
print("All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'")
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Рис. 3.17. Реалізація RNN нейронної мережі за допомогою мови програмування Python і бібліотеки Keras

Значення метрик оцінки результатів класифікації для методу RNN наведено у таблиці 3.14.

Таблиця 3.14 – Значення метрик Accuracy, Recall, Precision, F-score для RNN класифікатора для всіх метрик і важливих метрик

Метрики	Accuracy	Recall	Precision	F-score
Важливі метрики (7 метрик)	0.883	0.857	0.922	0.888
Всі метрики (21-а метрика)	0.779	0.745	0.849	0.793

4. LSTM Neural Networks

Довга короткочасна пам'ять (Long short-term memory; LSTM) – особливий різновид архітектури рекурентних нейронних мереж, здатна до навчання довгострокових залежностей. Вони чудово вирішують цілу низку різноманітних завдань і нині широко використовуються. LSTM розроблено спеціально, щоб уникнути проблеми довготривалої залежності. Запам'ятовування інформації на довгі періоди часу – це їхня звичайна поведінка, а не щось, чому вони намагаються навчитися. LSTM також нагадує ланцюжок, але модулі виглядають інакше. Замість одного шару нейронної мережі вони містять чотири, і ці шари взаємодіють особливим чином. Тут вирішується проблема зникаючого градієнта RNN. LSTM добре підходить для класифікації, обробки та прогнозування часових рядів [97].

Використовуючи бібліотеку Keras було реалізовано LSTM мережу, яка складається із 1-го вхідного шару, 3-х прихованих LSTM і 1-го вихідного шару (рис. 3.18). За допомогою функції GridSearchCV було визначено гіперпараметри для моделі: *batch* = 128; *epochs* = 20; *optimizer* = SGD; *neurons* = 20; *activation* = tanh.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, LSTM,LeakyReLU
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau


lstm_model = Sequential()
lstm_model.add(LSTM(30, input_shape=(7, 1), return_sequences=True))
lstm_model.add(LeakyReLU( alpha=0.3))
lstm_model.add(LSTM(15, return_sequences=True))
lstm_model.add(LeakyReLU( alpha=0.3))
lstm_model.add(LSTM(30, return_sequences=False))
lstm_model.add(Dense(1, activation='tanh'))

lstm_model.compile(optimizer='SGD', loss='binary_crossentropy', metrics=['accuracy'])

lstm_model.fit(X_train,y_train,batch_size=128, epochs = 20)

scores = lstm_model.evaluate(X_test, y_test)
print("Important features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'")
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Рис. 3.18. Реалізація LSTM нейронної мережі

Значення метрик оцінки результатів класифікації для методу LSTM наведено у таблиці 3.15.

Таблиця 3.15 – Значення метрик Accuracy, Recall, Precision, F-score для LSTM класифікатора для всіх метрик і важливих метрик

Метрики	Accuracy	Recall	Precision	F-score
Важливі метрики (7 метрик)	0.907	0.892	0.928	0.909
Всі метрики (21-а метрика)	0.805	0.766	0.877	0.818

Отже, було використано і реалізовано чотири нейронні мережі, які добре проявили себе в класифікації модулів на дефектні і такі що без дефектів (табл. 3.16). Три з чотирьох моделей дали кращі результати, ніж моделі машинного навчання описані в розділі 3.3.

Таблиця 3.16 – Оцінка класифікації з використанням нейронних мереж

НМ	Всі метрики (21-а метрика)				Важливі метрики (7 метрик)			
	Accuracy	Recall	Precision	F1	Accuracy	Recall	Precision	F1
MLP	0.704	0.673	0.794	0.728	0.829	0.797	0.883	0.837
RBF	0.747	0.709	0.831	0.765	0.8772	0.915	0.852	0.882
RNN	0.779	0.745	0.849	0.793	0.883	0.857	0.922	0.888
LSTM	0.805	0.766	0.877	0.818	0.907	0.892	0.928	0.909

Для покращення результатів прогнозування, було використано стекове ансамблювання нейронних мереж, яке часто дає можливість підвищити точність прогнозу моделей машинного навчання, особливо у випадку різнорідних даних, а також дає можливість розпаралелити отриману модель, що підвищує швидкість обчислень.

Ансамблеве навчання – це парадигма машинного навчання, де кілька моделей (називаються «слабкими учнями») вчаться вирішувати одну й ту саму проблему та об’єднуються для отримання кращих результатів. Основна гіпотеза полягає в тому, що якщо слабкі моделі правильно комбінувати, можна отримати більш точні та/або надійні моделі [98].

На основі розглянутих нейронних мереж було вирішено створити стековий ансамбль (Stacking Ensemble), який в якості мета моделі (супервайзера) використовує логістичну регресію LogisticRegression і складається із трьох нейронних мереж RBF, RNN і LSTM (рис. 3.19). Ідея стекування полягає в тому, щоб вивчити кілька різних слабких учнів і об’єднати їх шляхом навчання мета-моделі для виведення прогнозів на основі множинних передбачень, які повертають ці «слабкі» моделі.

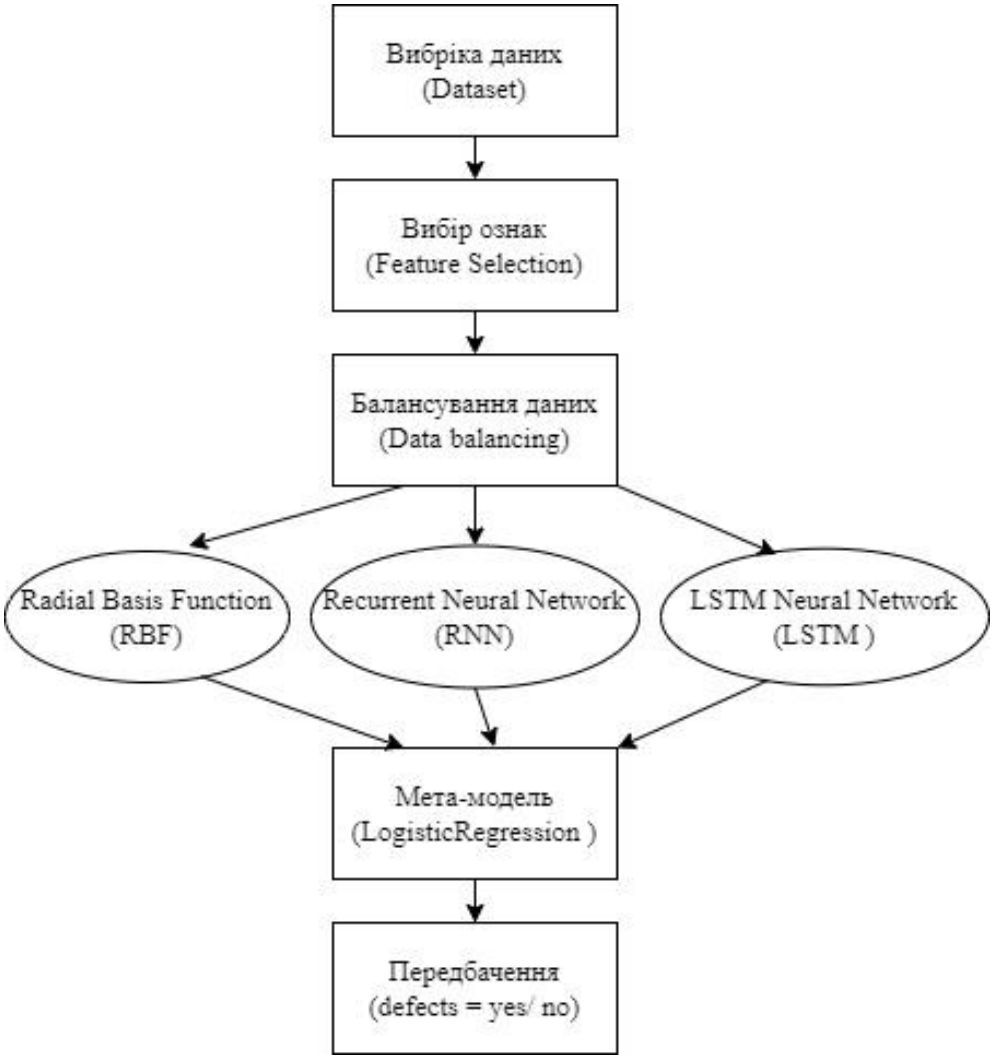


Рис. 3.19. Схема роботи стекового ансамбля із трьох нейронних мереж

За допомогою компонента StackingClassifier бібліотеки Sklearn і мови програмування Python було реалізовано стековий ансамбль (рис. 3.20).

```
rbfnn = KerasClassifier(build_fn=get_rbfnn_model,batch_size=256, epochs=10)
rbfnn._estimator_type = "classifier"
rbfnn_pipe = Pipeline([('scaler', scaler), ('rbfnn', rbfnn)])

rnn = KerasClassifier(build_fn=get_rnn_model, epochs=50, batch_size=256)
rnn._estimator_type = "classifier"
rnn_pipe = Pipeline([('scaler', scaler), ('rnn', rnn)])

lstm = KerasClassifier(build_fn=get_lstm_model, epochs=50, batch_size=256)
lstm._estimator_type = "classifier"
lstm_pipe = Pipeline([('scaler', scaler), ('lstm', lstm)])


# Make an ensemble
ensemble = StackingClassifier(estimators=[
                                    (('rnn', rnn_pipe),('rbfnn', rbfnn_pipe),('lstm', lstm_pipe)],
                              final_estimator=LogisticRegression())

print("All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'")
print("Accuracy: %.2f%%" % (ensemble.fit(X_train, y_train).score(X_test, y_test)))
```

Рис. 3.20. Реалізація стекового ансамблю із чотирьох нейронних мереж за допомогою мови програмування Python і бібліотеки StackingClassifier

Даний ансамбль показав хороші результати прогнозування дефектності системи, результати наведено на рисунку 3.24, і видно що при класифікації з важливими ознаками точність прогнозу становить 92.03%, дана точність є вищою, ніж точності, які були при класифікації із використанням окремих нейронних мереж.

Таблиця 3.17 – Значення метрик Accuracy, Recall, Precision, F-score стекового ансамблю для всіх метрик і важливих метрик

Метрики	Accuracy	Recall	Precision	F-score
Важливі метрики (7 метрик)	0.920	0.907	0.935	0.921
Всі метрики (21-а метрика)	0.819	0.784	0.881	0.830

Результати, отримані із використанням ансамблювання нейронних мереж було порівняно із існуючими дослідженнями в області прогнозування дефектів ПЗ на основі метрик коду. У статті [99] розглянуто дві моделі Stack Sparse Auto Encoder (SSAE) і Deep Belief Network (DBN), які використовуються для класифікації наборів даних з репозиторію NASA. Згідно з проведеним експериментом модель SSAE показує кращі результати точності порівняно з моделлю DBN і вона становить 90%.

В дослідженні [100] використовують вибірку даних з проєкту JM1 з репозиторію, який ми використовували у дослідженні. Для підвищення точності класифікації автори пропонують використання самоорганізуючих карт з ієрархічною кластеризацією та попередньою обробкою даних, при якому можна отримати точність прогнозування 86%.

Також, якщо враховувати наше попереднє дослідження (дивитись розд. 3.3) із використанням класифікаторів: Random forest, Support vector machine, K-nearest neighbor, Decision tree classifier, AdaBoost classifier, Gradient Boosting. За допомогою цих класифікаторів було отримано точність прогнозування в межах 79.8% - 86.0%. Тому, можна зробити висновок, що використання методів на основі нейронних мереж дозволяє покращити точність прогнозування дефектів програмного забезпечення. Використання розробленого ансамблю дозволяє прогнозувати дефектність модулів в програмному забезпеченні із точністю до 92,03%.

Питання міжпроєктного прогнозування дефектів за результатами, отриманими в даному розділі, є складним науковим завданням і потребує подальшого дослідження. Однак, оскільки для досліджень використовувалися різні проєкти, а метрики програмного коду широко прийняті та широко використовуються в розробці програмного забезпечення, можна вважати, що метрики коду, на які найбільше впливають на дефектність програмного забезпечення, є універсальними і можуть бути поширені на інші проекти програмного забезпечення.

3.5. Висновки до розділу 3

У даному розділі було проведено дослідження, метою якого було удосконалення моделей прогнозування дефектності ПЗ шляхом використання методів машинного навчання для вибору метрик, що найбільше впливають на дефектність модулів ПЗ і розроблення методу класифікації дефектів ПЗ на основі використанням стекового ансамблю нейронних мереж.

Для досліджень використано об’єднаний з датасетів KC1, KC2, PC1, CM1, JM1 датасет з репозиторію PROMISE Software Engineering, який містив дані про тестування програмних модулів та 21 метрику коду. Методами Boruta, Step-wise selection, Exhaustive Feature Selection, Random Forest Importance, LightGBM Importance, Genetic Algorithms, Principal Component Analysis, Xverse python здійснено вибір найважливіших ознак, які впливають на якість і надійність програмного коду. На основі голосування за результатами роботи методів вибору ознак з використанням логістичної регресії побудовано модель прогнозування дефектності ПЗ, яка встановлює взаємозв’язок між ймовірністю появи дефекту в програмному модулі та метриками його коду. Показано, що в цю модель входять такі метрики коду, як кількість рядків коду за МакКейбом (loc), загальна кількість операндів і операторів (N) , функціональність модуля (I), обсяг на мінімальному виконанні (V), зусилля для написання модуля (E), цикломатична складність за МакКейбом (v(g)), кількість гілок в репозиторії, в яких змінювався модуль (branchCount), які мають найбільший вплив на надійність програмного модуля. Приріст точності прогнозування дефектних програмних модулів у випадку використання побудованої моделі становить 10-21% (з 61.7-72.5% до 80.0- 85.5%) порівняно з використанням усього набору метрик.

Здійснено порівняння ефективності роботи різних методів вибору ознак, зокрема проведено дослідження впливу методу вибору ознак на точність класифікації. Показано, що використання будь-якого методу вибору ознак підвищує точність класифікації принаймні на 10%, у порівнянні з початковим датасетом, що підтверджує важливість цієї процедури для прогнозування дефектів ПЗ на основі метричних датасетів, що містять значну кількість метрик, які корелюють між собою.

Також для покращення точності прогнозування було використано і реалізовано чотири нейронні мережі, а саме: багатошаровий перцептрон, нейронна мережа на основі радіально-базисних функцій, рекурентна нейронна мережа та мережа довгої короткочасної пам’яті. Оптимальні параметри архітектури кожної нейронної мережі визначали методом GridSearch за

допомогою програмної реалізації мовою Python. Для оцінки класифікації за допомогою вибраних методів було використано метод перехресної перевірки – стратифікована k-кратна перехресна перевірка. Для оцінки моделей було використано такі метрики оцінки: Accuracy, Recall, Precision, F-score. Найкращої точності прогнозування було досягнуто при використанні моделей на основі трьох останніх мереж: нейронної мережі на основі радіально-базисних функцій, рекурентної нейронної мережі та мережі довгої короткочасної пам’яті. Використання розробленого методу класифікації ПЗ, який передбачає зменшення множини ознак до семи найважливіших, дає можливість збільшити точність прогнозування зазначеними методами до 87,72% (для RBF), 88,34% (для RNN) та 90,78% (у випадку LSTM).

На основі трьох нейронних мереж, які дали найвищі результати у дослідженні, було створено стековий ансамбль, який в якості мета моделі (супервайзера) використовує логістичну регресію. Використання такого ансамблю дало можливість збільшити точність прогнозування дефектності ПЗ до 92,03% у випадку використання семи найважливіших метрик коду, і до 81,95% у випадку використання усієї сукупності ознак.

РОЗДІЛ 4. ЗАСОБИ АВТОМАТИЗОВАНОГО АНАЛІЗУ НАДІЙНОСТІ СКЛАДНИХ ТЕХНІЧНИХ СИСТЕМ

Для автоматизованого використання, тестування і верифікації розроблених методів визначення функції працездатності для Марковських моделей надійності та подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами було спроектовано та розроблене відповідне програмне забезпечення. Дане програмне забезпечення дає змогу автоматизувати процес використання розроблених методів, зменшує вплив людського фактора, і, відповідно, знижує ймовірність внесення помилки на при використанні даних методів і моделей.

В даному розділі представлено опис функціональних можливостей і тестування розробленого програмного забезпечення.

4.1. Програмне забезпечення для автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами

Для автоматизації роботи методу подання Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами та верифікації результатів його роботи було розроблено відповідне програмне забезпечення. Дане ПЗ дозволяє задати інформацію про початковий (вхідний) граф і на його основі будувати граф вищого порядку з довільним значенням параметра порядку.

Програмне забезпечення розроблене як Web-аплікація із використанням мови програмування JavaScript і відкритої бібліотеки ReactJS для розробки користувацьких інтерфейсів. Для побудови та візуалізації графа станів і переходів було використано бібліотеку react-d3-graph – бібліотека, яка наділена зручним і швидким інтерфейсом для побудови, візуалізації та маніпуляцій з графами [114].

Дане програмне забезпечення виконує покрокове представлення Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами.

На першому кроці (рис. 4.1) користувач повинен ввести інформацію про порядок Марковського процесу вищого порядку, який він хоче змоделювати, та кількість вузлів у графі першого порядку, який буде початковим для побудови графа вищого порядку. Також можливе введення назви вузлів у поле «*Назви вузлів у графі першого порядку (*)*» (перелічення здійснюється через кому і кількість значень має відповідати значенню введеному в поле «*Кількість вузлів у графі першого порядку*») для вхідного графа, дана можливість була внесена для зручності користувача, якщо він працює із початковим графом, який вже має визначені певні назви для вузлів (при візуалізації і побудові графа вищого порядку будуть використовуватись внесені назви вузлів). Дане поле не є обов'язковим і значення можна не вводити, тоді іменування вузлів буде виконуватись за шаблоном S{i}, де $i=1...N$, N – кількість вузлів початкового графа.

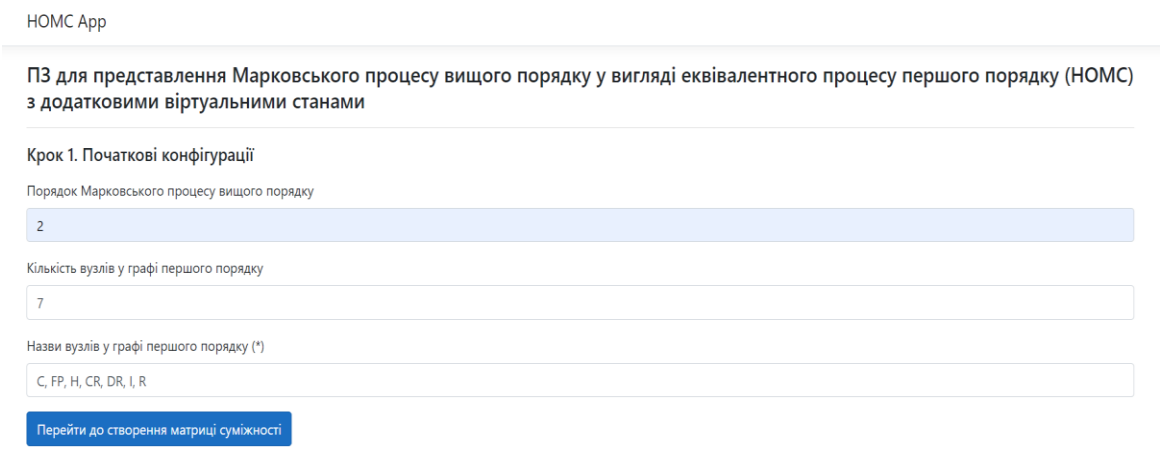


Рис. 4.1. Крок початкових конфігурацій у розробленому ПЗ

Після того, як були внесені початкові конфігурації для моделювання процесу вищого порядку потрібно перейти до наступного кроку, натиснувши кнопку «Перейти до створення матриці суміжності». На наступному кроці користувачу дається можливість заповнити матрицю суміжності для вхідного графа. На основі значень, які були внесені на попередньому кроці про кількість вузлів у початковому графі та назв вузлів у графі (це значення є необов'язковим) будується матриця суміжності, значення елементів якої заповнюється нулями (0). Користувачу потрібно ввести інформацію про те, які є взаємодії (ребра) між вузлами в графі. Для цього потрібно навести фокус миші на перетині відповідних

вузлів і ввести значення 1 (1 означає що між вибраними вузлами графа є зв’язок (ребро)).

На рисунках 4.2 і 4.3 наведені приклади побудови матриці суміжності із вказаними назвами вузлів графа і без назв вузлів відповідно.


Крок 2. Заповніть матрицю суміжності для графа першого порядку

i-j	C	FP	H	CR	DR	I	R
C	0	0	0	1	0	1	0
FP	0	0	0	1	1	1	0
H	0	0	0	1	0	1	0
CR	1	1	1	0	0	0	0
DR	0	1	0	0	0	0	0
I	0	0	0	0	0	0	1
R	0	0	0	0	0	1	1

Згенерувати граф Маркова першого порядку

Рис. 4.2. Приклади побудови матриці суміжності із вказаними назвами вузлів графа


Крок 2. Заповніть матрицю суміжності для графа першого порядку

i-j	S1	S2	S3	S4	S5	S6	S7
S1	0	0	0	0	0	0	0
S2	0	0	0	0	0	0	0
S3	0	0	0	0	0	0	0
S4	0	0	0	0	0	0	0
S5	0	0	0	0	0	0	0
S6	0	0	0	0	0	0	0
S7	0	0	0	0	0	0	0

Згенерувати граф Маркова першого порядку

Рис. 4.3. Приклади побудови матриці суміжності без вказаних назв вузлів графа

Наступним кроком після побудови матриці суміжності є візуалізація графа Маркова першого порядку на основі матриці суміжності, внесеної на кроці 2. Даний крок є зручним для представлення графа, який є вхідним у даному моделюванні (цей крок зручний, якщо проєктант з вхідних даних має тільки матрицю суміжності, даний крок допоможе швидко візуалізувати вхідний граф на основі матриці).

Візуалізація графа станів і переходів відбувається за допомогою бібліотеки react-d3-graph (вхідними даними для даної бібліотеки є структура даних, що є

близькою за формою до матриці суміжності), яка швидко дозволяє візуалізувати та проводити маніпуляції із графом станів і переходів (рис. 4.4).

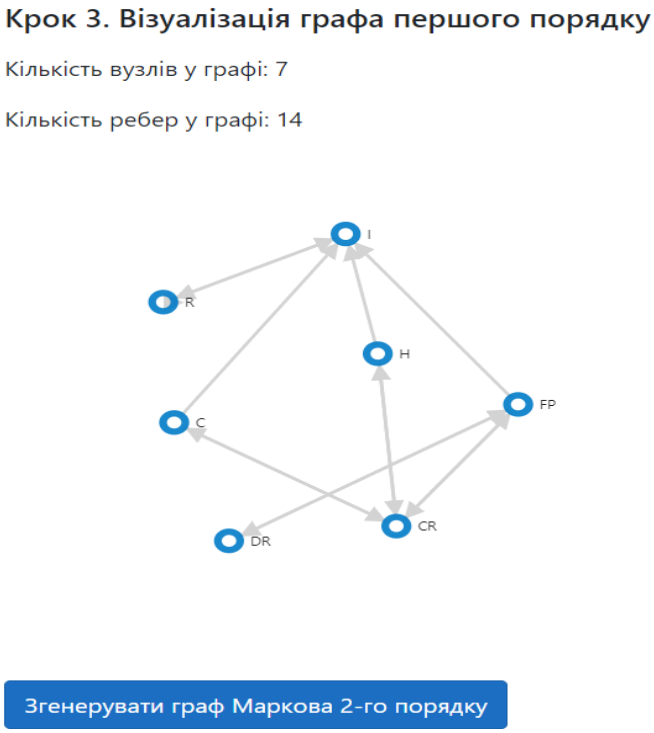


Рис. 4.4. Візуалізація графа першого порядку на основі матриці суміжності

Можливості бібліотеки react-d3-graph дозволяють «розумно» розташовувати вузли при візуалізації (залежно від кількості вузлів і зв'язків між ними), також можна робити ручні переміщення вузлів, збільшувати або зменшувати масштаб, виділяти певні лінії на графі із підсвіченням, виділяти певні вузли на графі із підсвіченням його зв'язків з іншими вузлами графа (рис. 4.5).

