**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Національний університет кораблебудування імені адмірала Макарова**

Навчально-науковий інститут комп'ютерних наук та управління проєктами
Кафедра програмного забезпечення автоматизованих систем
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма «Інженерія програмного забезпечення»

«Допущений до захисту»
Завідувач кафедри
_____ проф. Приходько С.Б.
«___» _____ 2024 р.

# *КВАЛІФІКАЦІЙНА РОБОТА*

**на здобуття ступеня вищої освіти «магістр»**

**на тему**: ***Математична модель для визначення якості десктопних застосунків на C# та розробка програмного забезпечення для її реалізації***

Виконав: здобувач групи 6151м
Казьмін І.В.
(підпис, ПІБ)

Керівник роботи:
доцент, к.ф.-м.н., доцент
(посада, науковий ступень вчене звання)
Латанська Л.О.
(підпис, ПІБ)

Миколаїв – 2024 р.

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Національний університет кораблебудування імені адмірала Макарова**

Навчально-науковий інститут комп'ютерних наук та управління проєктами
Кафедра програмного забезпечення автоматизованих систем
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»
Гарант освітньої програми
_____проф. С.Б.Приходько
(підпис)
« 14 » жовтня 2024 р.

***ЗАВДАННЯ***
***НА КВАЛІФІКАЦІЙНУ РОБОТУ***
**на здобуття ступеня вищої освіти «магістр»**

Студенту Казьміну Ігорю Віталійовичу
(Прізвище, ім'я, по батькові)

1. Тема роботи: Математична модель для визначення якості десктопних застосунків на C# та розробка програмного забезпечення для її реалізації

Керівник роботи Латанська Людмила Олексіївна
Затверджені наказом ректора № 1070-УЧ від « 11 » 10 2024 р.
2. Термін подання роботи: 09.12.2024 р.
3. Вихідні дані по роботі: _____

4. Перелік питань, що належать до розробки (найменування розділів) _____

Вступ (Актуальність теми. Зв'язок роботи з науковими програмами, планами, темами. Мета і завдання дослідження. Об'єкт дослідження. Предмет дослідження. Методи дослідження. Наукова новизна одержаних результатів. Практичне значення одержаних результатів. Особистий внесок здобувача. Огляд літератури та обґрунтування необхідності проведення досліджень за обраною темою; Викладення результатів власних досліджень з висвітленням того нового, що пропонується; Розділ з розробки програмного забезпечення; Організаційно-економічний розділ; Розділи з охорони праці та охорони навколишнього середовища; Висновки; Список використаних джерел; Додатки (технічне завдання, текст програми, опис програми, інструкція користувача, програма і методика випробувань програмного забезпечення)

5. Перелік презентаційних матеріалів 1. Титульний слайд; 2. Актуальність; 3. Мета та завдання; 4. об'єкт, предмет та методи дослідження; 5. Наукова новизна та практичне значення отриманих результатів; 5. Аналіз сучасних методів оцінювання якості програмного забезпечення; 6. Обгрунтування необхідності проведення дослідження за обраною темою; 8. Збір емпіричних даних проектів; 9. Нормалізація даних та вилучення викидів; 10. Побудови двофакторної регресійної моделі; 11. Порівняння математичних моделей оцінювання RFC; 12. Постановка задачі на розробку ПЗ; 13. Діаграма варіантів використання ПЗ; 14. Діаграма діяльності варіанту використання «Імпорт значень метрик»; 15. Діаграма класів ПЗ для оцінювання якості десктоп застосунків на C#; 16 Вибір засобів розробки; 17. Результати розробки ПЗ; 18. Висновки; 19. Подяка.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання 14.10.2024 р.