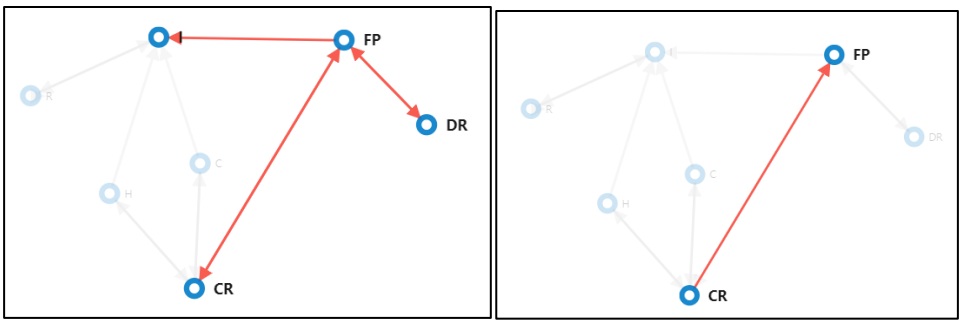


Рис. 4.5. Виділення елементів на графі при наведенні на них

Наступним етапом є безпосереднє представлення Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами (рис. 4.6).

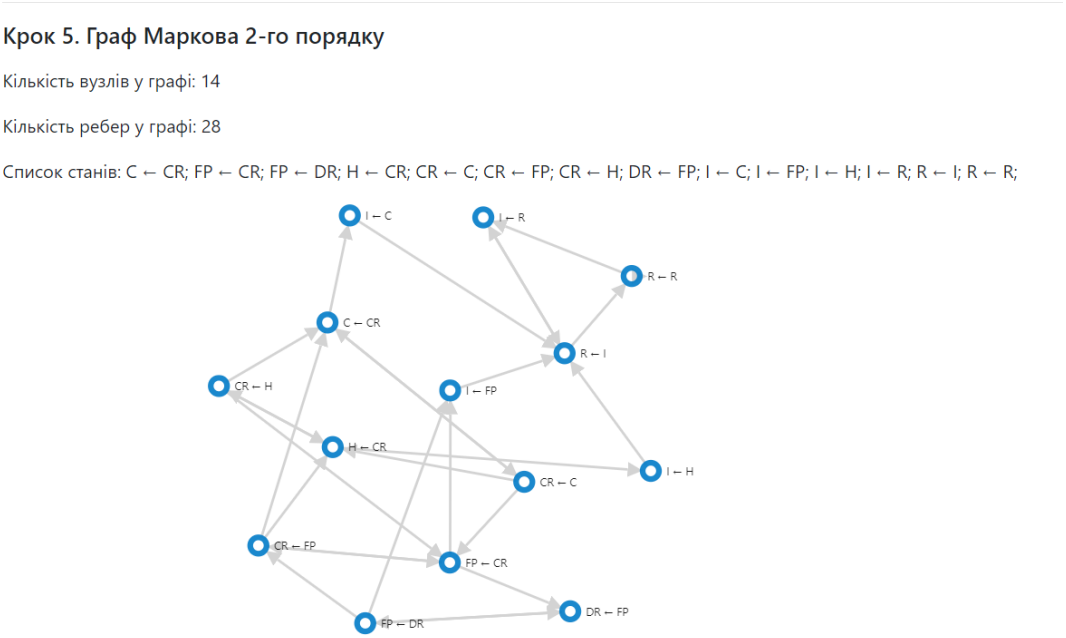


Рис. 4.6. Представлення Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами

На даному кроці можна побачити інформацію про кількість вузлів і ребер у створеному графі станів і переходів, а також список нових станів. Візуалізація графа відбувається за допомогою бібліотеки react-d3-graph і тут можна проводити всі вище перелічені маніпуляції із графом.

Відповідно для даного графа на рисунку 4.6 будується матриця суміжності (рис. 4.7).

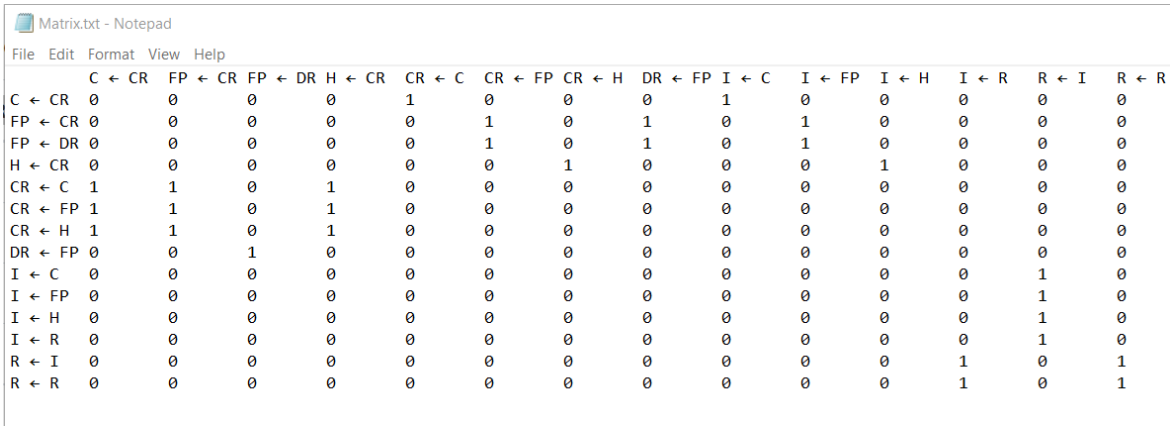
Матриця суміжності для граф Маркова 2-го порядку

i-j	C ← CR	FP ← CR	FP ← DR	H ← CR	CR ← C	CR ← FP	CR ← H	DR ← FP	I ← C	I ← FP	I ← H	I ← R	R ← I	R ← R
C ← CR	0	0	0	0	1	0	0	0	1	0	0	0	0	0
FP ← CR	0	0	0	0	0	1	0	1	0	1	0	0	0	0
FP ← DR	0	0	0	0	0	1	0	1	0	1	0	0	0	0
H ← CR	0	0	0	0	0	0	1	0	0	0	1	0	0	0
CR ← C	1	1	0	1	0	0	0	0	0	0	0	0	0	0
CR ← FP	1	1	0	1	0	0	0	0	0	0	0	0	0	0
CR ← H	1	1	0	1	0	0	0	0	0	0	0	0	0	0
DR ← FP	0	0	1	0	0	0	0	0	0	0	0	0	0	0
I ← C	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I ← FP	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I ← H	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I ← R	0	0	0	0	0	0	0	0	0	0	0	0	1	0
R ← I	0	0	0	0	0	0	0	0	0	0	0	1	0	1
R ← R	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Зберегти матрицю у файл

Рис. 4.7. Матриця суміжності для графа вищого порядку

Дану матрицю суміжності можна зберегти у зручному форматі у текстовий файл, це зручний спосіб зберегти дані для подальших досліджень (рис. 4.8)


Matrix.txt - Notepad

File Edit Format View Help

	C ← CR	FP ← CR	FP ← DR	H ← CR	CR ← C	CR ← FP	CR ← H	DR ← FP	I ← C	I ← FP	I ← H	I ← R	R ← I	R ← R
C ← CR	0	0	0	0	1	0	0	0	1	0	0	0	0	0
FP ← CR	0	0	0	0	0	1	0	1	0	1	0	0	0	0
FP ← DR	0	0	0	0	0	1	0	1	0	1	0	0	0	0
H ← CR	0	0	0	0	0	0	1	0	0	0	1	0	0	0
CR ← C	1	1	0	1	0	0	0	0	0	0	0	0	0	0
CR ← FP	1	1	0	1	0	0	0	0	0	0	0	0	0	0
CR ← H	1	1	0	1	0	0	0	0	0	0	0	0	0	0
DR ← FP	0	0	1	0	0	0	0	0	0	0	0	0	0	0
I ← C	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I ← FP	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I ← H	0	0	0	0	0	0	0	0	0	0	0	0	1	0
I ← R	0	0	0	0	0	0	0	0	0	0	0	0	1	0
R ← I	0	0	0	0	0	0	0	0	0	0	0	1	0	1
R ← R	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Рис. 4.8. Збережений файл із матрицею суміжності

Дане програмне забезпечення було протестоване на швидкодію, адже представлення Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами та візуалізація графа станів і переходів є ресурсозатратним процесом, а швидкодія у програмному забезпеченні даного типу є дуже важливою характеристикою, оскільки аналіз може виконуватись для графів, які складаються із великої кількості станів.

Тестування ПЗ проводили із використанням браузера Chrome v97 і на машині із наступними характеристиками: OS Windows 10, AMD Ryzen 7 4700U with Radeon Graphics 2.00 GHz, RAM 16 Гб.

При тестуванні розробленого програмного забезпечення розглядалася лише швидкодія програмної реалізації методу автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами без замірів часу візуалізації графа станів і переходів за допомогою бібліотеки react-d3-graph. Аналіз швидкодії для візуалізації графа станів і переходів не був виконаний через особливості роботи бібліотеки react-d3-graph, яка не має реалізованого механізму визначення часу візуалізації графа, оскільки вона приймає на вхід інформацію про вузли та зв’язки між ними і паралельно починає побудову вузлів і визначення масштабу графа (побудова вузлів і ребер графа відбувається відносно швидко, більше часу займає поступове

масштабування графа до потрібного розміру, проте результати є прийнятними з погляду швидкодії).

На першому етапі було використано вхідний граф, який складається із 8-ми вузлів і 15-ти ребер із поступовим збільшенням порядку Марковського процесу вищого порядку N=2…16 із кроком 2. (табл. 4.1).

Таблиця 4.1 – Аналіз швидкодії програмної реалізації представлення Марковського процесу вищого порядку при збільшенні порядку процесу

Порядок процесу	Кількість станів	Кількість ребер	Час роботи програмної реалізації методу, с
2	15	27	0.00007
4	48	449	0.00021
6	143	5 772	0.00139
8	409	58 909	0.00675
10	1 140	526 032	0.04612
12	3 123	4 329 524	0.35802
14	8453	33 801 657	2.74319
16	22 684	254 629 504	15.44469

На наступному етапі програмну реалізацію методу представлення Марковського процесу вищого порядку було протестовано для різної кількості вузлів у вхідному графі першого порядку із поступовим збільшенням їх кількості для моделювання процесів 2-го, 3-го і 4-го порядку. Кількість вузлів для тестування було вибрано на проміжку від 5 до 50 із кроком 5 із відповідним збільшенням кількості ребер у вхідному графі, так щоб один вузол мав щонайменше один вхідний і вихідний зв’язок.

У таблиці 4.2 наведено часові показники для визначення процесів 2-го, 3-го і 4-го порядку та інформація про кількість вузлів і ребер в початковому графі (графі першого порядку).

Із даних, які наведено у таблиці, можна зробити висновок, що програмна реалізація методу дозволяє швидко визначити граф вищого порядку для графа першого порядку із великою кількістю елементів, хоча даний час значно збільшується в залежності від порядку процесу.

Таблиця 4.2 – Аналіз швидкодії програмної реалізації представлення Марковського процесу вищого порядку при збільшенні кількості вузлів у вхідному графі для процесів 2-го, 3-го і 4-го порядку

Вузли (1-й порядок)	Ребра (1-й порядок)	Час визначення (2-й порядок), с	Час визначення (3-й порядок), с	Час визначення (4-й порядок), с
5	10	0.00014	0.00031	0.00015
10	20	0.00034	0.00041	0.00046
15	45	0.00038	0.00335	0.00608
20	80	0.00109	0.01284	0.03876
25	130	0.00187	0.02707	0.28466
30	170	0.00189	0.06489	1.04057
35	210	0.00258	0.09721	2.19348
40	260	0.00396	0.15297	4.14339
45	310	0.00552	0.20723	6.72101
50	370	0.00602	0.50723	7.98464

Відповідно до результатів швидкодії роботи програмної реалізації методу було продемонстровано залежності кількості станів і ребер у графі станів і переходів вищого порядку для процесів 2-го, 3-го і 4-го порядку. (рис. 4.9 і 4.10)

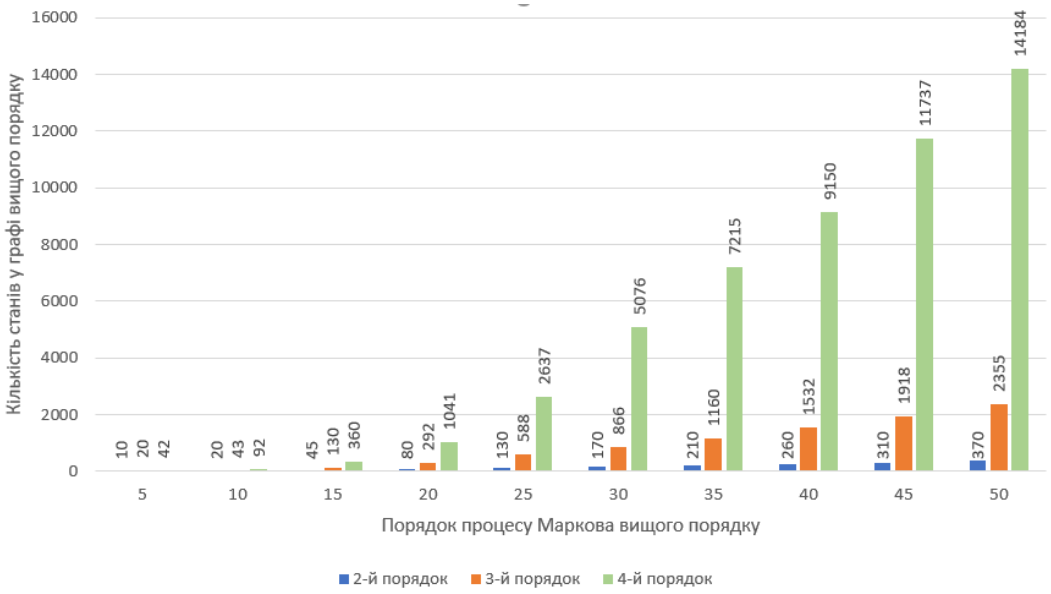


Рис. 4.9. Гістограма залежності кількості станів у графі вищого порядку для процесів 2-го, 3-го і 4-го порядку

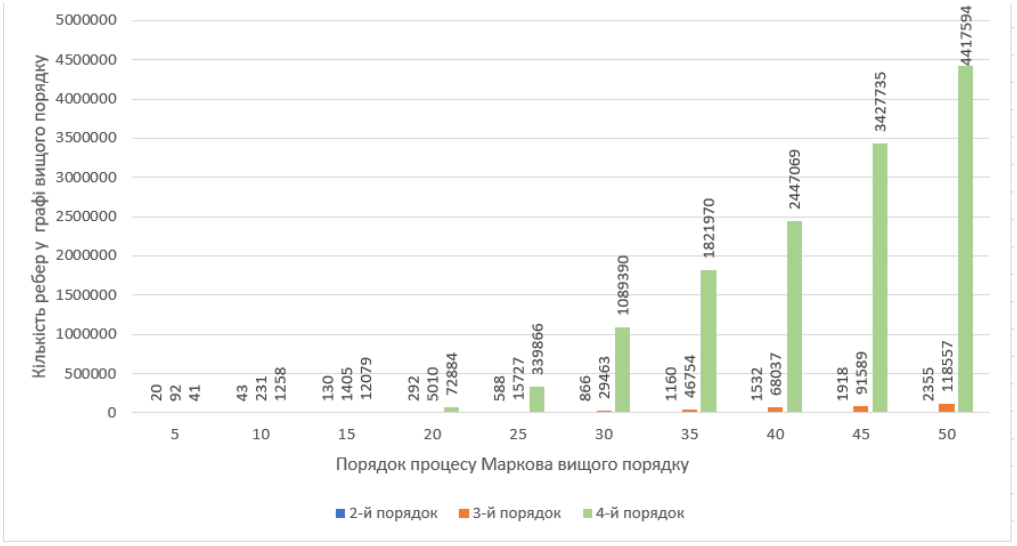


Рис. 4.10. Гістограма залежності кількості ребер у графі вищого порядку для процесів 2-го, 3-го і 4-го порядку

Проаналізувавши результати отримані при двох різних варіантах тестування можна зробити загальний висновок, що розроблене програмне забезпечення дає достатньо високий рівень швидкодії при вирішенні поставленого завдання. А якщо порівнювати результати наведені в таблицях 4.1 і 4.2, можна побачити, що більше затрат часу на визначення графа вищого порядку відбувається при збільшенні порядку процесу ніж від збільшення кількості вузлів у вхідному графі.

4.2. Процес визначення показників надійності із використанням функції працездатності та графа станів і переходів

В даному підрозділі описано процес визначення показників надійності із використанням функції працездатності та графа станів і переходів. Даний процес буде розроблений і реалізований у відповідному програмному забезпеченні, яке буде реалізовувати і використовувати метод автоматизованої побудови функції працездатності для Марковських моделей надійності.

Для аналізу показників надійності технічних систем традиційні Марковські моделі передбачають такі етапи:

Розроблення моделі об'єкта дослідження у вигляді графа станів і переходів.

Для розв'язання проблеми визначення станів і переходів між ними для опису функціонування системи, після попереднього аналізу предметної області, було реалізовано алгоритм, який використовує функцію працездатності та базується на послідовній імітації відмов і відновлень елементів системи.

Особливостями побудови графа станів і переходів у нашому випадку є те, що:

- нові стани, які додаються у множину станів системи після відновлення і стани, які мають такі ж самі робочі елементи, як і стан після відновлення розрізняються, як два різні стани (наприклад стан $1_0\,1_0\,1_0\,1_0$ і стан $1_0\,1_1\,1_0\,1_0$ це різні стани в множині станів (нижній індекс означає, що цей стан утворився після відмови і відновлення елемента системи));
- граф станів і переходів будується у вигляді дерева з верху вниз (тобто першим станом системи (верхній рівень) є стан, при якому всі елементи системи працюють і не відбувалось жодного відновлення, а наступні стани визначаються на основі функції працездатності та спроб перебору відмов і відновлень елементів системи).

Процес побудови графа станів і переходів із використанням умови працездатності складається із таких кроків (рис. 4.11):

Крок 1: Ініціалізація початкового стану системи. На даному кроці відбувається додавання стану при якому всі елементи системи працюють і не відбулося жодної відмови і відновлень. Ініціалізація першого рівня станів.

Крок 2: Імітація відмови/відновлення для стану. Відбувається прохід по всіх станах із вищого рівня і імітується відмова (у випадку, коли модуль системи є працездатний) або відновлення (у випадку, коли модуль є у стані відмови, але є можливість його відновлення) елементів.

Крок 3: Визначення типу нового стану. На цьому кроці визначається тип стану на основі функції працездатності і кількості можливих відновлень: робочий (якщо функція працездатності повертає результат *true* при певних встановлених значеннях функціонування елементів системи), простій (якщо умова працездатності повертає результат *false,* але хоча б один елемент системи може бути відновлений у працездатний стан), неробочий (якщо умова працездатності повертає результат *false* і немає більше відновлень для елементів).

Крок 4: Опрацювання стану із типом – «не робочий», при відмові. Якщо відбулась імітація відмови і новий стан є не робочим, даний стан вилучається з списку тимчасових станів і індекс елемента, який відмовив додається до списку елементів, які призводять до відмови в батьківському стані. Переходимо до кроку 7.

Крок 5: Опрацювання стану із типом – «робочий», при відмові або відновленні. Якщо відбулась імітація відмови і новий стан є робочим, тоді додаємо даний стан до списку тимчасових станів. Переходимо до кроку 7.

Крок 6: Видалення станів, що дублюються. На даному етапі відбувається перебір всіх станів, що були визначені для даного рівня, при якому визначаються стани, які дублюються в рівні і відбувається їхнє злиття. Також відбувається перевизначення батьківських елементів для злитого стану (на основі всіх батьківських елементів, які були у продубльованих станах).

Крок 7: Перевірка кінцевої умови пошуку станів. Якщо при проходженні по всіх станах із нижнього рівня і відбувається проходження по всіх станах, утворених на попередньому рівні, і якщо для жодного із елементів стану не можна

зробити відмову або відновлення, це означає, що було визначено всі можливі стани у системи, але якщо існує хоча б один такий стан, повертаємось на крок 2 і визначення нових станів.

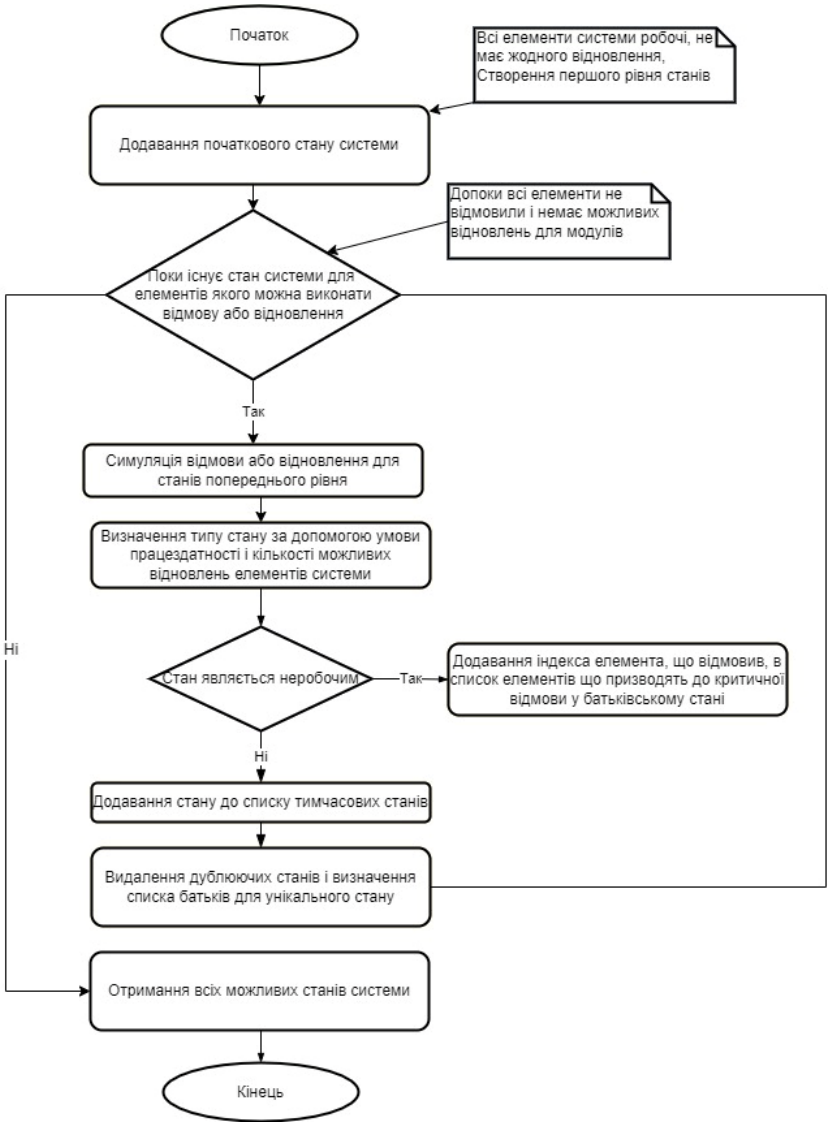


Рис. 4.11. Блок-схема процесу побудови графа станів і переходів на основі імітації послідовних відмов і відновлень елементів системи

Наступним етапом при аналізі показників надійності є **формування та розв'язання системи диференціальних рівнянь Колмогорова-Чепмена.**

Тому було описано процес, який дозволить побудувати систему диференціальних рівнянь Колмогорова-Чепмена для опису функціонування технічної системи та розв'язати цю систему диференціальних рівнянь за допомогою методу Рунге-Кутти (рис. 4.12). Вхідними даними для побудови системи рівнянь є множина станів і переходів для досліджуваної системи.

Процес складається з наступних кроків:

Крок 1. Проходимось по всіх станах системи та обробляємо кожен стан системи для того, щоб визначати рівняння (із системи диференціальних рівнянь) для поточного стану на основі інформації про вхідні і вихідні стани для поточного стану, який опрацьовується..

Крок 2. Підготовка вхідних даних для розв’язання системи диференціальних рівнянь Колмогорова-Чепмена із застосуванням методу Рунге-Кутти. Рівняння перетворюються до виду, необхідного для отримання розв’язку за допомогою модуля розв’язування диференціальних рівнянь.

Крок 3. Отримання і опрацювання розв’язку системи диференціальних рівнянь Колмогорова-Чепмена (на основі цих результатів можна визначати певні показники надійності системи).

Після розв’язання системи диференціальних рівнянь наступним і фінальним етапом є **визначення показників надійності** (таких як наприклад функція готовності або функція простою) відбувається на основі розподілу ймовірностей перебування в станах.

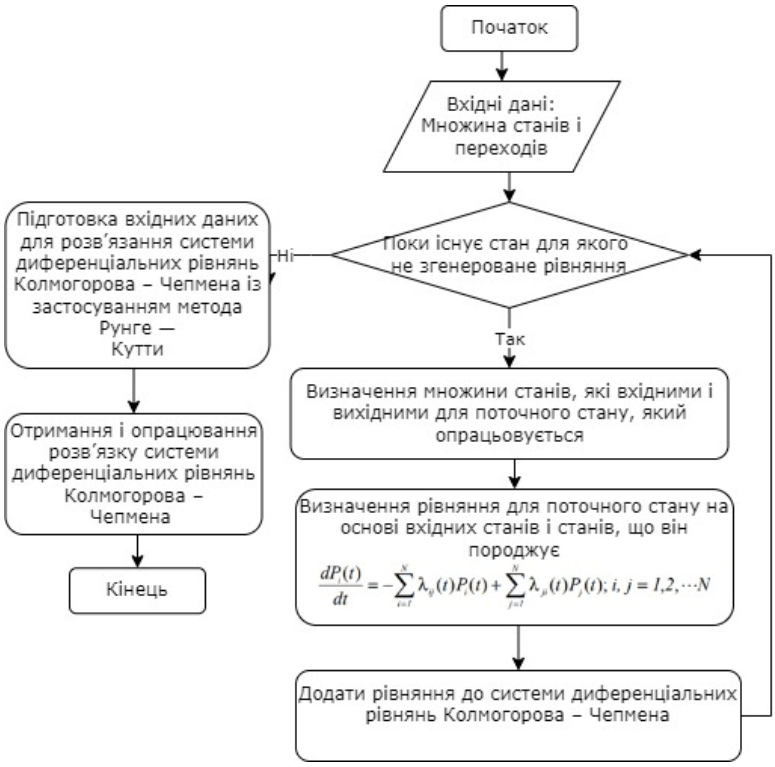


Рис. 4.12. Процес автоматизованої обробки системи диференціальних рівнянь Колмогорова-Чепмена.

4.3. Програмний засіб для розрахунку надійнісних характеристик складних технічних систем на основі функції працездатності

Для автоматизації роботи розробленого методу визначення функції працездатності та визначення показників надійності програмних систем на основі процесів описаних вище (див. розділ 4.2) було розроблено відповідне програмне забезпечення.

При розробленні архітектури програмного засобу був використаний модульний підхід проєктування, де кожен з модулів програми відповідає за виконання певної функції при аналізі надійнісних показників. Модульна структура комплексу робить його гнучким і дасть змогу швидко і без значних програмних змін додавати або модифікувати необхідні модулі (рис. 4.13).

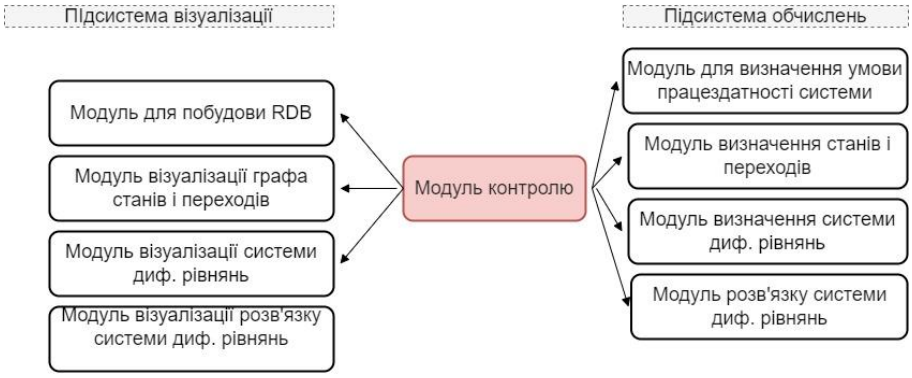


Рис. 4.13. Діаграма модулів програмного засобу

Програмний засіб містить один головний модуль, який відповідає за взаємозв’язок і передачу даних між всіма іншими модулями системи.

Модулі комплексу можна поділити на дві групи:

- модулі, які відповідають за опрацювання і визначення певних характеристик (функції працездатності, графа станів і переходів, визначення і розв’язку системи диференціальних рівнянь).
- модулі, які відповідають за візуалізацію інформації та налаштування графічних компонент.

Для реалізації функціональних можливостей комплексу було використано об’єктно-орієнтовану парадигму програмування і розроблено гнучку ієрархію класів (рис. 4.14). Вона дозволяє легко його модифікувати, додаючи новий функціонал та модулі.

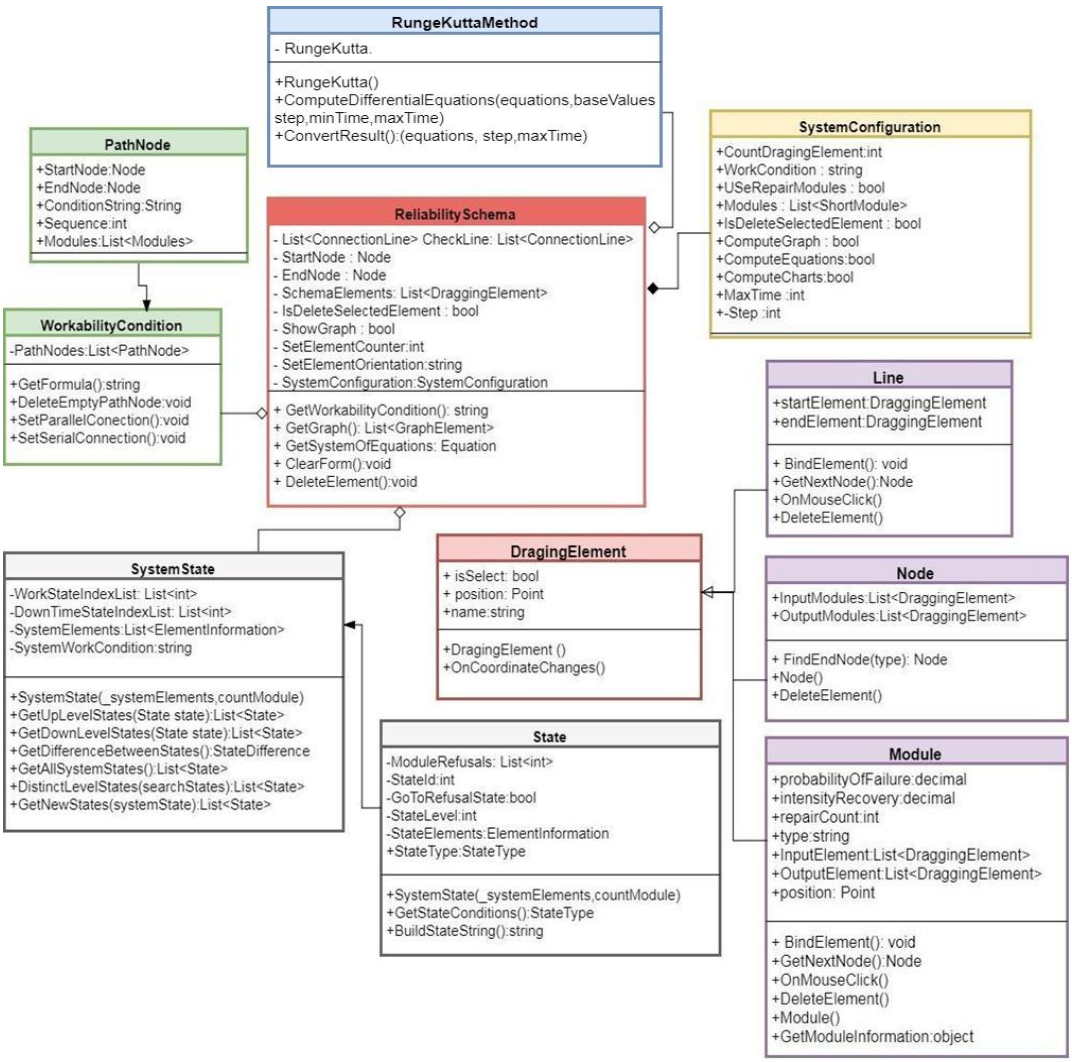


Рис. 4.14. Діаграма класів програмного засобу

Основним класом програмного комплексу є клас **ReliabilitySchema**. Він дозволяє виконувати маніпуляції в робочій області та в області налаштувань системи. Це головний клас, який відповідає за повне функціонування системи і його окремих модулів, які включені в нього. **WorkabilityCondition** – клас, який відповідає за визначення функції працездатності. **NodePath** – клас для опису сегментів схеми функціонування системи: початкового і кінцевого вузла сегмента; часткової функції працездатності; модулів сегмента; послідовність сегмента. **SystemState** – клас для визначення всіх можливих станів та переходів, в яких може перебувати досліджувана технічна система. **State –** клас для опису стану системи. Містить інформацію про тип, порядковий номер, елементи, які призводять до відмови. **SystemConfiguration** – статичний клас, який містить інформацію про налаштування системи і її характеристики.

Відповідно було розроблено програмне забезпечення в середовищі розробки Microsoft Visual Studio із використанням мови програмування C# і технології WPF для платформи Windows (програмний продукт працює на ОС Windows XP і вище). Початковим етапом роботи ПЗ є створення і конфігурація функціонування системи. За допомогою графічного конструктора ми можемо візуалізувати процес взаємодії між модулями технічної системи.

Головне вікно розробленого ПЗ (рис. 4.15) складається з робочої області (рис. 4.16) і області налаштувань (рис. 4.17).

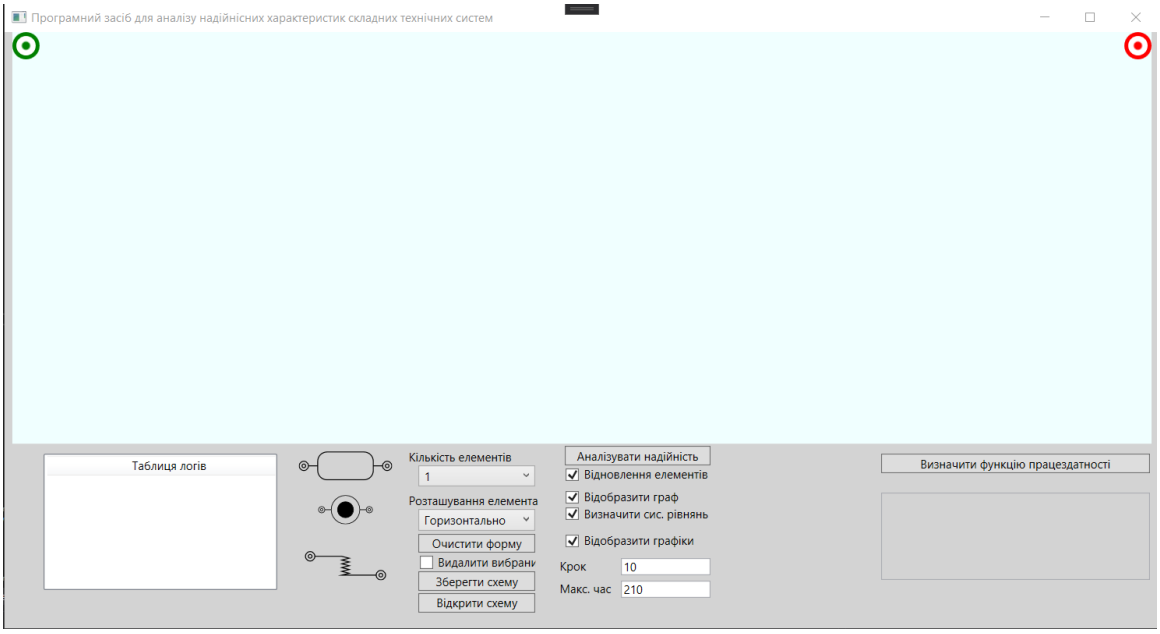


Рис. 4.15. Головне вікно програмного засобу

Робоча область призначена для візуалізації взаємодії модулів технічної системи. На початку і в кінці вікна розміщені вузли ініціалізації схеми взаємодії модулів системи, схема повинна починатись і завершуватись в цих точках. З'єднання елементів схеми лініями повинно відбуватися зліва на право.

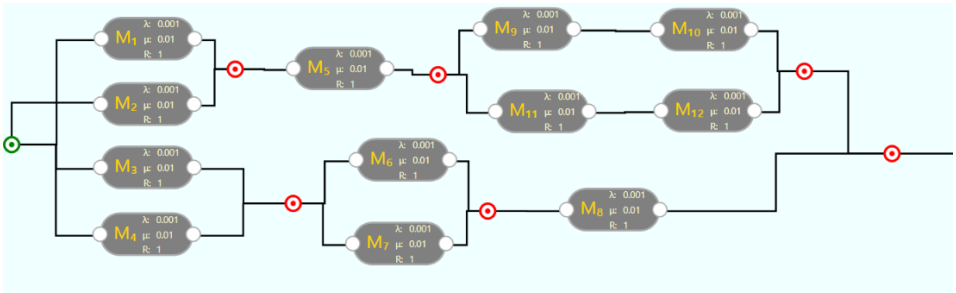


Рис. 4.16. Робоча область програмного засобу

В робочій області представлена візуалізація схеми взаємодії модулів системи для 12 елементів з різними типами з’єднань, що відповідає реальній технічній системі низького рівня складності.

Область налаштувань призначення для відстежування та керування процесом побудови візуальної схеми взаємодії модулів системи.

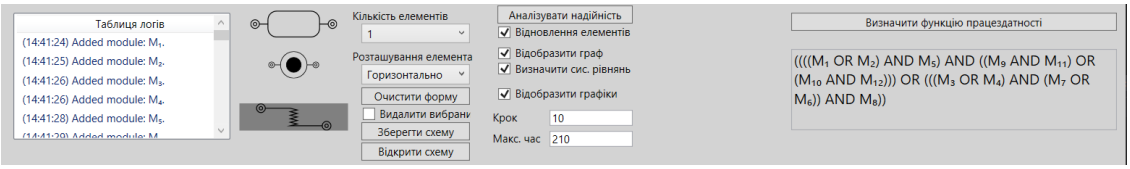


Рис. 4.17. Область налаштувань програмного засобу

Вона містить таблицю хронології створення графічних компонент схеми (рис. 4.18.). Це зручний засіб, який дозволяє відстежувати процес додавання компонент, з яких складається схема взаємодії модулів системи, а також містить час їх додавання.

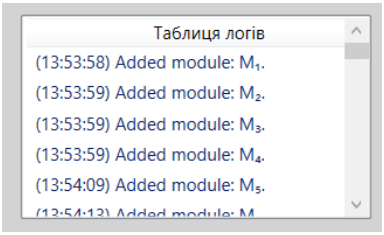


Рис. 4.18. Таблиця хронології створення графічних компонент схеми

Крім того, в області налаштувань натисканням мишею можна вибрати необхідний в даний момент часу графічний елемент, який користувач буде будувати натискаючи на певне місце робочої області (1. Модуль, 2. Вузол 3. Лінія) (рис. 4.19).

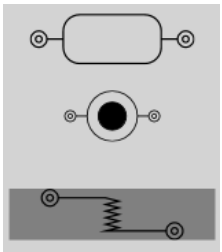


Рис. 4.19. Вибір поточного графічного елемента для побудови схеми

Також для зручності та пришвидшення побудови схеми можна вибрати кількість елементів, які потрібно одночасно додати, напрям їх побудови, очищення робочої області повністю або її окремих елементів (рис. 4.20).

- «Кількість елементів» – визначає скільки елементів будується на робочій формі за одне натискання на форму (стосується вузлів і модулів);
- «Розташування елемента» – напрям побудови елементів – вертикальний, горизонтальний;
- «Очистити форму» – повне очищення робочого вікна та всіх;
- «Видалити вибраний» – якщо вибрана дана мітка, то після натискання ПКМ наведеної на певний елемент робочого вікна даний елемент видалиться;
- «Зберегти схему» і «Відкрити схему» – дозволяє зберегти і відкрити файл із збереженою схемою взаємодії модулів (схема зберігається у JSON форматі).

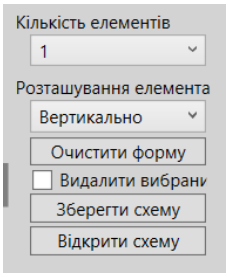


Рис. 4.20. Налаштування побудови схеми

Можливість зберігання конфігурації схеми взаємодії модулів системи є зручною функціональністю при роботі із великими системами, це дозволяє пришвидшити і покращити процес аналізу, оскільки дозволяє швидко повернутись до роботи над аналізом конкретної схеми у будь-який зручний час.

Основним компонентом даної схеми є модуль, частина технічної системи, яка відповідає за певний функціонал. За замовчуванням кожен модуль має такі значення характеристик (рис. 4.21): ймовірність відмови = 0,005; інтенсивність відновлення =0,05; кількість відновлень = 1. Користувач може їх змінити, вибравши необхідний модуль. Щоб вибрати модуль потрібно натиснути на нього правою кнопкою мишки (потрібно зважати чи не вибраний прапорець «Видалити вибраний»).

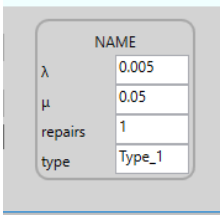


Рис. 4.21. Характеристики модуля

Після побудови схеми для певної системи можна переглянути функцію працездатності даної системи (рис. 4.22). Для цього потрібно натиснути на кнопку «Визначити функцію працездатності» в області налаштувань і у вікні з’явиться результат аналізу схеми взаємодії модулів системи.

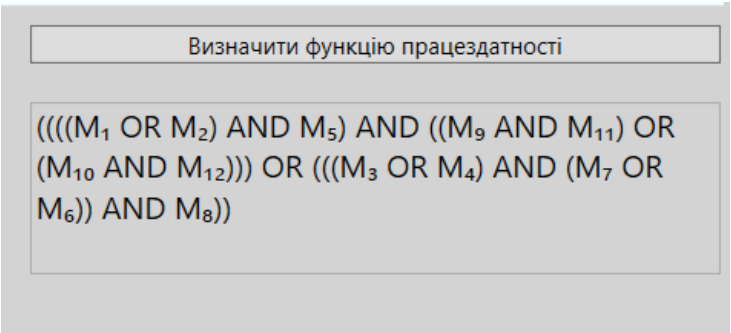


Рис. 4.22. Функція працездатності для схеми взаємодії модулів системи (рис. 4.16.)

Для подальшого аналізу на основі функції працездатності в окремому вікні налаштувань можна вибирати такі пункти (рис. 4.23):

- «Відновлення елементів» – використовувати відновлення модулів – якщо вибрано, то враховується відновлення модулів, якщо не вибрано – то ні;
- «Відобразити граф» – побудувати граф станів і переходів;
- «Визначити сис. рівнянь» – побудувати та розв’язати систему диференціальних рівнянь Колмогорова-Чепмена;
- «Відобразити графіки» – побудувати графіки залежності ймовірності перебування системи в певному стані від часу *t*.

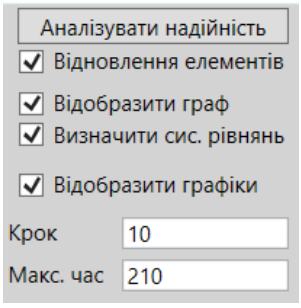


Рис. 4.23. Вікно налаштувань параметрів надійнісного аналізу

Після побудови схеми і вибору необхідних параметрів розрахунку потрібно натиснути кнопку «Аналізувати надійність». Після певного часу буде виконано розрахунки, час виконання яких буде показано у вікні історії (логування). З’явиться вікно із відкритою вкладкою «Граф станів і переходів» – дана вкладка

містить інформацію про всі стани, в яких може перебувати система, дозволяє переглянути загальний і деталізований граф станів та переходів (рис. 4.24).