## КАЛЕНДАРНИЙ ПЛАН

| Номер | Назва етапів роботи | Терміни виконання | Примітка |
|---|---|---|---|
| 1. | Підготовка розділу ВСТУП | 16.10.2024 | НС* |
| 2. | Підготовка розділу з огляду літератури та обґрунтування необхідності проведення досліджень за обраною темою | 18.10.2024 | НС* |
| 3. | Підготовка розділу (ів) за результатами власних досліджень | 21.10.2024 | НС* |
| 4. | Підготовка розділу з розробки програмного забезпечення | 27.11.2024 | |
| 5. | Підготовка організаційно-економічного розділу | 29.11.2024 | |
| 6. | Підготовка розділу з охорони праці | 02.12.2024 | |
| 7. | Підготовка розділу з охорони навколишнього середовища | 04.12.2024 | |
| 8. | Підготовка розділу ВИСНОВКИ | 05.12.2024 | |
| 9. | Оформлення списку використаних джерел та додатків | 06.12.2024 | |
| 10. | Подання на кафедру ПЗАС тексту остаточного варіанту роботи, підписаного її керівником, у роздрукованому та електронному форматі разом із заявами щодо самостійності виконання роботи та ідентичності друкованої та електронної версії роботи (Додатки 1 і 2 «Порядку здійснення заходів з перевірки робіт на наявність текстових збігів/ідентичності/схожості із використанням програмно-технічних засобів», який введений в дію наказом ректора НУК за №20 від 20.01.2020 р.) | 09.12.2024 | не пізніше, ніж за 14 діб до захисту (згідно п.4.1 зазначеного Порядку) |
| 11. | Підготовка презентації та доповіді | 13.12.2024 | |
| 12. | Попередній захист роботи на засіданні кафедри ПЗАС | 16.12.2024 | |
| 13. | Подання на кафедру ПЗАС електронних версії наступних документів у форматі pdf: кваліфікаційної роботи; файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК за №287-уч від 19.05.2020 р.); презентації доповіді | 19.12.2024 | |

* - за результатами наукового стажування (НС), яке було з 02.09.2024 по 13.10.2024 р.

**Студент** _____ Казьмін Ігор Віталійович
(підпис) (ПІБ)

**Керівник роботи** _____ Латанська Людмила Олексіївна
(підпис) (ПІБ)

# РЕФЕРАТ

Казьмін Ігор Віталійович

«Математична модель для визначення якості десктопних застосунків на C# та розробка програмного забезпечення для її реалізації»

Кваліфікаційна робота на здобуття ступеня вищої освіти магістр зі спеціальності 121 – «Інженерія програмного забезпечення». Національний університет кораблебудування імені адмірала Макарова. Миколаїв, 2024р.

**Обсяг роботи:** 114 стор., 14 табл., 22 рис., 49 використаних джерел, 5 додатків.

**Актуальність теми роботи** обумовлена швидким розвитком вимог до програмного забезпечення, зростанням складності застосунків та підвищенням очікувань користувачів. Використання математичних моделей для визначення показників, таких як RFC, дозволяє знизити рівень суб'єктивності під час оцінювання, автоматизувати аналіз і забезпечити високий рівень точності та об’єктивності.

**Мета та завдання дослідження. Метою** є підвищення достовірності оцінювання метрики RFC від CBO та WMC для десктоп застосунків мовою програмування C# для визначення їх якості

**Завдання дослідження:** виконати аналіз попередніх досліджень, існуючих моделей оцінювання якості програмного забезпечення; обґрунтувати необхідність побудови двофакторної нелінійної регресійної моделі для оцінювання метрики RFC для визначення якості десктоп застосунків на C#; побудувати двофакторну нелінійну регресійну модель для оцінювання метрики RFC для застосунків, що створюються з використанням мови програмування C#, в залежності від показників метрик CBO, WMC; розробити програмне забезпечення для реалізації побудованої математичної моделі оцінювання метрики RFC для визначення якості десктоп застосунків на C#.

**Об’єктом дослідження є** процес оцінювання метрики RFC десктоп застосунків на мові програмування C# для визначення їх якості.

**Предметом дослідження є** математичні моделі для оцінювання метрики RFC десктоп застосунків на мові C# для визначення їх якості.

**Методи дослідження**. Для досягнення поставлених цілей були використані підходи теорії ймовірностей, математичної статистики, побудови математичних моделей, регресійного аналізу, а також принципи об'єктно-орієнтованого програмування.

**Наукова новизна отриманих результатів** полягає у вдосконаленні математичної моделі, призначеної для оцінювання метрики RFC десктоп застосунків на мові C# для визначення їх якості. Це було досягнуто шляхом застосування нормалізуючого перетворення Бокса-Кокса, очистки значень метрик від викидів за допомогою квадрату відстані Махаланобіса, використання регуляризації для зменшення впливу мультиколінеарності, що дозволило підвищити достовірність оцінювання у порівнянні з іншими моделями.