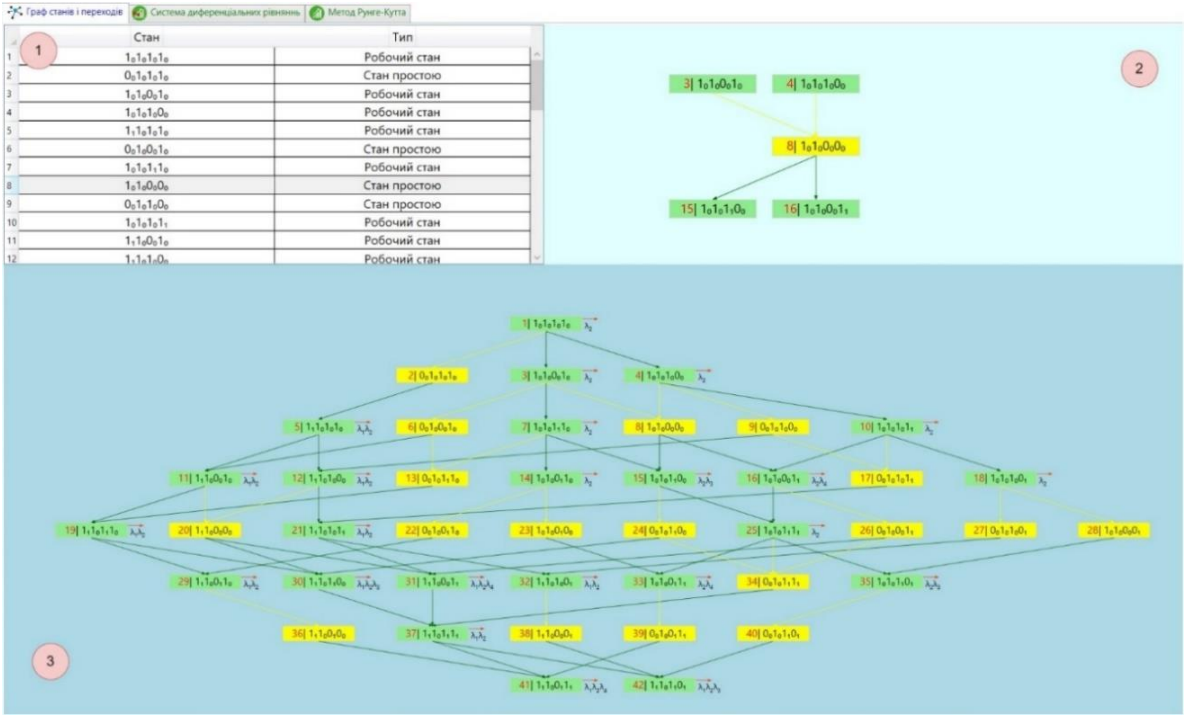


Рис. 4.24. Вікно “Граф станів і переходів”

Дане вікно поділяється на три секції:

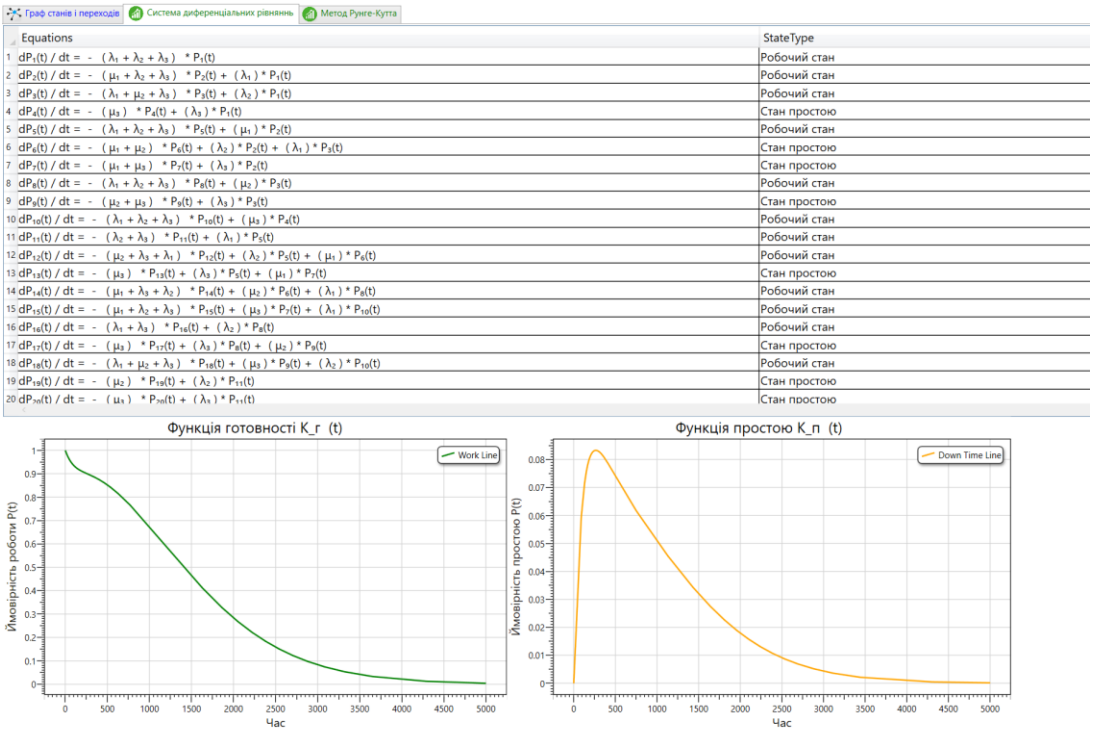
- **Секція перегляду станів (1) –** можна переглянути точну кількість станів, в яких може перебувати система і їх опис із вказаним типом;
- **Секція деталізації стану (2) –** дозволяє переглянути кожен стан в більш детальній інтерпретації з описом станів, які вгодять і виходять з нього, а також станів, які призводять до критичної відмови**;**
- **Секція перегляду графа станів і переходів (3)** – секція дозволяє побачити всі можливі стани і переходи між ними у графічному представленні, виконати збільшення і зменшення масштабу за допомогою колеса мишки.

В програмному засобі також є можливість переходу на вікно, яке дозволяє переглянути систему диференціальних рівнянь і графіків значень показників надійності досліджуваної системи (рис. 4.25).

Дане вікно складається з двох секцій:

- Секція перегляду системи рівнянь;
- Секція перегляду графіків – якщо аналіз проходить для відновлюваної системи відбувається побудова функції готовності і функції простою, а для

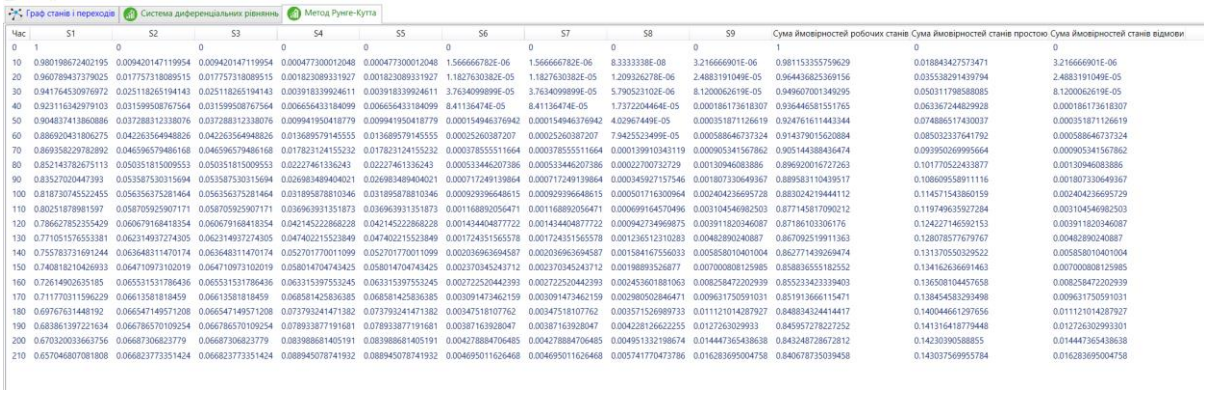
невідновлювальної системи визначається ймовірність безвідмовної роботи протягом часу *t*.



	Equations	StateType
1	dP1(t) / dt = - (λ1 + λ2 + λ3) * P1(t)	Робочий стан
2	dP2(t) / dt = - (μ1 + λ2 + λ3) * P2(t) + (λ1) * P1(t)	Робочий стан
3	dP3(t) / dt = - (λ1 + μ2 + λ3) * P3(t) + (λ2) * P1(t)	Робочий стан
4	dP4(t) / dt = - (μ3) * P4(t) + (λ3) * P1(t)	Стан простою
5	dP5(t) / dt = - (λ1 + λ2 + λ3) * P5(t) + (μ1) * P2(t)	Робочий стан
6	dP6(t) / dt = - (μ1 + μ2) * P6(t) + (λ2) * P2(t) + (λ1) * P3(t)	Стан простою
7	dP7(t) / dt = - (μ1 + μ3) * P7(t) + (λ3) * P2(t)	Стан простою
8	dP8(t) / dt = - (λ1 + λ2 + λ3) * P8(t) + (μ2) * P3(t)	Робочий стан
9	dP9(t) / dt = - (μ2 + μ3) * P9(t) + (λ3) * P3(t)	Стан простою
10	dP10(t) / dt = - (λ1 + λ2 + λ3) * P10(t) + (μ3) * P4(t)	Робочий стан
11	dP11(t) / dt = - (λ2 + λ3) * P11(t) + (λ1) * P5(t)	Робочий стан
12	dP12(t) / dt = - (μ2 + λ3 + λ1) * P12(t) + (λ2) * P5(t) + (μ1) * P6(t)	Робочий стан
13	dP13(t) / dt = - (μ3) * P13(t) + (λ3) * P5(t) + (μ1) * P7(t)	Стан простою
14	dP14(t) / dt = - (μ1 + λ3 + λ2) * P14(t) + (μ2) * P8(t) + (λ1) * P8(t)	Робочий стан
15	dP15(t) / dt = - (μ1 + λ2 + λ3) * P15(t) + (μ3) * P7(t) + (λ1) * P10(t)	Робочий стан
16	dP16(t) / dt = - (λ1 + λ3) * P16(t) + (λ2) * P8(t)	Робочий стан
17	dP17(t) / dt = - (μ3) * P17(t) + (λ3) * P8(t) + (μ2) * P9(t)	Стан простою
18	dP18(t) / dt = - (λ1 + μ2 + λ3) * P18(t) + (μ3) * P9(t) + (λ2) * P10(t)	Робочий стан
19	dP19(t) / dt = - (μ2) * P19(t) + (λ2) * P11(t)	Стан простою
20	dP20(t) / dt = - (μ1) * P20(t) + (λ1) * P11(t)	Стан простою

Рис. 4.25. Вікно перегляду системи диференціальних рівнянь

Для більш детального огляду і аналізу розв’язку системи диференціальних рівнянь можна перейти на вкладку «Метод Рунге-Кутта». Тут можна переглянути інформацію про залежність ймовірності роботи системи для станів в певний момент часу (ця таблиця, є детальним описом розв’язку системи рівнянь). Також є можливість зберегти в файл або в буфер обміну дані з таблиці. (рис. 4.26)



Час	S1	S2	S3	S4	S5	S6	S7	S8	S9	Сума ймовірностей робочих станів	Сума ймовірностей станів простою	Сума ймовірностей станів відмови
0	1	0	0	0	0	0	0	0	0	1	0	0
10	0.980198672402195	0.009420147119954	0.009420147119954	0.000477300012048	0.000477300012048	1.566666782E-06	1.566666782E-06	8.3333338E-08	3.216666901E-06	0.981153355759629	0.018843427573471	3.216666901E-06
20	0.960789437379025	0.017757318089515	0.017757318089515	0.001823089331927	0.001823089331927	1.1827630382E-05	1.1827630382E-05	1.2093262678E-06	2.4883191049E-05	0.964436825369156	0.035538291439794	2.4883191049E-05
30	0.941764530976972	0.025118265194143	0.025118265194143	0.003918339924611	0.003918339924611	3.7634099899E-05	3.7634099899E-05	5.7905231026E-06	8.1200062619E-05	0.949607001349295	0.050311798588085	8.1200062619E-05
40	0.923116342979103	0.031599508767564	0.031599508767564	0.006656433184099	0.006656433184099	8.41136474E-05	8.41136474E-05	1.7372204464E-05	0.000186173618307	0.936446581551765	0.063367244829928	0.000186173618307
50	0.904837413860886	0.037288312338076	0.037288312338076	0.009941950418779	0.009941950418779	0.000154946376942	0.000154946376942	4.02967449E-05	0.000351871126619	0.924761611443344	0.074886517430037	0.000351871126619
60	0.886920431806275	0.042263564948826	0.042263564948826	0.013689579145555	0.013689579145555	0.00025260387207	0.00025260387207	7.9425523499E-05	0.000588646737324	0.914379015620884	0.085032337641792	0.000588646737324
70	0.869358229782892	0.046596579486168	0.046596579486168	0.017823124155232	0.017823124155232	0.000378555511664	0.000378555511664	0.000139910343119	0.000905341567862	0.905144388436474	0.093950269995664	0.000905341567862
80	0.852143782675113	0.050351815009553	0.050351815009553	0.02227461336243	0.02227461336243	0.000533446207386	0.000533446207386	0.000227007327229	0.001309460883886	0.896920016727263	0.101770522433877	0.001309460883886
90	0.83527020447393	0.053587530315694	0.053587530315694	0.026983489404021	0.026983489404021	0.000717249139864	0.000717249139864	0.000345927157546	0.001807330649367	0.889583110439517	0.108609558911116	0.001807330649367
100	0.818730745522455	0.056356375281464	0.056356375281464	0.031895878810346	0.031895878810346	0.000929396648615	0.000929396648615	0.000501716300964	0.002404236695728	0.883024219444112	0.114571543860159	0.002404236695729
110	0.80251878981597	0.058705925907171	0.058705925907171	0.036963931351873	0.036963931351873	0.001168920056471	0.001168920056471	0.000699164570496	0.003104546982503	0.877145817090212	0.119749635927284	0.003104546982503
120	0.786627852355429	0.060679168418354	0.060679168418354	0.042145222868228	0.042145222868228	0.001434404877722	0.001434404877722	0.000942734969875	0.003911820346087	0.871861033306176	0.124227146592153	0.003911820346087
130	0.771051576553381	0.062314937274305	0.062314937274305	0.047402215523849	0.047402215523849	0.001724351565578	0.001724351565578	0.001235612310283	0.004828902408887	0.867092519911363	0.128078577679767	0.004828902408887
140	0.755783731691244	0.063648311470174	0.063648311470174	0.052701777001099	0.052701777001099	0.002036963694587	0.002036963694587	0.001584167556033	0.005858010401004	0.862771439269474	0.131370550329522	0.005858010401004
150	0.740818210426933	0.064710973102019	0.064710973102019	0.058014704743425	0.058014704743425	0.002370345243712	0.002370345243712	0.001988935268777	0.007000808125985	0.858836555182552	0.134162636691463	0.007000808125985
160	0.726149402635185	0.065531531786436	0.065531531786436	0.063315397553245	0.063315397553245	0.002722520442393	0.002722520442393	0.002453601881063	0.008258472202939	0.855233423339403	0.136508104457658	0.008258472202939
170	0.711770311596229	0.066135818458459	0.066135818458459	0.068581425836385	0.068581425836385	0.003091473462159	0.003091473462159	0.002980502846471	0.009631750591031	0.851913666115471	0.138454583293498	0.009631750591031
180	0.69767631448192	0.066547149571208	0.066547149571208	0.073793241471382	0.073793241471382	0.003475181077762	0.003475181077762	0.003571526989733	0.011121014287927	0.848834324414417	0.140044661297656	0.011121014287927
190	0.683861397221634	0.066786570109254	0.066786570109254	0.078933877191681	0.078933877191681	0.003871639280747	0.003871639280747	0.004228126622255	0.012726302993331	0.845957278227252	0.141316418779448	0.012726302993301
200	0.670320033663756	0.066873068023779	0.066873068023779	0.083988681405191	0.083988681405191	0.004278847064854	0.004278847064854	0.004951332198674	0.014447365438638	0.843248728672812	0.142303905888555	0.014447365438638
210	0.657046807081808	0.066823773351424	0.066823773351424	0.088945078741932	0.088945078741932	0.004695011626468	0.004695011626468	0.005741770473786	0.016283695004758	0.840678735039458	0.143037569955784	0.016283695004758

Рис. 4.26. Вкладка “ Метод Рунге-Кутта”.

Програмний комплекс також має можливість вибору мови інтерфейсу із двох можливих варіантів – англійська (рис. 4.27) і українська мова. Для налаштування мови інтерфейсу потрібно вказати в App.config файлі (файл

розміщений в кореневій папці виконавчого файлу програми), яку саме мову користувач хочемо вибрати (en-US або uk-Ua).

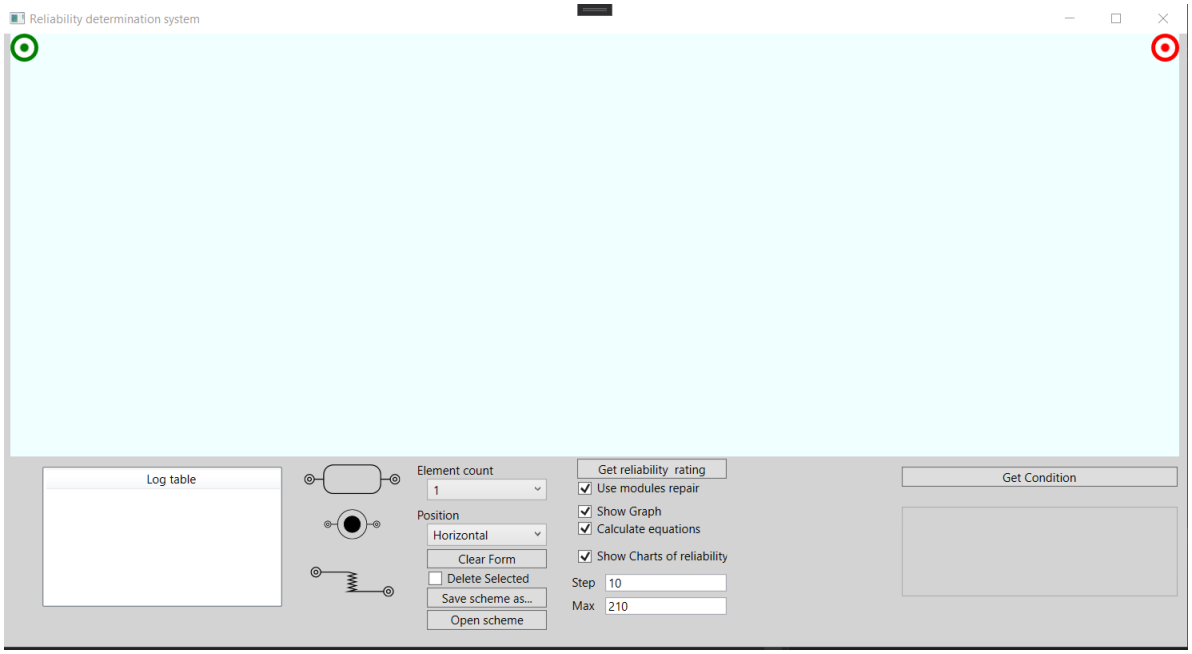


Рис. 4.27. Головне вікно програмного засобу (англійська мова)

4.4. Аналіз роботи та тестування засобу для розрахунку надійнісних характеристик складних технічних систем на основі функції працездатності

Для верифікації коректної роботи та валідації розробленого програмного засобу було проведено комплексне тестування всіх складових та модулів комплексу, а також виконано аналіз роботи розробленого методу для різного набору вхідних даних із оцінкою швидкодії роботи. Оскільки оцінка швидкодії є однією з основних характеристик ПЗ даного типу, які працюють зі схемами, що складаються із сотень або тисяч елементів.

Було визначено основні параметри, які впливають на швидкодію роботи методів і процесів реалізованих у ПЗ, та проведено їх аналіз із різною конфігурацією і комбінацією даних параметрів. До цих параметрів належать: кількість модулів в системи, кількість зв’язків між модулями в системі і також кількість відновлень для модулів.

Тестування системи відбувалося на машині із такими характеристиками:

OS Windows 10, Intel Core i7, max 2.7 ГГц, RAM 8 Гб DDR4 2133 МГц, 256 Гб SATA SSD, NVIDIA GeForce 940MX 2 Gb.

Аналіз залежності швидкодії роботи методу від кількості модулів в системі для послідовних, паралельних і змішаних з’єднань було проведено на трьох різних конфігураціях системи :

- **послідовне з’єднання** – всі елементи системи розташовані послідовно один за одним;
- **паралельне з’єднання** – всі елементи системи розташовані паралельно один до одного;
- **змішане з’єднання** – елементи системи розташовані в різному поєднанні послідовних і паралельних з’єднань (рис 4.28).

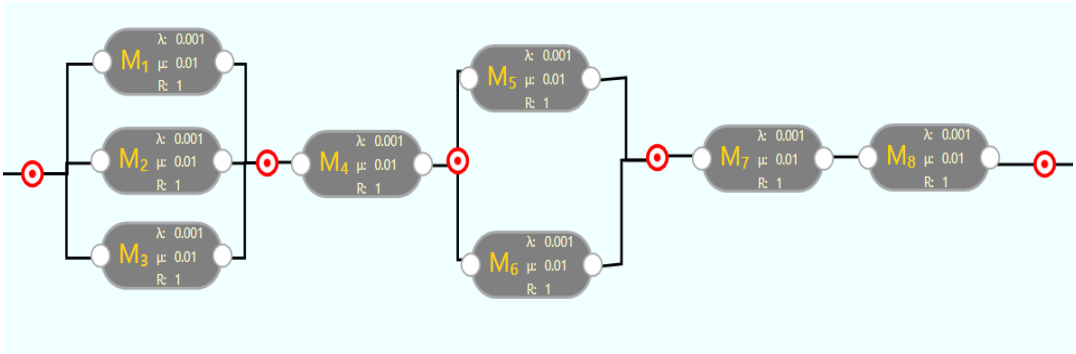


Рис 4.28. Приклад схеми взаємодії модулів технічної системи із змішаним з’єднанням її елементів

Для програмної реалізації методу автоматизованого визначення функції працездатності технічних систем було виконано аналіз із різною кількістю модулів в системі та різною конфігурацією з’єднань між модулями (паралельні та послідовні). Результати аналізу швидкодії представлені в таблиці 4.3 і рисунку 4.29. Кількість відновлень елементів не є визначальним фактором впливу на швидкодію роботи, оскільки на даному кроці це не враховується.

Таблиця 4.3 – Аналіз швидкодії роботи програмної реалізації методу автоматизованого визначення функції працездатності технічних систем

№	Кількість модулів	Час (послідовне з’єднання), с	Час (змішане з’єднання), с	Час (паралельне з’єднання), с
1	1	0. 0000308	0. 0000321	0. 0000330
2	2	0. 0000474	0. 0000484	0.0000488

3	5	0. 0000791	0. 0000821	0.0000922
4	10	0.0001703	0.0002093	0.0002418
5	20	0.0002004	0.0004931	0.0007001
6	50	0.0003784	0.0058938	0.0037932
7	100	0.0004165	0.0486683	0.0571824
8	150	0.0004484	0.1509184	0.2522243
9	200	0.0005171	0.3154700	0.4550050
10	250	0.0060551	0.6377826	1.1881978
11	500	0.0106783	2.1342265	4.4533016
12	750	0.9105432	5.2297644	11.6021298
13	1000	1.4004321	10.5671256	18.8623663

Рис. 4.29. Графік залежності швидкодії роботи методу від кількості модулів в системі для послідовних, паралельних і змішаних з'єднань

Як видно з отриманого графіка залежності швидкодії роботи методу від кількості модулів в системі для послідовних, паралельних і змішаних з'єднань розроблений програмний комплекс дозволяє за невеликий час (близько 10 с.) визначити функцію працездатності для системи з 1000 елементів (модулів) зі змішаним типом з'єднання, що відповідає складній технічній системі та найпоширенішому типу з'єднання елементів системи (змішаному).

Аналіз роботи програмної реалізації процесу побудови графа станів і переходів було виконано для схем різних конфігурацій, але при даному аналізі необхідно враховувати кількість відновлень модулів системи, оскільки даний фактор вагомо впливає на кількість станів і переходів між ними.

В таблиці 4.4 наведено вплив кількості відновлень для системи із 5-ти модулів із послідовним і паралельним з'єднанням на кількість станів в графі (враховуються робочі стани, стани простою і стан критичної відмови).

Таблиця 4.4 – Аналіз впливу кількості відновлень кожного елементу для системи із 5-ти модулів на кількість станів в графі

№	Кількість відновлень	Кількість станів при послідовному з'єднанні	Кількість станів при паралельному з'єднанні
1	0	2	32
2	1	244	1 024
3	2	3 126	7 776
4	3	16 808	32 768
5	4	59 050	100 000
6	5	161 052	248 832
7	6	371 294	537 824
8	7	759 376	1 048 576
9	8	1 419 858	1 889 568
10	9	2 476 100	3 200 000
1	10	4 084 102	5 153 632

Як видно з рисунка 4.30 кількість можливих станів системи зростає відповідно кількість відновлень і тип з'єднання модулів в системі суттєво впливає на кількість станів, в яких може перебувати система.

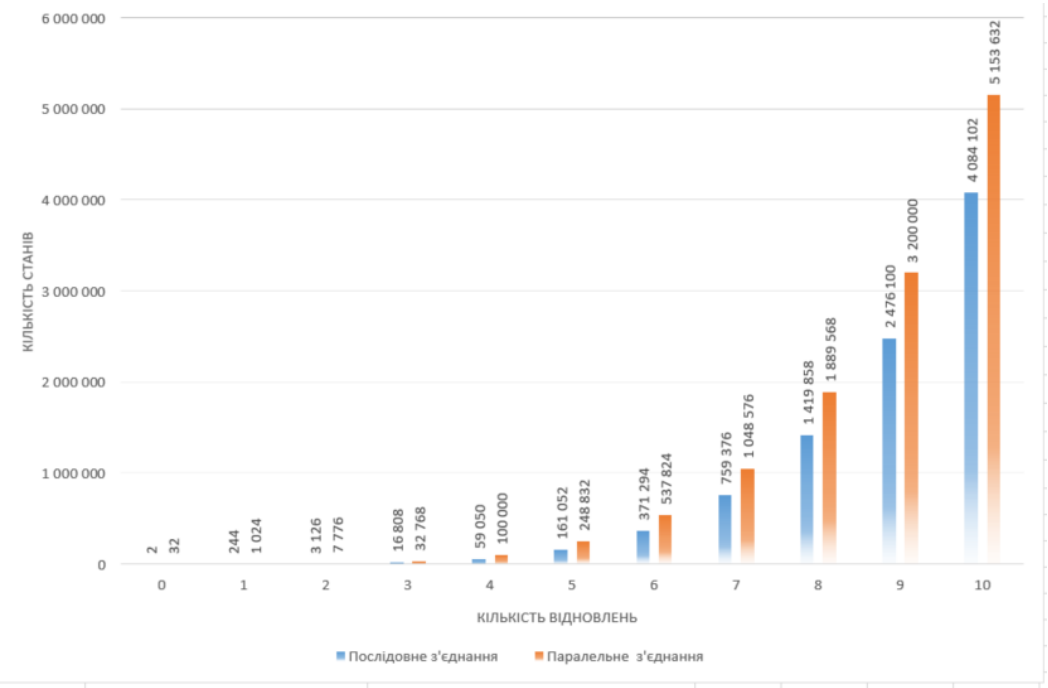


Рис. 4.30. Гістограма залежності кількості станів в графі від кількості відновлень при паралельному і послідовному з’єднанні елементів системи

Аналіз швидкодії програмної реалізації процесу побудови графа станів і переходів було виконано для системи із паралельним з’єднанням модулів без відновлення (таблиця 4.5).

Таблиця 4.5 – Аналіз швидкодії програмної реалізації процесу побудови графа станів і переходів

№	Кількість модулів	Кількість станів	Час роботи алгоритму, с
1	1	2	0.0451121
2	2	4	0.0119204
3	5	32	0.3929208
4	10	1 024	3.3207597
5	15	32 768	8.6198466
6	20	1 048 576	58.1923007
7	25	33 554 432	> 6 хв

З отриманих даних можна зробити висновок, що візуалізація графу станів та переходів для технічних систем з великою кількістю модулів є нетривіальною

задачею (залежно від конфігурацій системи, для прикладу кількість можливих станів для системи з 30-и паралельно розташованих елементів є ≈ 1.07* $10^{9)}$. Час роботи програмного комплексу теж зростає експоненційно, відповідно, існує потреба в декомпозиції таких технічних систем на менші підсистеми.

Робота процесу опрацювання системи диференціальних рівнянь Колмогорова-Чепмена складається із двох частин:

- визначення системи рівнянь – даний процес складається із визначення рівнянь на основі формули (1.2) – для кожного стану системи.
- розв’язок системи рівнянь методом Рунге-Кутта – використання програмної реалізації методу Рунге-Кутта для розв’язку системи рівнянь.

Аналіз швидкодії роботи програмної реалізації процесу опрацювання системи диференціальних рівнянь Колмогорова-Чепмена наведено в таблиці 4.6. на проміжку від 1 000 до 1 000 000 станів системи. Як неважко помітити, розроблений програмний комплекс за досить короткий час (≈ 35с.) розв’язує систему диференціальних рівнянь для $1*10^9$ рівнянь в системі.

Таблиця 4.6 – Аналіз швидкодії процесу опрацювання рівнянь

№	Кількість станів	Час опрацювання (визначення і розв’язок), с
1	1 000	0.023
2	10 000	0.586
3	50 000	1.127
4	100 000	2.012
5	250 000	4.827
6	500 000	10.142
7	750 000	18.721
8	1 000 000	35.299

На рисунку 4.31 зображено схему взаємодії елементів системи для невідновлювальної системи із 22 елементів і наведено час виконання аналізу для кожного етапу.

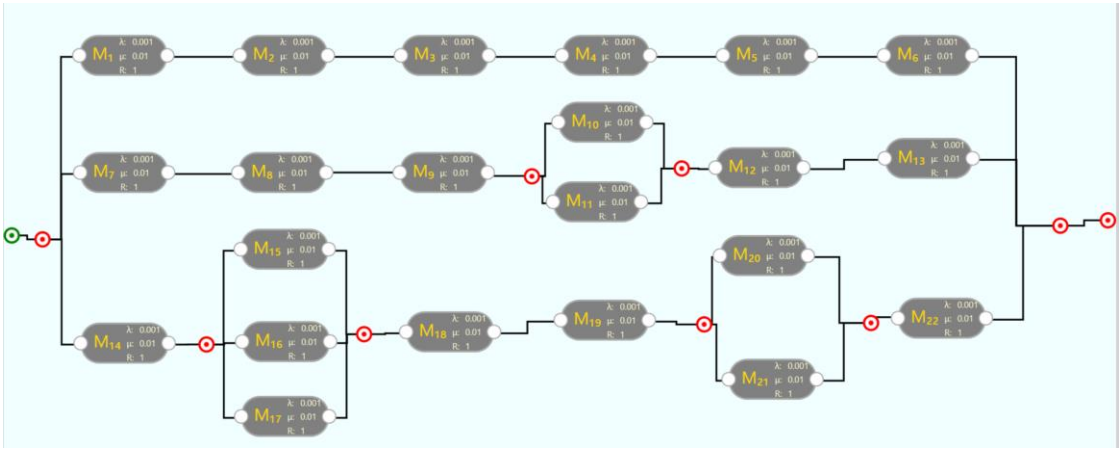


Рис. 4.31. Схема для невідновлювальної системи із 22 елементів

Час визначення функції працездатності - 0.0004122 с.

Час виконання процесу побудови графа станів і переходів (визначено станів 52 694) – 35.1682 с.

Час опрацювання системи диференціальних рівнянь Колмогорова-Чепмена – 1.5435 с.

Загальний час виконання конкретного завдання для схеми взаємодії елементів технічної системи з 22 елементів становить – 36.7116. Основна частина часу припадає на програмну реалізацію процесу побудови графа станів і переходів та, можливо в подальшому, повинна бути вдосконалена для покращення швидкодії роботи програми загалом.

4.5. Висновки до розділу 4

В даному розділі було описано програмне забезпечення для автоматизації процесу роботи методу подання Марковського процесу вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами і методу визначення функції працездатності для Марковських моделей надійності.

Програмне забезпечення для автоматизації подання Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами реалізоване за допомогою бібліотеки ReactJS (для візуалізації графа було використано бібліотеку react-d3-graph) і мови програмування JavaScript у вигляді Web застосунку. Даний програмний продукт

дозволяє проєктанту поетапно виконати моделювання процесу вищого порядку за допомогою процесу першого порядку: внесення інформації про вхідний граф першого порядку з матрицею суміжності і на основі вхідних даних виконати представлення Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами.

Для програмного забезпечення подання Марковського процесу вищого порядку було проведено тестування на швидкодію при різних випадках конфігурацій вхідних даних: при збільшенні порядку процесу і при різній кількості вузлів і ребер у вхідному графі першого порядку із поступовим збільшенням їх кількості для моделювання процесів 2-го, 3-го і 4-го порядку. Аналіз результатів тестування продемонстрував, що розроблене ПЗ дає достатньо високий рівень швидкодії при моделюванні процесів вищого порядку.

Також у даному розділі описано процес визначення показників надійності із використанням функції працездатності та графа станів і переходів. Даний процес складається із таких етапів: розроблення моделі об’єкта дослідження у вигляді графа станів і переходів із використанням функції працездатності**;** формування та розв’язання системи диференціальних рівнянь Колмогорова-Чепмена; визначення показників надійності на основі розподілу ймовірностей перебування в станах. Даний процес був покроково реалізований у програмному засобі для розрахунку надійнісних характеристик складних технічних систем на основі функції працездатності.

Програмний засіб для розрахунку надійнісних характеристик складних технічних систем на основі функції працездатності було розроблено із використанням мови програмування C# і технології WPF для платформи Windows. Він має гнучку модульну структуру та може легко модифікуватись і розширюватись. Кожен модуль відповідає за певне конкретне завдання: визначення функції працездатності; побудова графа станів і переходів; визначення і розв’язування системи диференціальних рівнянь; візуалізація графічних компонент тощо.

При роботі із програмним засобом користувач може за допомогою графічних компонент візуалізувати схему взаємодії модулів технічної системи на основі якої за допомогою розробленого методу побудови функції працездатності автоматизовано визначається функція працездатності технічної системи, яка дозволяє надалі побудувати граф станів і переходів та визначити певні показники надійності для системи: для відновлюваної системи функцію готовності і функцію простою, а для невідновлювальної системи визначається ймовірність безвідмовної роботи протягом часу *t*.

Даний програмний засіб також протестовано на швидкодію, тестування відбувалось для окремих модулів системи. Як результат, можна підкреслити, що програмна реалізація методу визначення функції працездатності дозволяє за невеликий час (близько 10 с.) визначити функцію працездатності для системи з 1000 елементів (модулів) зі змішаним типом з’єднання.

Розроблене програмне забезпечення дозволяє автоматизувати процес надійнісного аналізу на основі розроблених методів, що дозволяє зменшити ймовірність внесення помилок при визначенні функції працездатності або представлення графа вищого порядку, і зменшити вплив людського фактора і відповідно підвищити достовірність при оцінці надійності із використанням даних методів і моделей.

ОСНОВНІ РЕЗУЛЬТАТИ ТА ВИСНОВКИ

У дисертації розв’язано актуальну наукову задачу підвищення точності прогнозування та оцінювання показників надійності програмно-апаратних систем шляхом удосконалення відповідних моделей надійності та розроблення методів і засобів автоматизації їх побудови.

Основні наукові та практичні результати дисертаційної роботи:

1) Проведено огляд та аналіз літературних джерел відповідно до теми дисертаційного дослідження. Виконано аналіз і опис основних тверджень і критерій із теорії надійності. Проведено аналіз і порівняння найпоширеніших моделей надійності і моделей прогнозування дефектності ПЗ.

2) Розроблено метод автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами. Використання розробленого методу дозволяє автоматизувати процес аналізу надійності ПЗ на етапах тестування і супроводу за допомогою моделі надійності у вигляді ланцюга Маркова вищого порядку, що дає змогу підвищити достовірність оцінювання показників надійності. Зокрема, верифікація цього методу на прикладі розрахунку функції готовності ПЗ для польотів наносупутників CubeSat дала можливість підвищити достовірність оцінки до 50% порівняно з моделлю надійності першого порядку, та до 10% у випадку використання моделей другого та третього порядку.

3) Розроблено метод автоматизованого визначення функції працездатності для Марковських моделей надійності, який дозволяє автоматизувати процес побудови такої функції, зменшивши вплив людського фактора і ймовірність внесення помилок, що дає змогу підвищити точність прогнозу таких моделей.

4) На основі результатів тестування і випробувань методу визначення функції працездатності та побудови на її основі графа станів і переходів отримано співвідношення, які дозволяють визначити максимальну та мінімальну кількість станів системи, а також окремі вирази для визначення максимальної і мінімальної

кількості працездатних станів і станів простою із загальної множини для системи із n елементів і r відновлень.

5) З використанням таких методів вибору ознак, як Boruta, Step-wise selection, Exhaustive Feature Selection, Random Forest Importance, LightGBM Importance, Genetic Algorithms, Principal Component Analysis, Xverse python, визначено обмежену кількість метрик коду ПЗ, які найбільше впливають на його дефектність. Визначена множина включає наступні метрики: (кількість рядків коду за МакКейбом (loc), загальна кількість операндів і операторів (N), функціональність модуля (I), обсяг на мінімальному виконанні (V), зусилля на написання модуля (E), цикломатична складність за МакКейбом (v(g)), кількість гілок в репозиторії, в яких змінювався модуль (branchCount)).

6) Використання цієї обмеженої кількості метрик покращило точність класифікації модулів на дефектні і без дефектів на 10-21% (з 61.7-72.5% до 80.0-85.5%) порівняно з використанням усього набору метрик для таких класифікаторів: Random Forest, Support Vector Machine, k-nearest neighbor, Decision Tree classifier, AdaBoost classifier, Gradient Boosting.

7) З використанням визначеної множини метрик коду та статистичного регресійного методу побудовано модель дефектності ПЗ. Точність прогнозу з використанням цієї моделі становить 82,9%.

8) Розроблено метод класифікації модулів ПЗ за дефектністю на основі метрик коду з використанням стекового ансамблю нейронної мережі на основі радіально-базисних функцій, рекурентної нейронної мережі та мережі довгої короткочасної пам’яті. Метод дає можливість підвищити точність класифікації програмних модулів на дефектні і без дефектів з 79-86% до 92%. Даний підхід дозволяє визначати потенційно небезпечні програмні модулі з погляду дефектності та вжити заходів для усунення цих проблем в модулі на етапі розробки і тестування.

9) Розроблено програмне забезпечення, яке дає можливість автоматизувати використання розроблених моделей і методів на етапах проєктування, кодування

і тестування життєвого циклу ПЗ і таким чином підвищити точність оцінки показників надійності ПЗ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Musa, John D. 1999. Software Reliability Engineering : More Reliable Software, Faster Development and Testing. New York: Mcgraw-Hill.
2. ISO/IEC 25030:2007, Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — System and software quality model
3. Hoang Pham. 2000. Software Reliability. Singapore ; New York: Springer..
4. Peled, Doron A. 2011. Software Reliability Methods. New York; London: Springer.
5. Birman, Kenneth P. 1996. Building Secure and Reliable Network Applications. Greenwich: Manning ; Prentice Hall.
6. Cervantes, Humberto, and Rick Kazman. 2016. Designing Software Architectures : A Practical Approach. Boston: Addison-Wesley.
7. Lyu MR. Handbook of software reliability engineering. NY: McGraw-Hill/IEEE Computer Society Press; 1996.
8. “Software Reliability Models: A Review.” 1984. Microelectronics Reliability 24 (5): 991. https://doi.org/10.1016/0026-2714(84)90073-8.
9. Chaurasia PK. Classification of Software Reliability Models. IJARCSSE. 2014 Aug; 4(8):1084–91.
10. William F. Software reliability modeling survey – Naval Surface Warfare Center, 1996.
11. Gayathry, G., and R. Thirumalai Selvi. 2015. “Classification of Software Reliability Models to Improve the Reliability of Software.” Indian Journal of Science and Technology 8 (29). https://doi.org/10.17485/ijst/2015/v8i29/85287.
12. “Software Reliability Analysis Models.” 1985. Microelectronics Reliability 25 (3): 583. https://doi.org/10.1016/0026-2714(85)90228-8.
13. Goel A.L. Software reliability models: assumptions, limitations, and applicability / A.L. Goel // IEEE Transactions on software engineering. – 1985. – Vol. SE-11, No 12. – P. 1411–1423.
14. Bharathi R, and Selvarani R. 2019. “Software Reliability Assessment of Safety Critical System Using Computational Intelligence.” International Journal of

Software Science and Computational Intelligence 11 (3): 1–25. https://doi.org/10.4018/ijssci.2019070101.

15. Pham H. System software reliability // Springer-Verlag London Limited, 2006. – 440 p.
16. Pham, Hoang. 2003. “Software Reliability and Cost Models: Perspectives, Comparison, and Practice.” European Journal of Operational Research 149 (3): 475–89. https://doi.org/10.1016/s0377-2217(02)00498-8.
17. ZHOU, Na-qin. 2008. “Reliability Model of Component-Based Software.” Journal of Computer Applications 28 (6): 1630–31. https://doi.org/10.3724/sp.j.1087.2008.01630.
18. Al Turk, Lutfiah Ismail, and Eftekhar Gabel Alsolami. 2016. “Jelinski-Moranda Software Reliablity Growth Model : A Brief Literature and Modification.” International Journal of Software Engineering & Applications 7 (2): 33–44. https://doi.org/10.5121/ijsea.2016.7204.
19. Moranda P. B. Prediction of software reliability during debugging / P. B. Moranda // Proc. of Annu. reliability and maintainability symp. – 1975. – P. 327–332.
20. “Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures.” 1980. Microelectronics Reliability 20 (4): 541. https://doi.org/10.1016/0026-2714(80)90640-x.
21. Musa, J. D. 1975. “A Theory of Software Reliability and Its Application.” IEEE Transactions on Software Engineering SE-1 (3): 312–27. https://doi.org/10.1109/tse.1975.6312856.
22. Musa J.D. A Logarithmic Poisson Execution Time Model for Software Reliability Measurement / J.D. Musa, K. Okumoto // Proceedings of the 7th International Conference on Software Engineering. – 1984. – P. 230–238.
23. Littlewood, B. 1979. “The Littlewood-Verrall Model for Software Reliability Compared with Some Rivals.” Journal of Systems and Software 1 (January): 251–58. https://doi.org/10.1016/0164-1212(79)90025-6.

24. Prasad, Gopal, and Ram Nivas. 2015. “Software Reliability Estimation of Component Based Software System Using Fuzzy Logic.” International Journal of Computer Applications 127 (7): 16–20. https://doi.org/10.5120/ijca2015904871.
25. Goševa-Popstojanova, Katerina, and Kishor S Trivedi. 2001. “Architecture-Based Approach to Reliability Assessment of Software Systems.” Performance Evaluation 45 (2-3): 179–204. https://doi.org/10.1016/s0166-5316(01)00034-7.
26. W. Wang, Y. Wu, M. Chen, An architecture-based software reliability model, in: Proceedings of the Pacific Rim International Symposium on Dependable Computing, 1999, pp. 143–150
27. M. Xie, C. Wohlin, An additive reliability model for the analysis of modular software failure data, in: Proceedings of the Sixth International Symposium on Software Reliability Engineering (ISSRE’95), 1995, pp. 188–194
28. C. Wang, K.S. Trivedi, Integration of specification form modeling and specification for system design, in: Proceedings of the 14th International Conference App
29. M. Shooman, Structural models for software reliability prediction, in: Proceedings of the Second International Conference on Software Engineering, 1976, pp. 268–280.
30. S. Krishnamurthy, A.P. Mathur, On the estimation of reliability of a software system using reliabilities of its components, in: Proceedings of the Eighth International Symposium on Software Reliability Engineering (ISSRE’97), 1997, pp. 146–155.
31. S. Yacoub, B. Cukic, H. Ammar, Scenario-based reliability analysis of component-based software, in: Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE’99), 1999, pp. 22–31.
32. Malaiya, Y.K., Srimani, P.K.: Software Reliability Models: Theoretical Developments, Evaluation and Applications. IEEE Computer Society Press, Los Angeles (1990).
33. Gokhale, S.S., Wong, W.E., Horgan, J.R., Trivedi, K.S.: An Analytical Approach to Architecture-Based Software Performance and Reliability Prediction. Performance Evaluation 58(4), 391–412 (2004).
34. Krishnamurthy, S., Mathur, A.P.: On the estimation of reliability of a software system using reliabilities of its components. In: Proceedings of the 8th International

Symposium on Software Reliability Engineering, pp. 146–155. IEEE, Albuquerque, NM, USA (1997).

35. Goševa -Popstojanova, K., Hamill, M.: Architecture-Based Software Reliability: Why Only a Few Parameters Matter? In: 31st Annual International Computer Software and Applications Conference, pp. 423–430. IEEE, Beijing, China (2007).
36. Gokhale, Swapna S., W. Eric Wong, J.R. Horgan, and Kishor S. Trivedi. 2004. "An Analytical Approach to Architecture-Based Software Performance and Reliability Prediction." Performance Evaluation 58 (4): 391–412. https://doi.org/10.1016/j.peva.2004.04.003.
37. Laprie, Jean-Claude. 1984. "Dependability Evaluation of Software Systems in Operation." IEEE Transactions on Software Engineering SE-10 (6): 701–14. https://doi.org/10.1109/tse.1984.5010299.
38. "Software Reliability Model for Modular Program Structure." 1980. Microelectronics Reliability 20 (4): 542. https://doi.org/10.1016/0026-2714(80)90649-6.
39. Kołowrocki, Krzysztof. 2003. "Semi-Markov Processes and Reliability." Reliability Engineering & System Safety 79 (1): 117. https://doi.org/10.1016/s0951-8320(02)00112-6.
40. von Bochmann, G., Jourdan, G.-V., Wan, B.: Improved Usage Model for Web Application Reliability Testing. In: Wolff, B., Zaidi, F. (eds.) ICTSS 2011, LNCS, vol. 7019, pp. 15–31. Springer, Heidelberg (2011).
41. Takagi, T., Furukawa, Z., Yamasaki, T.: Accurate Usage Model Construction Using High-Order Markov Chains. In: Supplementary Proceedings of 17th International Symposium on Software Reliability Engineering, pp. 1–2. IEEE, Raleigh, NC, USA (2006).
42. Yakovyna, V., Nytrebych, O.: Discrete and Continuous Time High-Order Markov Models for Software Reliability Assessment. In: Proceedings of the 11th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer, pp. 419–431. CEUR-WS.org, Vol. 1356 (2015) online.

43. Yakovyna, V., Nytrebych, O., Fedasyuk, D.: The representation of high order Markov process through equivalent first order process. In: Proceedings of 6th International Conference of Young Scientists Computer Science and Engineering, pp. 216–217. Lviv Polytechnic Publishing, Lviv, Ukraine (2013).
44. Berchtold, A., Raftery, A.E.: The mixture transition distribution model for high-order Markov chains and non-Gaussian time series. Statistical Science 17(3), 328–356 (2002).
45. Shamshad, A., Bawadi, M.A., Wan Hussin, W.M.A., Majid, T.A.: First and second order Markov chain models for synthetic generation of wind speed time series. S.A.M. Sanusi Energy 30(5), 693–708 (2005).
46. X. Chen, Q. Gu, W. S. Liu, S. L. Liu, & C. Ni. "Survey of static software defect prediction". *Journal of Software (in Chinese).* 2016; Vol. 27 No. 1: 1–25.
47. "Software Defect Prediction Analysis by Using Machine Learning Algorithms." 2019. International Journal of Recent Technology and Engineering 8 (2S11): 3544–46. https://doi.org/10.35940/ijrte.b1438.0982s1119.
48. Li, Libo, Stefan Lessmann, and Bart Baesens. 2019. "Evaluating Software Defect Prediction Performance: An Updated Benchmarking Study." SSRN Electronic Journal. https://doi.org/10.2139/ssrn.3312070.
49. Khan,et al., Ali Athar. 2016. "Comparison of Software Complexity Metrics." International Journal of Computing and Network Technology 4 (1): 19–26. https://doi.org/10.12785/ijcnt/040103.
50. Software Defect Prediction Models for Quality Improvement: A Literature Study September 2012 International Journal of Computer Science Issues 9(5):288-296
51. Hussain*, Mandi Akif, Revoori Veeharika Reddy, Kedharnath Nagella, and Vidya S. 2021. "Software Defect Estimation Using Machine Learning Algorithms." International Journal of Recent Technology and Engineering 10 (1): 204–8. https://doi.org/10.35940/ijrte.a5898.0510121.
52. Sunghun Kim, Hongyu Zhang, Rongxin Wu and Liang Gong, "Dealing with Noise in Defect Prediction", CSE"11, Waikiki, Honolulu, HI, USA, ACM 978-1-4503-0445-0/11/05, 2011.

53. Balogun, Abdullateef O., Shuib Basri, Saipunidzam Mahamad, Said J. Abdulkadir& Amos O. Bajeh. “Impact of Feature Selection Methods on the Predictive Performance of Software Defect Prediction Models: An Extensive Empirical Study”. *Symmetry*. 2020; Vol. 12 No. 7: 1147-1156. DOI: https://doi.org/10.3390/sym12071147.
54. LEI Tianwei, XUE Jingfeng, WANG Yong, NIU Zequn, SHI Zhiwei & ZHANG Yu. “WCM-WTrA: A Cross-Project Defect Prediction Method Based on Feature Selection and Distance-Weight Transfer Learning”. *Chinese Journal of Electronics*. 2021. p. 133-140. DOI: doi: 10.1049/cje.2021.00.119.
55. Jiarpakdee, Jirayus, Chakkrit Tantithamthavorn, & Christoph Treude. “The Impact of Automated Feature Selection Techniques on the Interpretation of Defect Models”. *Empirical Software Engineering.* 2020; Vol. 25 No. 5: 3590–3638. DOI: https://doi.org/10.1007/s10664-020-09848-1.
56. Belouch, Mustapha, Salah Elhadaj & Mohamed Idhammad. “A Hybrid Filter-Wrapper Feature Selection Method for DDoS Detection in Cloud Computing”. I*ntelligent Data Analysis.* 2018: Vol. 22 No. 6: 1209–1226. DOI: https://doi.org/10.3233/ida-173624.
57. Anbu, M., & G. S. Anandha Mala. “Feature Selection Using Firefly Algorithm in Software Defect Prediction”. *Cluster Computing.* 2017; Vol. 22 No. S5: 10925–34. DOI: https://doi.org/10.1007/s10586-017-1235-3.
58. Hans, Rahul, & Harjot Kaur. “Opposition-Based Enhanced Grey Wolf Optimization Algorithm for Feature Selection in Breast Density Classification”. *International Journal of Machine Learning and Computing.* 2020: Vol. 10 No. 3: 458–64. DOI: https://doi.org/10.18178/ijmlc.2020.10.3.957.
59. P. Sridhar & Mehta S. “Stacking Based Ensemble Learning for Improved Software Defect Prediction”. Proceeding of Fifth International Conference on Microelectronics, Computing and Communication Systems. 2021. p. 167–178.
60. WEI, H., SHAN, C., HU, C., ZHANG, Y., & YU, X. “Software Defect Prediction via Deep Belief Network”. *Chinese Journal of Electronics*. 2019; Vol. 28 No. 5: 925–932. DOI: https://doi.org/10.1049/ cje.2019.06.012

61. Hu, Changhong, Heqi Wang, Xiangzhi Li, Hailong Liu, Ming Sun, & Wu Sun. “Decision-Level Defect Prediction Based on Double Focuses”. *Chinese Journal of Electronics.* 2017; Vol. 26 No. 2: 256–62. DOI: https://doi.org/10.1049/cje.2017.01.005.
62. Takuya Asano & Masateru Tsunoda. “Using Bandit Algorithms for Project Selection in Cross-Project Defect Prediction”. 37th International Conference on Software Maintenance and Evolution (ICSME). 2021. p. 626-643
63. Tang, Shiqi, Song Huang, Changyou Zheng, Erhu Liu, Cheng Zong, and Yixian Ding. “A Novel Cross-Project Software Defect Prediction Algorithm Based on Transfer Learning”. *Tsinghua Science and Technology.* 2020; Vol. 27 No.1: 41–57. DOI: https://doi.org/10.26599/tst.2020.9010040.
64. Lov Kumar, Mukesh Kumar & Lalita Bhanu Murthy. “An Empirical Study on Application of Word Embedding Techniques for Prediction of Software Defect Severity Level”. Proceedings of 16th Conference on Computer Science and Intelligence Systems (FedCSIS). 2021. p. 477-484. DOI: https://doi.org/ 10.15439/2021F100
65. Qiao, Lei, Xuesong Li, Qasim Umer, and Ping Guo. 2020. “Deep Learning Based Software Defect Prediction.” Neurocomputing 385 (April): 100–110. https://doi.org/10.1016/j.neucom.2019.11.067.
66. Albahli, Saleh. 2019. “A Deep Ensemble Learning Method for Effort-Aware Just-In-Time Defect Prediction.” Future Internet 11 (12): 246. https://doi.org/10.3390/fi11120246.
67. David Lo, Xin Xia, and Jianling Sun. 2017. “TLEL: A Two-Layer Ensemble Learning Approach for Just-In-Time Defect Prediction.” Information and Software Technology 87 (July): 206–20. https://doi.org/10.1016/j.infsof.2017.03.007.
68. Wang, Song, Taiyue Liu, Jaechang Nam, and Lin Tan. 2018. “Deep Semantic Feature Learning for Software Defect Prediction.” IEEE Transactions on Software Engineering, 1–1. https://doi.org/10.1109/tse.2018.2877612.

69. Manjula, C., and Lilly Florence. 2018. “Deep Neural Network Based Hybrid Approach for Software Defect Prediction Using Software Metrics.” Cluster Computing, January. https://doi.org/10.1007/s10586-018-1696-z.
70. Jingfei Chang, and Zhen Wei. 2020. “PathPair2Vec: An AST Path Pair-Based Code Representation Method for Defect Prediction.” Journal of Computer Languages 59 (August): 100979. https://doi.org/10.1016/j.cola.2020.100979.
71. Albahli, Saleh. 2019. “A Deep Ensemble Learning Method for Effort-Aware Just-In-Time Defect Prediction.” Future Internet 11 (12): 246. https://doi.org/10.3390/fi11120246.
72. WANG, Pei, Cong JIN, and He-he GE. 2013. “Mutual Information-Based Feature Selection Approach for Software Defect Prediction.” Journal of Computer Applications 32 (6): 1738–40. https://doi.org/10.3724/sp.j.1087.2012.01738.
73. Bin Liu, and Shihai Wang. 2018. “Software Defect Prediction Using Stacked Autoencoders.” Information and Software Technology 96 (April): 94–111. https://doi.org/10.1016/j.infsof.2017.11.008.
74. Zhao, Linchang, Zhaowei Shang, Ling Zhao, Taiping Zhang, and Yuan Yan Tang. 2019. “Software Defect Prediction via Cost-Sensitive Siamese Parallel Fully-Connected Neural Networks.” Neurocomputing 352 (August): 64–74. https://doi.org/10.1016/j.neucom.2019.03.076.
75. Gonzalez, C.E., Rojas, C.J., Bergel, A., Diaz, M.A. (2019) “An Architecture-Tracking Approach to Evaluate a Modular and Extensible Flight Software for CubeSat Nanosatellites.” *IEEE Access* **7**: 126409–126429.
76. Seniv Maksym, Mykuliak Andrii, Senechko Andrij Recursive algorithm of traversing reliability block diagram for creation reliability and refuse logical expressions // Perspective technologies and methods in MEMS design (MEMSTECH’2016) : pros. of XII-th Intern. Conf., 20-24 april 2016, Lviv - Polyana, Ukraine: ІПП "Вежа і Ко". – P. 199-201.
77. “Getting Started — Scikit-Learn 0.23.2 Documentation.” n.d. Scikit-Learn.org. https://scikit-learn.org/stable/getting_started.html.

78. Team, Keras. n.d. “Keras Documentation: Getting Started.” Keras.io. https://keras.io/getting_started/.
79. “PROMISE DATASETS PAGE.” n.d. Promise.site.uottawa.ca. Accessed January 19, 2022. http://promise.site.uottawa.ca/SERepository/datasets-page.html.
80. Bekkar, Mohamed, and Taklit Akrouf Alitouche. 2013. “Imbalanced Data Learning Approaches Review.” International Journal of Data Mining & Knowledge Management Process 3 (4): 15–33. https://doi.org/10.5121/ijdkp.2013.3402.
81. Chandrashekar, Girish, and Ferat Sahin. 2014. “A Survey on Feature Selection Methods.” Computers & Electrical Engineering 40 (1): 16–28. https://doi.org/10.1016/j.compeleceng.2013.11.024.
82. Dougherty, Edward, Jianping Hua, and Chao Sima. 2009. “Performance of Feature Selection Methods.” Current Genomics 10 (6): 365–74. https://doi.org/10.2174/138920209789177629.
83. Venkatesh, B., & J. Anuradha. “A Review of Feature Selection and Its Methods”. Cybernetics and Information Technologies. 2019; Vol. 19 No. 1: 3–26. DOI: https://doi.org/10.2478/cait-2019-0001.
84. Analytics Vidhya. Feature Selection using Wrapper Method - Python Implementation. 2020. [online] Available at: https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python [Accessed 19 Nov. 2021].
85. Walowe Mwadulo, Mary. “A Review on Feature Selection Methods for Classification Tasks”. International Journal of Computer Applications Technology and Research. 2016; Vol. 5 No. 6: 395–402. DOI: https://doi.org/10.7753/ijcatr0506.1013.
86. Kursa, Miron B., Aleksander Jankowski, & Witold R. Rudnicki. “Boruta – a System for Feature Selection”. Fundamenta Informaticae. 2010; Vol. 101 No. 4: 271–85. DOI: https://doi.org/10.3233/fi-2010-288.
87. Kohavi, Ron, & George H. John. “Wrappers for Feature Subset Selection”. Artificial Intelligence. 1997: Vol. 97 No.1-2) 273–324. DOI: https://doi.org/10.1016/s0004-3702(97)00043-x.

88. Chen, Rung-Ching, Christine Dewi, Su-Wen Huang, & Rezzy Eko Caraka. "Selecting Critical Features for Data Classification Based on Machine Learning Methods". Journal of Big Data. 2020; Vol. 7 No. 1. DOI: https://doi.org/10.1186/s40537-020-00327-4.
89. Gualtiero I. Colombo & Luca Piacentini. "GARS: Genetic Algorithm for the identification of a Robust Subset of features in high-dimensional datasets". BMC Bioinformatics. 2020; Vol. 54 No.1: 123-130. DOI: https://doi.org/10.1186/s12859-020-3400-64
90. Guo, Q, W Wu, D.L Massart, C Boucon & S de Jong. "Feature Selection in Principal Component Analysis of Analytical Data". Chemometrics and Intelligent Laboratory Systems. 2002: Vol. 61 No. 1-2: 123–32. DOI: https://doi.org/10.1016/s0169-7439(01)00203-9.
91. David, Davis. 2020. "Feature Selection by Using Voting Approach." Analytics Vidhya. April 26, 2020. https://medium.com/analytics-vidhya/feature-selection-by-using-voting-approach-e0d1c7182a21#:~:text=How%20does%20it%20 work%3F.
92. Brownlee, Jason. 2020. "Autoencoder Feature Extraction for Classification." Machine Learning Mastery. December 6, 2020. https://machinelearningmastery.com/autoencoder-for-classification/#:~:text =for%20Predictive%20Model-.
93. Yakovyna, V. „Method of software reliability analysis considering its complexity". Radioelectronic and Computer Systems. 2015: No. 2 (72): 127–133.
94. "Sklearn.model_selection.GridSearchCV — Scikit-Learn 0.22 Documentation." 2019. Scikit-Learn.org. 2019. https://scikit-learn.org/ stable/modules/generated /sklearn.model_selection.GridSearchCV.htm/
95. Rudenko, Oleg, Oleksandr Bezsonov, and Oleksandr Romanyk. 2019. "Neural Network Time Series Prediction Based on Multilayer Perceptron." Development Management 17 (1): 23–34. https://doi.org/10.21511/dm.5(1).2019.03.
96. Dash, Ch. Sanjeev Kumar, Ajit Kumar Behera, Satchidananda Dehuri, and Sung-Bae Cho. 2016. "Radial Basis Function Neural Networks: A Topical State-of-The-

Art Survey." Open Computer Science 6 (1). https://doi.org/10.1515/comp-2016-0005.

97. Sherstinsky, Alex. 2020. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network." Physica D: Nonlinear Phenomena 404 (March): 132306. https://doi.org/10.1016/j.physd.2019.132306.

98. Saxena, Utkarsh, Soumen Moulik, Soumya Ranjan Nayak, Thomas Hanne, and Diptendu Sinha Roy. 2021. "Ensemble-Based Machine Learning for Predicting Sudden Human Fall Using Health Data." Edited by Jude Hemanth. *Mathematical Problems in Engineering* 2021 (November): 1–12. https://doi.org/10.1155/2021/8608630.

99. Zhu, K., Zhang, N., Zhang, Q., Ying, S. and Wang, X. (2020). Software defect prediction based on non-linear manifold learning and hybrid deep learning techniques. Computers, Materials & Continua, 65(2), pp.1467–1486.

100. Shakhovska, N., Yakovyna, V., Kryvinska, N.: An improved software defect prediction algorithm using self-organizing maps combined with hierarchical clustering and data preprocessing. In DEXA 2020, LNCS, 12391, 414–424. (2020).

101. Yakovyna, V. S., Seniv, M. M., Symets, I. I., & Sambir, N. B. (2020). Algorithms and software suite for reliability assessment of complex technical systems. Radio Electronics, Computer Science, Control, (4), 163–177. https://doi.org/10.15588/1607-3274-2020-4-16.

102. Yakovyna, V. S., Seniv, M. M., Lytvyn, V. V., & Symets I. I. (2019). Програмний модуль розв'язування систем диференціальних рівнянь Колмогорова-Чепмена для автоматизації надійнісного проектування. Науковий вісник НЛТУ України, 29(5), 141-146. https://doi.org/10.15421/40290528.

103. Яковина, В. С., Сенів, М. М., & Симець, І. І. (2019). Засоби автоматизованого формулювання умов працездатності складних технічних систем. Науковий вісник НЛТУ України, 29(9), 136-141. https://doi.org/10.36930/40290924.

104. Vitaliy Yakovyna, Ivan Symets. A method of high-order Markov chain representation through an equivalent first-order chain for software reliability assessment // Комп'ютерні системи та інформаційні технології. – 2021. – № 3. – С. 66–73. https://doi.org/10.31891/CSIT-2021-5-9.

105. Vitaliy Yakovyna, Ivan Symets. Towards a software defect proneness model: feature selection // Прикладні аспекти інформаційних технологій. – 2021. – Vol. 4, № 4. – P. 354–365. https://doi.org/10.15276/aait.

106. Яковина, В. С., & Симець, І. І. (2021). Прогнозування дефектів програмного забезпечення ансамблем нейронних мереж. Науковий вісник НЛТУ України, 31(6), 104-111. https://doi.org/10.36930/40310616.

107. Yuriy Bobalo, Maksym Seniv, Vitaliy Yakovyna, Ivan Symets Method of Reliability Block Diagram Visualization and Automated Construction of Technical System Operability Condition // Advances in Intelligent Systems and Computing III, vol 871. Springer, Cham. P. 599-610. https://doi.org/ 10.1007/978-3-030-01069-0_43.

108. Yakovyna, Vitaliy, Ivan Symets. 2021. “Reliability Assessment of CubeSat Nanosatellites Flight Software by High-Order Markov Chains.” Procedia Computer Science 192: 447–56. https://doi.org/10.1016/j.procs.2021.08.046.

109. Yuriy Bobalo, Maksym Seniv, Ivan Symets Algorithms of automated formulation of the operability condition of complex technical systems // Perspective technologies and methods in MEMS design (MEMSTECH’2018) : pros. of XIV-th Intern. Conf., 18-22 april 2018, Lviv - Polyana, Ukraine. – P. 220-224.

110. Maksym Seniv, Vitaliy Yakovyna, Ivan Symets Software for visualization of reliability block diagram and automated formulation of operability conditions of technical systems// Perspective technologies and methods in MEMS design (MEMSTECH’2018) : pros. of XIV-th Intern. Conf., 18-22 april 2018, Lviv - Polyana, Ukraine. – P. 191-195.

111. Yuriy Bobalo, Vitaliy Yakovyna, Maksym Seniv, Ivan Symets. Technique of automated construction of states and transitions graph for the analysis of technical

systems reliability. // Proceedings of the 13th International scientific and technnical conference CSIT-2018, 11-14 September 2018. – Lviv, Ukraine 2018. – P. 314.

112. Vitaliy Yakovyna, Maksym Seniv, Ivan Symets. Techniques of Automated Processing of Kolmogorov – Chapman Differential Equation System for Reliability Analysis of Technical Systems. // Proceedings of the 15th Intern. Conf. on The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM'2019, 26 February – 2 March, 2019. – Lviv–Polyana, Ukraine 2019

113. Yakovyna, V., Seniv, M., Symets, I. The Relation between Software Development Methodologies and Factors Affecting Software Reliability // Proceedings of IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT), CSIT 2020, 23-26 Sept. 2020. - Zbarazh, Ukraine. - 377 - 381; https://ieeexplore.ieee.org/document/9321937.

114. “React-D3-Graph.” n.d. Npm. Accessed February 27, 2022. https://www.npmjs.com/package/react-d3-graph.

ДОДАТКИ

Додаток А. Приклад роботи методу визначення функції працездатності

Розглянемо покрокову роботу методу визначення функції працездатності на прикладі схем візуалізації взаємодії елементів технічної системи із 10-ти елементів (рис. А.1).

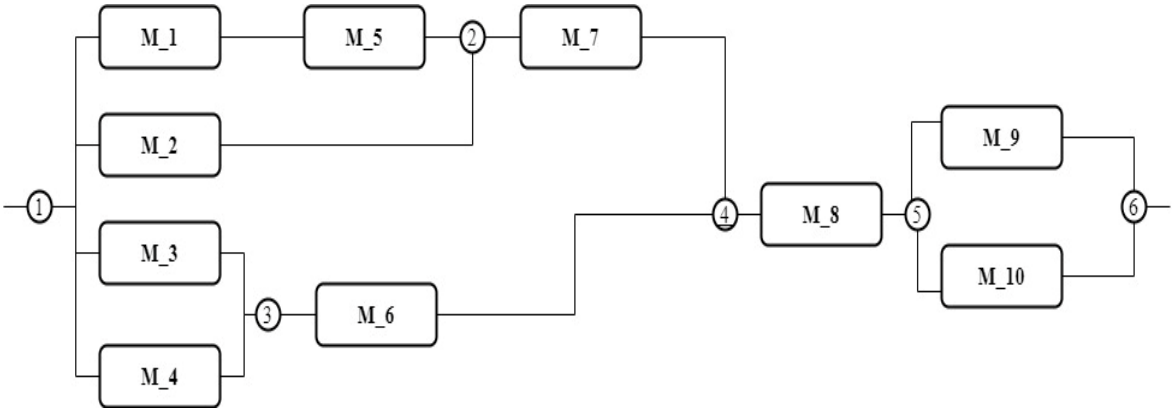


Рис. А.1. Схема візуалізації взаємодії елементів технічної системи із 10-ти елементів

Крок 1. Першим кроком є загальний аналіз схеми і отримання інформації про елементи схеми і як вони взаємодіють між собою, визначення масиву сегментів схеми. Після використання алгоритму визначення масиву сегментів ми отримали наступні значення:

[0] = StartNode = *1*, EndNode=*2*, PathString = ***M_1 AND M_5***

[1] = StartNode = *1*, EndNode=*2*, PathString = ***M_2***

[2] = StartNode = *1*, EndNode=*3*, PathString = ***M_3***

[3] = StartNode = *1*, EndNode=*3*, PathString = ***M_4***

[4] = StartNode = *2*, EndNode=*4*, PathString = ***M_7***

[5] = StartNode = *3*, EndNode=*4*, PathString = ***M_6***

[6] = StartNode = *4*, EndNode=*5*, PathString = ***M_8***

[7] = StartNode = *5*, EndNode=*6*, PathString = ***M_9***

[8] = StartNode = *5*, EndNode=*6*, PathString = ***M_10***

Крок 2. Початок паралельного і послідовного об'єднання сегментів схеми і часткових функцій працездатності (в масиві сегментів повинен залишитися один елемент)

Перша ітерація.

Використання алгоритму паралельного з’єднання. У загальному масиві сегментів є три пари сегментів, які мають одинакові початкові та кінцеві вузли, тому ті сегменти потрібно паралельно об’єднати і перевизначити їхні часткові функції працездатності.

Пара 1:

[0] = StartNode = *1*, EndNode=*2*, PathString = ***M_1 AND M_5***

[1] = StartNode = *1*, EndNode=*2*, PathString = ***M_2***

Пара 2:

[2] = StartNode = *1*, EndNode=*3*, PathString = ***M_3***

[3] = StartNode = *1*, EndNode=*3*, PathString = ***M_4***

Пара 3:

[7] = StartNode = *5*, EndNode=*6*, PathString = ***M_9***

[8] = StartNode = *5*, EndNode=*6*, PathString = ***M_10***

Після об’єднання загальний масив сегментів виглядає наступним чином:

[0] = StartNode = *1*, EndNode=*2*, PathString = ***(M_1 AND M_5) OR M_2***

[1] = StartNode = *1*, EndNode=*3*, PathString = ***M_3 OR M_4***

[2] = StartNode = *2*, EndNode=*4*, PathString = ***M_7***

[3] = StartNode = *3*, EndNode=*4*, PathString = ***M_6***

[4] = StartNode = *4*, EndNode=*5*, PathString = ***M_8***

[5] = StartNode = *5*, EndNode=*6*, PathString = ***M_9 OR M_10***

Після паралельного об’єднання сегментів схеми, потрібно пройтись по масиву сегментів та визначити і об’єднати послідовно розташовані сегменти.

Якщо виконується умова, що в масиві сегментів є один єдиний сегмент у якого кінцевий вузол (EndNode) збігається із початковим вузлом (StartNode) іншого єдиного сегмента із множини всіх сегментів, то такі сегменти розташовані послідовно і їх потрібно з’єднати із використанням логічної операції AND (I).

У нашому масиві сегментів є три пари таких сегментів:

Пара 1:

[0] = StartNode = *1*, EndNode=*2*, PathString = ***(M_1 AND M_5) OR M_2***

[2] = StartNode = *2*, EndNode=*4*, PathString = ***M_7***

Пара 2:

[1] = StartNode = *1*, EndNode=*3*, PathString = ***M_3 OR M_4***

[3] = StartNode = *3*, EndNode=*4*, PathString = ***M_6***

Пара 3:

[4] = StartNode = *4*, EndNode=*5*, PathString = ***M_8***

[5] = StartNode = *5*, EndNode=*6*, PathString = ***M_9 OR M_10***

Після послідовного з'єднання загальний масив сегментів виглядає наступним чином:

[0] = StartNode =*1*, EndNode=*4*, PathString = ***((M_1 AND M_5) OR M_2) AND M_7***

[1] = StartNode = *1*, EndNode=*4*, PathString = ***(M_3 OR M_4) AND M_6***

[2] = StartNode = *4*, EndNode=*6*, PathString = ***M_8 AND (M_9 OR M_10)***

Друга ітерація.

Використання алгоритму паралельного з'єднання. У загальному масиві сегментів є одна пара сегментів, які мають одинакові початкові і кінцеві вузли.

[0] = StartNode =*1*, EndNode=*4*, PathString = ***((M_1 AND M_5) OR M_2) AND M_7***

[1] = StartNode = *1*, EndNode=*4*, PathString = ***(M_3 OR M_4) AND M_6***

Після паралельного об'єднання загальний масив сегментів виглядає наступним чином:

[0] = StartNode =*1*, EndNode=*4*, PathString =

(((M_1 AND M_5) OR M_2) AND M_7) OR ((M_3 OR M_4) AND M_6)

[1] = StartNode = *4*, EndNode=*6*, PathString = ***M_8 AND (M_9 OR M_10)***

Після паралельного об'єднання сегментів схеми, потрібно пройтись по масиву сегментів та визначити і об'єднати послідовно розташовані сегменти.

В нас залишилось два сегменти, які розташовані послідовно, тому після послідовного з'єднання загальний масив сегментів виглядає наступним чином:

[0] = StartNode =*1*, EndNode=*6*, PathString =

((((M_1 AND M_5) OR M_2) AND M_7) OR ((M_3 OR M_4) AND M_6)) AND (M_8 AND (M_9 OR M_10))

Отже, після другої ітерації в масиві сегментів залишився один сегмент, тому метод завершив свою роботу і кінцевою функцією працездатності є значення параметра PathString єдиного елемента в масиві.

Результат: ***((((M_1 AND M_5) OR M_2) AND M_7) OR ((M_3 OR M_4) AND M_6)) AND (M_8 AND (M_9 OR M_10))***

Додаток Б. Приклад роботи методу автоматизації подання процесу маркова вищого порядку еквівалентним процесом першого порядку

Нехай маємо певний початковий граф, що показує модулі і потоки взаємодії між ними для певної тестової системи і потрібно визначити еквівалентний граф для Марковського ланцюга 2-го порядку (рис. Б.1)

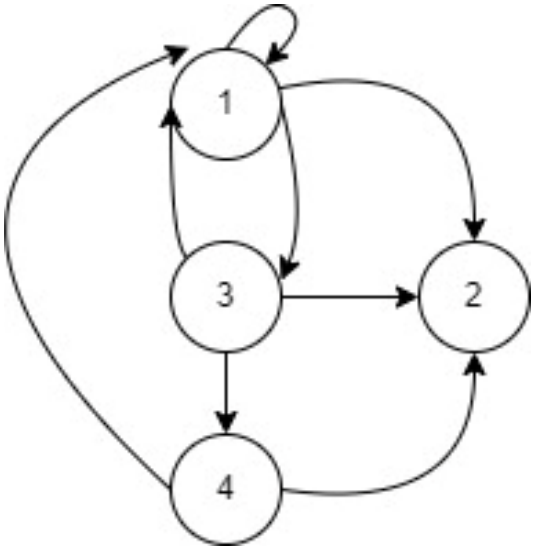


Рис Б.1. Граф станів і переходів тестової системи

Граф станів і переходів відповідно можна представити у вигляді матриці суміжності (табл. Б.1).

Таблиця Б.1 – Матриця суміжності для графа для вхідного графа

i\j	M_1	M_2	M_3	M_4
M_1	1	1	1	0
M_2	0	0	0	0
M_3	1	1	0	1
M_4	1	1	0	0

Матриця читається наступним чином: із M_i вершини можливий перехід у M_j або у вершину M_j можна перейти з вершини M_i .

Крок 1. Потрібно пройтись по всіх вершинах графа і визначити список вершин, які ведуть до кожної із вершин. Повторюємо ми цей процес N-1 раз, оскільки у нас N = 2, потрібно виконати 1-ну ітерацію.

Вершина 1. Як видно із матриці суміжності у вершини 1 є можливість переходу з вершин 1 (петля), 3 і 4. Відповідно дерево для вершини 1 матиме наступний вигляд (рис. Б.2).

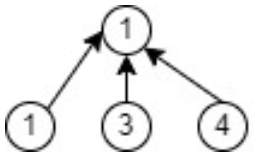


Рис. Б.2. Дерево, яке описує попередні стани, для вершини 1 (N=2)

Вершина 2. Як видно із матриці суміжності у вершини 2 є можливість переходу з вершин 1, 3, 4. Відповідно дерево для вершини 2 матиме наступний вигляд (рис. Б.3).

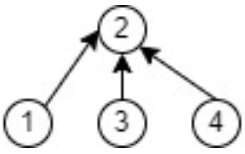


Рис. Б.3. Дерево, яке описує попередні стани, для вершини 2 (N=2)

Вершина 3. Як видно із матриці суміжності у вершини 3 є можливість переходу тільки з вершини 1. Відповідно дерево для вершини 3 матиме наступний вигляд (рис. Б.4).

Рис. Б.4. Дерево, яке описує попередні стани, для вершини 3 (N=2)

Вершина 4. Як видно із матриці суміжності у вершини 4 є можливість переходу тільки з вершини 3. Відповідно дерево для вершини 4 матиме наступний вигляд (рис. Б.5).

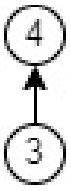


Рис. Б.5. Дерево, яке описує попередні стани, для вершини 4 (N=2)

Отже, після завершення кроку 1, ми маємо 4-и дерева, які відображають батьківські та дочірні вершини, як стани системи і стани, які її породжують (рис. В.6).

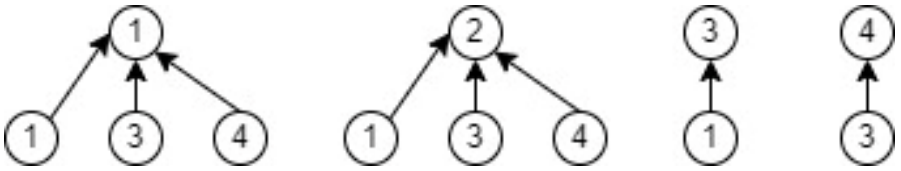


Рис. Б.6. Список дерев для тестової системи (N=2)

Крок 2. На цьому кроці нам потрібно визначити множину станів еквівалентного графа для ланцюга Маркова другого порядку. Для цього нам потрібно пройти всі гілки сформованих дерев (рис. Б.7) і визначити стани, які будуть дорівнювати шляху, що веде від кожного із листків до кореневого вузла.

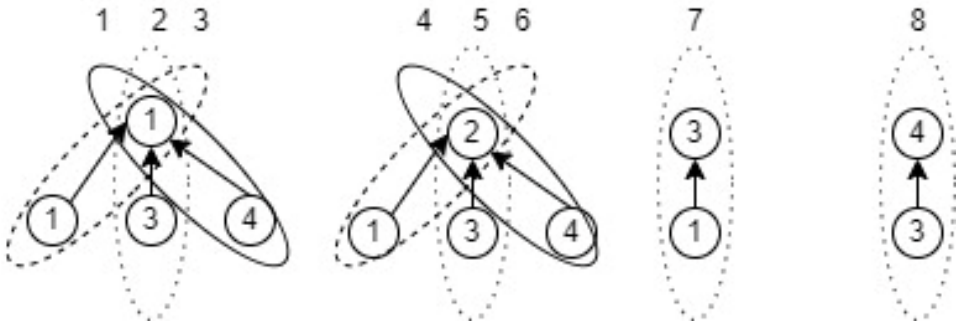


Рис. Б.7. Список дерев для кожної вершини та ідентифікація гілок дерев.

В результаті ми сформували наступні стани еквівалентного графа для ланцюга Маркова другого порядку та занесли їх у таблицю Б.2.

Таблиця Б.2 – Список станів еквівалентного графа для ланцюга Маркова другого порядку.

Перехід у початковому графі	Результуючий стан в еквівалентному графі
$1 \rightarrow 1$	1_1
$3 \rightarrow 1$	1_3
$4 \rightarrow 1$	1_4
$1 \rightarrow 2$	2_1
$3 \rightarrow 2$	2_3

4 → 2	2_4
1 →3	3_1
3 →4	4_3

Базуючись на вхідній матриці суміжності, матриця суміжності для для ланцюга Маркова другого порядку виглядатиме наступним чином (табл. Б. 3):

Таблиця Б.3 – Матриця суміжності для еквівалентний граф 2-го порядку

i\ j	$M_{1\leftarrow1}$	$M_{1\leftarrow3}$	$M_{1\leftarrow4}$	$M_{2\leftarrow1}$	$M_{2\leftarrow3}$	$M_{2\leftarrow4}$	$M_{3\leftarrow1}$	$M_{4\leftarrow3}$
$M_{1\leftarrow1}$	1	0	0	1	0	0	1	0
$M_{1\leftarrow3}$	0	0	0	0	1	0	1	1
$M_{1\leftarrow4}$	0	0	0	1	0	0	1	0
$M_{2\leftarrow1}$	0	0	0	0	0	0	0	0
$M_{2\leftarrow3}$	0	0	0	0	0	0	0	0
$M_{2\leftarrow4}$	0	0	0	0	0	0	0	0
$M_{3\leftarrow1}$	0	0	0	0	1	0	0	1
$M_{4\leftarrow3}$	0	0	1	0	0	0	0	1

Відповідно до множини станів еквівалентний граф для ланцюга Маркова другого порядку буде виглядати так, як показано на рисунку Б.8.

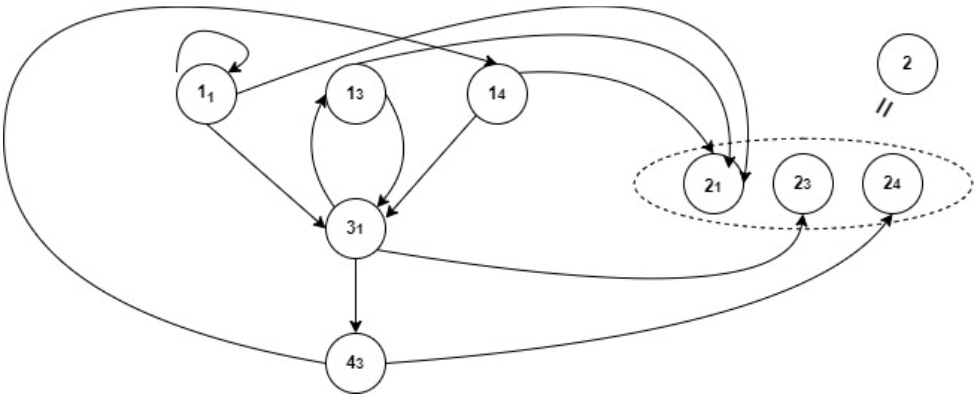


Рис. Б.8. Еквівалентний графік для ланцюга Маркова другого порядку.

Додаток В. Опис метрик коду із набору даних PROMISE

loc - LOC (McCabe's) – рядки коду вимірюються відповідно до підрахунку рядків Маккейба;

v(g) - Cyclomatic Complexity (McCabe's) – цикломатична складність, вимірює кількість "лінійно незалежних шляхів";

ev(g) – Essential Complexity (McCabe's) – суттєва складність;

iv(g) – Design Complexity (McCabe's), складність проектування - це вимірювання складності відображає модулі, які викликають шаблони до своїх безпосередньо підлеглих модулів;

n – Operator and Operand total (Halstead), загальна кількість операндів і операторів в модулі;

v – Volume (Halstead), обсяг на мінімальному виконанні, визначається за формулою (2 + mu2)*log2(2 + mu2), де mu1 – кількість унікальних операторів в модулі, mu2 – кількість унікальних операндів в модулі;

l – Program Length (Halstead), розмір програми, визначається за формулою **l = v/ n;**

d –Difficulty (Halstead), складність, визначається за формулою **d = 1/ l;**

i – Intelligence (Halstead), інтелект, визначається за формулою **i = l * v**

e – Effort (Halstead), зусилля на написання програми/модуля, визначається за формулою **e = v / l;**

b – метрика Halstead;

t – Time (Halstead), час на написання модуля, визначається за формулою **t = e / 18;**

lOCode – LOC (Halstead), кількість рядків коду за Halstead;

lOComment – Line of Comments (Halstead), кількість рядків із коментарями за Halstead;

lOBlank – Line of Blanks (Halstead), кількість порожніх рядків за Halstead;

lOCodeAndComment – Line of Code and Comment (Halstead), загальна кількість рядків коду із коментарями;

uniq_Op – No. of unique operator, кількість унікальних операторів;

uniq_Opnd – No. of unique operand, кількість унікальних операндів;

total_Op – Total operator, загальна кількість операторів;

total_Opnd – Total operand, загальна кількість операндів;

branchCount – Count of Branch, кількість гілок в репозиторії, в яких задіювався/змінювався модуль.

Додаток Г. Програмний код пз для подання процесу маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами

Назва файлу: HOMC_ConfigPage.js

Опис програмного коду: Даний програмний файл містить всю необхідну логіку для побудова графа станів і переходів за допомогою бібліотеки react-d3-graph (конфігурації для візуалізації і побудови графа), введення початкових конфігурацій моделювання і визначення матриці суміжності для вхідного графа.

Програмний код:

Конфігурації для бібліотеки react-d3-graph:

```
this.state.GraphConfig = {
            "automaticRearrangeAfterDropNode": true,
            "collapsible": true,
            "directed": true,
            "focusAnimationDuration": 0.75,
            "focusZoom": 1,
            "freezeAllDragEvents": false,
            "height": 400,
            "highlightDegree": 2,
            "highlightOpacity": 0.2,
            "linkHighlightBehavior": true,
            "maxZoom": 12,
            "minZoom": 0.05,
            "nodeHighlightBehavior": true,
            "panAndZoom": false,
            "staticGraph": false,
            "staticGraphWithDragAndDrop": false,
            "width": 800,
            "d3": {
                "alphaTarget": 0.05,
                "gravity": -250,
                "linkLength": 120,
                "linkStrength": 2,
                "disableLinkForce": false
            },
            "node": {
                "color": "#d3d3d3",
                "fontColor": "black",
```

```
            "fontSize": 10,
            "fontWeight": "normal",
            "highlightColor": "#F85C50",
            "highlightFontSize": 14,
            "highlightFontWeight": "bold",
            "highlightStrokeColor": "#F85C50",
            "highlightStrokeWidth": 1.5,
            "mouseCursor": "crosshair",
            "opacity": 0.9,
            "renderLabel": true,
            "size": 200,
            "strokeColor": "none",
            "strokeWidth": 1.5,
            "svg": "Blue_circle_logo.svg",
            "symbolType": "circle"
        },
        "link": {
            "color": "lightgray",
            "fontColor": "black",
            "fontSize": 8,
            "fontWeight": "normal",
            "highlightColor": "#F85C50",
            "highlightFontSize": 8,
            "highlightFontWeight": "normal",
            "labelProperty": "label",
            "mouseCursor": "pointer",
            "opacity": 1,
            "renderLabel": false,
            "semanticStrokeWidth": true,
            "strokeWidth": 2,
            "markerHeight": 6,
            "markerWidth": 6,
            "strokeDasharray": 0,
            "strokeDashoffset": 0,
            "strokeLinecap": "butt"
        }
```


Основні функції з файлу:

```
generateFirstOrderMatrix(event) {
    var First_Order_Matrix_Data = [];
    var First_Order_Matrix_Columns = [];
    var Node_Names = [];
```

```
        if (this.state.Node_NamesString) {
            Node_Names = this.state.Node_NamesString.split(",").map(function
(item) {
                return item.trim();
            });
        } else {
            for (var i = 0; i < this.state.First_Order_Graph_NodeCount; i++) {
                Node_Names.push("S" + (i+1));
            }
        }
        for (var i = 0; i < this.state.First_Order_Graph_NodeCount; i++) {

            var nodeItem = {};
            nodeItem.id = i + 1;
            nodeItem.nodeName = Node_Names[i];

            for (var j = 0; j < this.state.First_Order_Graph_NodeCount; j++) {
                var exist = false;
                var existVal = 0;

                if (this.state.First_Order_Matrix_Data[i] !== undefined) {
                    exist =
this.state.First_Order_Matrix_Data[i].hasOwnProperty(Node_Names[j]);
                    existVal =
this.state.First_Order_Matrix_Data[i][Node_Names[j]]
                }

                if (exist && existVal> 0)
                    nodeItem[Node_Names[j]] = 1;
                else
                    nodeItem[Node_Names[j]] = 0
            }
            First_Order_Matrix_Data.push(nodeItem);
        }
        First_Order_Matrix_Columns.push({
            dataField: "nodeName",
            text: "i-j"
        });

        for (var i = 0; i < this.state.First_Order_Graph_NodeCount; i++) {
            First_Order_Matrix_Columns.push({
                dataField: Node_Names[i],
                text: Node_Names[i]
```

```
            });
        }
        this.setState({
            First_Order_Matrix_Columns: First_Order_Matrix_Columns,
            First_Order_Matrix_Data: First_Order_Matrix_Data,
            showFirstOrderMatrixSection: true,
            Node_Names: Node_Names
        });
        event.preventDefault();
    }
    generateFirstOrderGraph(event) {

        var First_Order_Graph_Data = [];
        First_Order_Graph_Data.nodes = [];
        First_Order_Graph_Data.links = [];
        var First_Order_Graph_EdgeCount = 0;

        var adjacencyMatrix = [];
        for (var i = 0; i < this.state.First_Order_Graph_NodeCount; i++) {
            adjacencyMatrix[i] = new
Array(parseInt(this.state.First_Order_Graph_NodeCount)).fill(0);
        }

        for (var i = 0; i < this.state.First_Order_Graph_NodeCount; i++) {

            var nodeItem = {};
            nodeItem.id = this.state.First_Order_Matrix_Data[i].nodeName;

            First_Order_Graph_Data.nodes.push(nodeItem);


            for (var j = 0; j < this.state.First_Order_Graph_NodeCount; j++) {

                if
(this.state.First_Order_Matrix_Data[i][this.state.Node_Names[j]] > 0) {

                    First_Order_Graph_EdgeCount++;
                    First_Order_Graph_Data.links.push({ source:
this.state.First_Order_Matrix_Data[i].nodeName, target: this.state.Node_Names[j]
});

                    adjacencyMatrix[i][j] = 1;
                }
```

```
            }

        }
        this.setState({
            showFirstOrderGraphSection: true,
            First_Order_Graph_Data: First_Order_Graph_Data,
            First_Order_AdjacencyMatrix: adjacencyMatrix,
            First_Order_Graph_EdgeCount: First_Order_Graph_EdgeCount
        });
        event.preventDefault();
    }
    async getHOMC_Data() {
        var startTime = performance.now();
        var graphNodes = [];

        for (var i = 0; i < this.state.First_Order_Graph_NodeCount; i++) {

            var nodeItem = {};
            nodeItem.Id = i;
            nodeItem.TopName = this.state.Node_Names[i];
            graphNodes.push(nodeItem);
        }
        const requestOptions = {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ adjacencyMatrix:
this.state.First_Order_AdjacencyMatrix, First_Order_Graph_NodeCount:
parseInt(this.state.First_Order_Graph_NodeCount), HOMC_Number:
parseInt(this.state.HOMC_Number), GraphNodes: graphNodes})
        };

        fetch('HOMC', requestOptions)
            .then(response => response.json())
            .then(data => this.generateHOMCGraph(data));

        var endTime = performance.now();
    }

    generateHOMCGraph(data) {

        var HOMC_Graph_Data = [];
        HOMC_Graph_Data.nodes = [];
        HOMC_Graph_Data.links = [];
```

```
        for (var i = 0; i < data.homc_NodeCount; i++) {

            var nodeItem = {};
            nodeItem.id = data.graphNodes[i].nodeName;
            HOMC_Graph_Data.nodes.push(nodeItem);

            for (var j = 0; j < data.homc_NodeCount; j++) {
                if (data.adjacencyMatrix[i][j] > 0) {
                    HOMC_Graph_Data.links.push({ source:
data.graphNodes[i].nodeName, target: data.graphNodes[j].nodeName});
                }
            }
        }
        var HOMC_Matrix_Data = [];
        var HOMC_Matrix_Columns = [];

        for (var i = 0; i < data.homc_NodeCount; i++) {
            var nodeItem = {};
            nodeItem.id = data.graphNodes[i].id;
            nodeItem.nodeName = data.graphNodes[i].nodeName;

            for (var j = 0; j < data.homc_NodeCount; j++) {
                nodeItem[data.graphNodes[j].nodeName] =
data.adjacencyMatrix[i][j];
            }
            HOMC_Matrix_Data.push(nodeItem);
        }
        HOMC_Matrix_Columns.push({
            dataField: "nodeName",
            text: "i-j"
        });
        for (var i = 0; i < data.homc_NodeCount; i++) {
            HOMC_Matrix_Columns.push({
                dataField: data.graphNodes[i].nodeName,
                text: data.graphNodes[i].nodeName
            });
        }
        this.setState({
            showHOMCGraphSection: true,
            HOMC_Graph_Data: HOMC_Graph_Data,
            HOMC_Graph_NodeCount: data.homc_NodeCount,
            HOMC_Graph_EdgeCount: data.homc_EdgeCount,
```

```
            HOMC_graphNodes: data.graphNodes,
            HOMC_Matrix_Data: HOMC_Matrix_Data,
            HOMC_Matrix_Columns: HOMC_Matrix_Columns,
        });
    }
```

Рендеринг html частини:

```
render() {
        return (
            <div>
                <div className="mb-3" >
                    <h4 className="mb-3" >ПЗ для представлення Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку (HOMC) з додатковими віртуальними станами</h4>
                    <hr/>
                <h5 className="mb-3"  >Крок 1. Початкові конфігурації</h5>
                <Form className="register-form">
                    <Form.Group className="mb-3" controlId="HOMC_Number">
                        <Form.Label>Порядок Марковського процесу вищого порядку</Form.Label>
                        <Form.Control
                            type="text"
                            placeholder="Введіть порядок HOMC"
                            name="HOMC_Number"
                            onChange={this.handleChange}
                        />
                    </Form.Group>
                        <Form.Group className="mb-3" controlId="First_Order_Graph_NodeCount">
                        <Form.Label>Кількість вузлів у графі першого порядку</Form.Label>
                        <Form.Control
                            type="text"
                                placeholder="Введіть кількість вузлів у графі першого порядку"
                                name="First_Order_Graph_NodeCount"
                            onChange={this.handleChange}
                        />
                        </Form.Group>
                        <Form.Group className="mb-3" controlId="Node_NamesString">
                            <Form.Label>Назви вузлів у графі першого порядку (*)</Form.Label>
```

```
                            <Form.Control
                                type="text"
                                placeholder="Введіть назви вузлів у графі першого
порядку (значення вводити через кому)"
                                name="Node_NamesString"
                                onChange={this.handleChange}
                            />
                        </Form.Group>
                    <Button className="mb-3"
                        variant="primary"
                        onClick={this.generateFirstOrderMatrix}>
                        Перейти до створення матриці суміжності
                    </Button>
                    </Form>
                </div>
                {this.state.showFirstOrderMatrixSection && <div className="mb-3"
id="FirstOrderMatrixSection">
                    <hr/>
                    <h5 className="mb-3" >Крок 2. Заповніть матрицю суміжності
для графа першого порядку</h5>
                    <div id="Matrix">
                        <BootstrapTable
                            bootstrap4
                            keyField="id"
                            data={this.state.First_Order_Matrix_Data}
                            columns={this.state.First_Order_Matrix_Columns}
                            cellEdit={cellEditFactory({
                                mode: "click",
                                blurToSave: true
                            })}
                        />
                    </div>
                    <Button className="mb-3"
                        variant="primary"
                        onClick={this.generateFirstOrderGraph}>
                        Згенерувати граф Маркова першого порядку
                    </Button>
                </div>}
                {this.state.showFirstOrderGraphSection && <div
id="FirstOrderGraphSection">
                    <hr />
                    <h5 className="mb-3"  >Крок 3. Візуалізація графа першого
порядку</h5>
```

```
                    <p className="mb-3" >Кількість вузлів у графі:
{this.state.First_Order_Graph_NodeCount}</p>
                    <p className="mb-3" >Кількість ребер у графі:
{this.state.First_Order_Graph_EdgeCount}</p>

                    <div className="mb-3" id="Graph">
                        <Graph style="height: 500px; width: 2800px;"
                            id="graph-id"
                            data={this.state.First_Order_Graph_Data}
                            config={this.state.GraphConfig}
/>
                    </div>
                    <Button className="mb-3"
                        variant="primary"
                        onClick={this.getHOMC_Data}>
                        Згенерувати граф Маркова {this.state.HOMC_Number}-го
порядку
                    </Button>
                </div>}
                {this.state.showHOMCGraphSection && <div id="HOMCGraphSection">
                    <hr />
                    <h5 className="mb-3"  >Крок 5. Граф Маркова
{this.state.HOMC_Number}-го порядку</h5>
                    <p className="mb-3" >Кількість вузлів у графі:
{this.state.HOMC_Graph_NodeCount}</p>
                    <p className="mb-3" >Кількість ребер у графі:
{this.state.HOMC_Graph_EdgeCount}</p>

                       <p className="mb-3" >Список станів:
{this.state.HOMC_graphNodes.map((object, i) => <span>{object.nodeName};{'
'}</span>)}</p>

                    <div className="mb-3" id="Graph_HOMC">
                        <Graph style="height: 500px; width: 2800px;"
                            id="graph-id_Graph_HOMC"
                            data={this.state.HOMC_Graph_Data}
                            config={this.state.GraphConfig}
                        />
                    </div>
                    <h5 className="mb-3" >Матриця суміжності для граф Маркова
{this.state.HOMC_Number}-го порядку</h5>
                    <div id="Matrix_HOMC">
                        <BootstrapTable
```

```
                        bootstrap4
                        keyField="id"
                        data={this.state.HOMC_Matrix_Data}
                        columns={this.state.HOMC_Matrix_Columns}
                    />
                </div>
                <Button className="mb-3"
                    variant="primary">
                   Зберегти матрицю у файл
                </Button>
            </div>}
        </div>
    );
    }
```

Назва файлу: HOMC_Service.cs

Опис програмного коду: даний клас є сервісом в якому реалізована основна логіка методу представлення Марковського процесу вищого порядку у вигляді еквівалентного процесу першого порядку з додатковими віртуальними станами.

Програмний код:

```
public class HOMC_Service
{
    public readonly List<GraphTop> graphTops;
    public readonly byte[,] adjacencyMatrix;
    public HOMC_Service(List<GraphTop> _graphTops, byte[,] adjacencyMatrix)
    {
        this.graphTops = _graphTops; this.adjacencyMatrix = adjacencyMatrix;
    }
    public List<GraphTop> GetInputsForTop(GraphTop top)
    {
        List<GraphTop> topList = new List<GraphTop>();

        for (int indexI = 0; indexI < this.adjacencyMatrix.GetLength(0);
indexI++)
        {
            if (Convert.ToBoolean(this.adjacencyMatrix[indexI, top.Id]))
            {
```

```
                    topList.Add(this.graphTops.First(fr => fr.Id == indexI));
                }
            }
            return topList;
        }
        public List<TreeNode<GraphTop>> GetListOfTreesForTops(int orderNumber)
        {
            List<TreeNode<GraphTop>> listOfTrees = new
List<TreeNode<GraphTop>>();

            foreach (GraphTop graphTop in graphTops)
            {
                TreeNode<GraphTop> rootNode = new TreeNode<GraphTop>(graphTop);
                listOfTrees.Add(rootNode);
            }
            foreach (var tree in listOfTrees)
            {
                int level = 0;
                var allNodes = new List<TreeNode<GraphTop>>();
                allNodes.Add(tree);

                while (level != orderNumber - 1)
                {
                    var nodesForProcessing =
TreeService.GetLeafNodesByRootNode(tree);

                    foreach (var nodeForProcessing in nodesForProcessing)
                    {
                        var inputsForNode =
this.GetInputsForTop(nodeForProcessing.NodeDate);
allNodes.AddRange(nodeForProcessing.AddChildren(inputsForNode));
                    }
                    level++;
                }
            }
            return listOfTrees;
        }
        public List<HOMC_GraphState> Get_HOMC_GraphStates
(List<TreeNode<GraphTop>> listOfTrees)
        {
           List<HOMC_GraphState> HOMC_GraphStates = new List<HOMC_GraphState>();

            foreach (var tree in listOfTrees)
```

```
            {
                var leafNodes = TreeService.GetLeafNodesByRootNode(tree);
                foreach (var leafNode in leafNodes)
                {
                    var path = TreeService.GetBranchByLeafNode(leafNode);
                    HOMC_GraphStates.Add(new HOMC_GraphState { Id =
HOMC_GraphStates.Count, GraphTopPath = path });
                }
            }
            return HOMC_GraphStates;
        }
        public byte[,] Get_AdjacencyMatrix_for_HOMC_GraphStates
(List<HOMC_GraphState> HOMC_GraphStates){
            byte[,] adjacencyMatrix = new byte[HOMC_GraphStates.Count,
HOMC_GraphStates.Count];

            foreach (var currentHOMC_GraphState in HOMC_GraphStates)
            {
                foreach (var previusHOMC_GraphState in HOMC_GraphStates)
                {
                    if
(previusHOMC_GraphState.GraphTopPath[previusHOMC_GraphState.GraphTopPath.Count -
2].TopName == currentHOMC_GraphState.
GraphTopPath[currentHOMC_GraphState.GraphTopPath.Count - 1].TopName){
      adjacencyMatrix[currentHOMC_GraphState.Id, previusHOMC_GraphState.Id] = 1;
                    }
                }
            }
            return adjacencyMatrix;
        }
    }
```

Додаток Д. Код програмної реалізації методу визначення функції працездадності

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using VIPER.Presenters;

namespace Logic
{
      class WorkabilityConditionGenerator
    {
         List<Segment> Segments { get; set; } = new List<Segment>();

        public  string Operation { get; set; } = String.Empty;

        public string WorkabilityCondition { get; set; } = String.Empty;

        public WorkabilityConditionGenerator(List<Segment> segments)
        {
            Segments = segments;
        }
        //метод визначення функції працездатності системи
        public  string GenerateCondition()
        {
            var currentSegments = this.Segments;
          //видалення порожніх сегментів
         //    currentSegments = MergeEmptySegments(currentSegments);

            //головний цикл з використанням алгоритмів послідовного і
паралельного злиття
            do
            {
                // злиття паралельних сегментів системи
                currentSegments = MergeConsistentSegments(currentSegments);

                // злиття послідовних сегментів системи
                currentSegments = MergeParallelSegments(currentSegments);
            } while (currentSegments.Count != 1);

            string condition = ($"({currentSegments[0].SubCondition})");
```

```
            this.WorkabilityCondition = condition;
            Lines.LineBuilders.Clear();

            ConfigurationHelper.CountGetFormula = 0;
            return condition;
        }

        //метод видалення порожніх вузлів
          List<Segment> MergeEmptySegments(List<Segment> segments)
        {
            var emptySegmentIds = new List<int>();

            // знаходження вузлів, які мають порожню  умову працездатності і їх об*єднання з суміжними
            foreach (var emptySegment in segments.Where(w => !Convert.ToBoolean(w.Modules.Count)))
            {
                    foreach (var segment in segments.Where(w => Convert.ToBoolean(w.Modules.Count)))
                    {
                        if (emptySegment.StartNode == segment.EndNode)
                        {
                            segment.EndNode = emptySegment.EndNode;
                            emptySegmentIds.Add(emptySegment.Id);
                        }
                        if (emptySegment.EndNode == segment.StartNode)
                        {
                            segment.StartNode = emptySegment.StartNode;
                            emptySegmentIds.Add(emptySegment.Id);
                        }
                    }
            }
            return segments.Where(w=>!emptySegmentIds.Contains(w.Id)).ToList();
        }
        //алгоритм послідовного злиття елементів
          List<Segment> MergeConsistentSegments(List<Segment> segments)
        {
            var mergedSegmentIds = new List<int>();
            segments = segments.OrderBy(o => o.Id).ToList();
            foreach (var currentSegment in segments)
            {
                foreach (var loopSegment in segments)
                {
```

```
                    var isUniqueInNode = segments.Where(w => w.StartNode ==
currentSegment.EndNode).Count() == 1;

                    var isUniqueOutNode = segments.Where(w => w.EndNode ==
loopSegment.StartNode).Count() == 1;

                    // якщо EndNode поточного елемента дорівнює StartNode
наступного і вони зустрічаються по одному разі в масиві сегментів, то потрібно
об*єднати сегменти послідовно

                    if (currentSegment.Id < loopSegment.Id && isUniqueOutNode &&
isUniqueInNode && currentSegment.EndNode == loopSegment.StartNode)
                    {
                        String[] expressionAnd = { "AND" };
                        String[] expressionOr = { "OR" };

                        var subCondition1 = currentSegment.SubCondition;
                        var subCondition2 = loopSegment.SubCondition;

                        currentSegment.Modules.Union(loopSegment.Modules);
                        currentSegment.EndNode = loopSegment.EndNode;

                        //додавання індексв елемента який вже був використаний
                        mergedSegmentIds.Add(loopSegment.Id);

                        if (String.IsNullOrEmpty(subCondition1))
                        {
                            currentSegment.SubCondition = subCondition2;
                            continue;
                        } else if (String.IsNullOrEmpty(subCondition2))
                        {
                            currentSegment.SubCondition = subCondition1;
                            continue;
                        }

                        //перевірка на попередню операцію, якщо вона змінилась
потрібно додати дужки
                        if (subCondition1.Split(expressionOr,
System.StringSplitOptions.RemoveEmptyEntries).Count() > 1)
                            subCondition1 = $"({subCondition1})";

                        if (subCondition2.Split(expressionOr,
System.StringSplitOptions.RemoveEmptyEntries).Count() > 1)
```

```
                        subCondition2 = $"({subCondition2})";

                    //    var subCondition1LastOperation= subCondition1

                        currentSegment.SubCondition = $"{(subCondition1)}
{LogicOperation.AND} {subCondition2}";

                        if (subCondition1 == "")
                            currentSegment.SubCondition = subCondition2;

                        if (subCondition2 == "")
                            currentSegment.SubCondition = subCondition1;
                    }
                }
            }
            return segments.Where(w =>
!mergedSegmentIds.Contains(w.Id)).ToList();
        }
        //алгоритм паралельного злиття елементів
          List<Segment> MergeParallelSegments(List<Segment> segments)
        {
            var mergedSegmentIds = new List<int>();

            segments = segments.OrderBy(o => o.Id).ToList();

        foreach (var currentSegment in segments)
            {
                foreach (var loopSegment in segments)
                {
                    //якщо існують два або більше сегментів з однаковими
початковими і кінцевими вузлами, то треба злити такі вузли паралельно
                    if (currentSegment.Id<loopSegment.Id &&
currentSegment.StartNode == loopSegment.StartNode && currentSegment.EndNode ==
loopSegment.EndNode)
                    {
                        String[] expressionAnd = { "AND" };
                        String[] expressionOr = { "OR" };

                        var subCondition1 = currentSegment.SubCondition;
                        var subCondition2 = loopSegment.SubCondition;

                        currentSegment.Modules.Union(loopSegment.Modules);
```

```
                        //перевірка на попередню операцію, якщо вона змінилась
потрібно додати дужки
                        if (subCondition1.Split(expressionAnd,
System.StringSplitOptions.RemoveEmptyEntries).Count() > 1)
                            subCondition1 = $"({subCondition1})";

                        if (subCondition2.Split(expressionAnd,
System.StringSplitOptions.RemoveEmptyEntries).Count() > 1)
                            subCondition2 = $"({subCondition2})";

                        currentSegment.SubCondition = $"{subCondition1}
{LogicOperation.OR} {subCondition2}";

                        if (subCondition1 == "")
                            currentSegment.SubCondition = subCondition2;

                        if (subCondition2 == "")
                            currentSegment.SubCondition = subCondition1;

                        //додавання індексв елемента який вже був використаний
                        mergedSegmentIds.Add(loopSegment.Id);
                    }
                }
            }
            return segments.Where(w =>
!mergedSegmentIds.Contains(w.Id)).ToList();
        }
    }
}
```

Додаток Е. Список публікацій за темою дисертації

1. Yakovyna, V. S., Seniv, M. M., Symets, I. I., & Sambir, N. B. (2020). Algorithms and software suite for reliability assessment of complex technical systems. Radio Electronics, Computer Science, Control, (4), 163–177. https://doi.org /10.15588/1607-3274-2020-4-16.

2. Yakovyna, V. S., Seniv, M. M., Lytvyn, V. V., & Symets I. I. (2019). Програмний модуль розв’язування систем диференціальних рівнянь Колмогорова-Чепмена для автоматизації надійнісного проектування. Науковий вісник НЛТУ України, 29(5), 141-146. https://doi.org/ 10.15421/40290528.

3. Яковина, В. С., Сенів, М. М., & Симець, І. І. (2019). Засоби автоматизованого формулювання умов працездатності складних технічних систем. Науковий вісник НЛТУ України, 29(9), 136-141. https://doi.org/ 10.36930/40290924.

4. Vitaliy Yakovyna, Ivan Symets. A method of high-order Markov chain representation through an equivalent first-order chain for software reliability assessment // Комп'ютерні системи та інформаційні технології. – 2021. – № 3. – С. 66–73. https://doi.org/10.31891/CSIT-2021-5-9.

5. Vitaliy Yakovyna, Ivan Symets. Towards a software defect proneness model: feature selection // Прикладні аспекти інформаційних технологій. – 2021. – Vol. 4, № 4. – P. 354–365. https://doi.org/10.15276/aait.

6. Яковина, В. С., & Симець, І. І. (2021). Прогнозування дефектів програмного забезпечення ансамблем нейронних мереж. Науковий вісник НЛТУ України, 31(6), 104-111. https://doi.org/10.36930/40310616.

7. Yuriy Bobalo, Maksym Seniv, Vitaliy Yakovyna, Ivan Symets Method of Reliability Block Diagram Visualization and Automated Construction of Technical System Operability Condition // Advances in Intelligent Systems and Computing III, vol 871. Springer, Cham. P. 599-610. https://doi.org/ 10.1007/978-3-030-01069-0_43.

8. Yakovyna, Vitaliy, Ivan Symets. 2021. “Reliability Assessment of CubeSat Nanosatellites Flight Software by High-Order Markov Chains.” Procedia Computer Science 192: 447–56. https://doi.org/10.1016/j.procs.2021.08.046.

9. Yuriy Bobalo, Maksym Seniv, Ivan Symets Algorithms of automated formulation of the operability condition of complex technical systems // Perspective technologies and methods in MEMS design (MEMSTECH’2018) : pros. of XIV-th Intern. Conf., 18-22 april 2018, Lviv - Polyana, Ukraine. – P. 220-224.

10. Maksym Seniv, Vitaliy Yakovyna, Ivan Symets Software for visualization of reliability block diagram and automated formulation of operability conditions of technical systems// Perspective technologies and methods in MEMS design (MEMSTECH’2018) : pros. of XIV-th Intern. Conf., 18-22 april 2018, Lviv - Polyana, Ukraine. – P. 191-195.

11. Yuriy Bobalo, Vitaliy Yakovyna, Maksym Seniv, Ivan Symets. Technique of automated construction of states and transitions graph for the analysis of technical systems reliability. // Proceedings of the 13th International scientific and technnical conference CSIT-2018, 11-14 September 2018. – Lviv, Ukraine 2018. – P. 314.

12. Vitaliy Yakovyna, Maksym Seniv, Ivan Symets. Techniques of Automated Processing of Kolmogorov – Chapman Differential Equation System for Reliability Analysis of Technical Systems. // Proceedings of the 15th Intern. Conf. on The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM'2019, 26 February – 2 March, 2019. – Lviv–Polyana, Ukraine 2019

13.Yakovyna, V., Seniv, M., Symets, I. The Relation between Software Development Methodologies and Factors Affecting Software Reliability // Proceedings of IEEE 15th International Conference on Computer Sciences and Information Technologies (CSIT), CSIT 2020, 23-26 Sept. 2020. - Zbarazh, Ukraine. - 377 - 381; https://ieeexplore.ieee.org/document/9321937.

Додаток Ж. Акти про впровадження та дослідне випробовування результатів дисертації

ЗАТВЕРДЖУЮ
Проректор з наукової роботи
НУ «Львівська політехніка»
д.т.н., доцент Демидов І.В.
08 квітня 2022 р.

АКТ

про використання результатів дисертаційної роботи
Симця Івана Ігоровича
«Моделі і методи прогнозування та аналізу надійності технічних систем з урахуванням процесу розробки ПЗ»
у процесі виконання держбюджетної науково-дослідної роботи
«Розроблення криптозахищеної системи високошвидкісного передавання даних у діапазонах УВЧ і НВЧ з підвищеними завадостійкістю та відмовостійкістю»
шифр ДБ/Демодуляція

Цей акт складений комісією у складі:
– Бобало Ю.Я. – керівник наукової групи ДБ/Демодуляція, д.т.н., проф., професор кафедри теоретичної радіотехніки та радіовимірювань Національного університету «Львівська політехніка»;
– Горбатий І.В. – відповідальний виконавець наукової групи ДБ/Демодуляція, д.т.н., проф. професор кафедри теоретичної радіотехніки та радіовимірювань Національного університету «Львівська політехніка»;
– Сенів М.М. – член наукової групи ДБ/Демодуляція к.т.н., доц., доцент кафедри програмного забезпечення Національного університету «Львівська політехніка»
про те, що в межах виконання держбюджетної науково-дослідної роботи «Розроблення криптозахищеної системи високошвидкісного передавання даних у діапазонах УВЧ і НВЧ з підвищеними завадостійкістю та відмовостійкістю» (*шифр ДБ/Демодуляція*) на кафедрі теоретичної радіотехніки та радіовимірювань Національного університету «Львівська політехніка» було використано програмний засіб для розрахунку надійнісних характеристик складних технічних систем на основі функції працездатності, який є частиною дисертаційної роботи «Моделі і методи прогнозування та аналізу надійності технічних систем з урахуванням процесу розробки ПЗ» аспіранта Симця Івана Ігоровича.

Програмний засіб для розрахунку надійнісних характеристик складних технічних систем реалізує розроблений у дисертації метод визначення функції працездатності для Марковських моделей надійності, що дає змогу автоматизувати використання цього методу і зменшує вплив людського фактора при визначенні складної логічної функції, і, відповідно, знижує ймовірність внесення помилки при використанні такого методу і моделі.

Програмний засіб дає змогу за допомогою графічних компонент візуалізувати схему взаємодії модулів технічної системи, на основі якої автоматизовано визначається функція працездатності технічної системи, яка дає можливість надалі побудувати граф станів і переходів, систему диференціальних рівнянь Колмогорова-Чепмена і визначити певні показники надійності для системи: для відновлюваної системи – функцію готовності і функцію простою, а для невідновлювальної системи – імовірність безвідмовної роботи протягом часу t.

Члени комісії: _____ Бобало Ю.Я.
_____ Горбатий І.В.
_____ Сенів М.М.

ЗАТВЕРДЖУЮ
Проректор з науково-педагогічної роботи
НУ «Львівська політехніка»
_____ к.т.н., доц. Давидчак О.Р.
"16" березня 2022 р.

АКТ
про впровадження результатів дисертаційної роботи
Симця Івана Ігоровича
«Моделі і методи прогнозування та аналізу надійності технічних систем з урахуванням процесу розробки ПЗ»
у навчальному процесі кафедри «Програмного забезпечення»

Даний акт складений комісією у складі:
д.т.н., проф. Федасюк Д.В. – завідувач кафедри програмного забезпечення;
д.т.н., проф. Журавчак Л.М. – професор кафедри програмного забезпечення;
к.т.н., доц. Сенів М.М. - доцент кафедри програмного забезпечення, лектор дисципліни,
про те, що в навчальному процесі кафедри програмного забезпечення використано результати дисертаційної роботи «Моделі і методи прогнозування та аналізу надійності технічних систем з урахуванням процесу розробки ПЗ» для студентів спеціальності 121 «Інженерія програмного забезпечення» в лекційному курсі та практикумі дисципліни «Теорія надійності програмних систем».

Вивчення і аналіз надійності із використання розроблених у дисертації методу визначення функції працездатності для Марковських моделей надійності та методу автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку дають змогу підвищити точність оцінки надійності складних програмних систем із нижчими затратами часу під час використання моделей надійності. Розроблене відповідне програмне забезпечення дає змогу автоматизувати процес використання розроблених методів, зменшує вплив людського фактору, і, відповідно, знижує ймовірність внесення помилки при використанні даних моделей.

Члени комісії: _____ Федасюк Д.В.
_____ Журавчак Л.М.
_____ Сенів М.М.

ЗАТВЕРДЖУЮ

Директор ПП «Лінк Ап Студіо»

"16" вересня 2022 р.

АКТ

про дослідне випробування результатів

дисертаційного дослідження аспіранта кафедри програмного забезпечення

Національного університете «Львівська політехніка»

Симця Івана Ігоровича

Даний акт складений про те, що метод прогнозування дефектності ПЗ на основі стекового ансамблю нейронних мереж (нейронна мережа на основі радіально-базисних функції, рекурентної нейронної мережі та довгої короткострокової пам'яті), розроблені під час виконання дисертаційної роботи Симця Івана Ігоровича на тему «Моделі і методи прогнозування та аналізу надійності технічних систем з урахуванням процесу розробки ПЗ» пройшло дослідне випробування на ПП «Лінк Ап Студіо» для прогнозування дефектності комерційних програмних систем.

Даний метод дозволяє з високою точністю класифікувати програмні модулі на дефектні і ті що без дефектів на основі визначених метрик коду: (кількість рядків коду за МакКейбом (loc), загальна кількість операндів і операторів (N), функціональність модуля (I), обсяг на мінімальному виконанні (V), зусилля на написання модуля (E), цикломатична складність за МакКейбом (v(g)), кількість гілок в репозиторії, в яких змінювався модуль (branchCount)).

Використання розробленого методу дає можливість підвищити точність класифікації програмних модулів на дефектні і без дефектів з 79-86% до 92%.

Даний підхід дозволяє визначати потенційно небезпечні програмні модулі з точки зору дефектності і вжити заходів для усунення цих проблем в модулі на етапі розробки ПЗ або для більш ретельного тестування певного функціоналу на дефекти на етапі тестування.

Даний акт не є підставою для взаємних фінансових розрахунків.

edvantis

ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ "ЕДВАНТІС"
ТзОВ «ЕДВАНТІС»
ЄДРПОУ 33648459
Факс: (032) 232 17 24, ел.пошта: contact@edvantis.com

ЗАТВЕРДЖУЮ
Директор ТзОВ «Едвантіс»
_____ Олександр ГЛАЗУНОВ
"15" Березня 2022 р.

АКТ

про дослідне випробування результатів
дисертаційного дослідження аспіранта кафедри програмного забезпечення
Національного університете «Львівська політехніка»
Симця Івана Ігоровича

Даний акт складений про те, що метод автоматизації подання процесу Маркова вищого порядку еквівалентним процесом першого порядку з додатковими віртуальними станами та відповідне програмне забезпечення,, розроблені під час виконання дисертаційної роботи Симця Івана Ігоровича на тему «Моделі і методи прогнозування та аналізу надійності технічних систем з урахуванням процесу розробки ПЗ» пройшли дослідне випробування на ТзОВ «Едвантіс» для оцінювання якості розроблених програмних систем, зокрема показників їх надійності.

Використання розробленого метода і ПЗ дозволяє автоматизувати процес аналізу надійності ПЗ на етапах тестування і супроводу за допомогою моделі надійності у вигляді ланцюга Маркова вищого порядку і тим самим дає змогу підвищити достовірність оцінювання інтенсивності відмов та середнього часу між відмовами комерційних програмних систем. Верифікація цього методу на прикладі розрахунку функції готовності дала можливість підвищити достовірність оцінки до 50% порівняно з моделлю надійності першого порядку, та до 10% у випадку використання моделей другого та третього порядку.

Даний акт не є підставою для взаємних фінансових розрахунків.