**Практичне значення отриманих результатів** полягає у створенні програмного забезпечення, яке реалізує вдосконалену математичну модель для оцінювання метрики RFC, залежної від CBO та WMC, для десктоп застосунків на C# та визначає якість застосунків.

**Ключові слова:** оцінювання якості, нормалізуюче перетворення Бокса-Кокса, нелінійна регресійна модель, C#, регуляризація, квадрат відстані Махаланобіса, мультиколінеарність, RFC.

# ABSTRACT

Kazmin Ihor

«Mathematical model for identifying the quality of desktop applications in C# and development of the software for its implementation»

The qualification work for the degree of higher education Master`s degree in specialty 121 – «Software Engineering». Admiral Makarov National University of Shipbuilding. Mykolaiv, 2024.

**The qualification work** is presented on the 114 pages of typewritten text, contains 14 tables, 22 figures, 49 references and 5 appendices.

**Relevance of the topic of the work.** The relevance of the research is due to the rapid development of software requirements, the increasing complexity of applications, and the growing expectations of users. The use of mathematical models for evaluating the RFC metric helps automate the analysis process, reduce subjective assessments, and achieve accuracy in quality analysis.

**The goal and objectives of the research.** The goal of the research is to enhance the accuracy and reliability of RFC metric evaluation from CBO and WMC for desktop applications developed in C# through the use of a two-factor nonlinear regression model. Research tasks: analyze previous studies and existing models for evaluating software quality; justify the need to build a two-factor nonlinear regression model for evaluating the RFC metric to determine the quality of C# desktop applications; construct a two-factor nonlinear regression model for evaluating the RFC metric of desktop applications written in C#, based on CBO and WMC metric indicators; develop software to implement the constructed mathematical model for evaluating the quality of desktop applications created using the C# programming language.

**The object of the research** is the process of evaluating the RFC metric of desktop applications in the C# programming language to determine their quality.

**The subject of the research** is a mathematical model for evaluating the RFC

metric of desktop applications in the C# programming language to determine their quality.

**Research methods**. The following methods were used to achieve the research goals: probability theory, mathematical statistics, mathematical modeling, regression analysis, and the principles of object-oriented programming.

**Scientific novelty of the obtained results.** The scientific novelty lies in the improvement of the mathematical model designed to evaluate the RFC metric of desktop applications in C#. This was achieved through the application of the Box-Cox normalization transformation, outlier cleaning using Mahalanobis distance squared, and regularization to reduce the impact of multicollinearity, which enhanced the reliability of evaluations compared to other models.

**The practical significance of the obtained results.** The practical significance lies in the creation of software that implements the improved mathematical model for evaluating the RFC metric, dependent on CBO and WMC, for desktop applications in C# and determines their quality.

**Keywords:** quality evaluation, Box-Cox normalizing transformation, nonlinear regression model, C#, regularization, Mahalanobis distance squared, multicollinearity, RFC.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

ООП – об'єктно-орієнтованого програмування

RFC – Response for Class

CBO – Coupling Between Object classes

WMC – Weighted Methods per Class

DIT – Depth of Inheritance Tree

LCOM – Lack of Cohesion in Methods

NOC – Number of Children

Box-Cox – Box Cox Transformation

# ВСТУП

У сучасному світі якість програмного забезпечення (ПЗ) є одним із ключових факторів успіху будь-якого програмного продукту. Особливу важливість це питання набуває для десктопних застосунків, які продовжують відігравати значну роль у багатьох галузях, включаючи бізнес, освіту, медицину та розваги.

Десктопні програми часто використовуються для виконання критично важливих завдань, тому їх якість безпосередньо впливає на ефективність роботи користувачів і організацій, а також на їхню конкурентоспроможність.

Важливим є те, що швидкість розвитку сучасного світу, сильно зросла. Через це, програмне забезпечення постійно вдосконалюється і потребує більше ресурсів, контроль якості в таких задачах є неймовірно важливим.

Проблема задачі оцінка якості ПЗ полягає в тому, що на сьогодні не існує єдиного підходу до оцінювання якості програмного забезпечення. Більшість методів зосереджуються на окремих аспектах, таких як продуктивність, інтерфейс, розмір чи стабільність. Однак для комплексної оцінки якості важливо враховувати різні чинники.

У контексті ООП основними метриками визначення якості можуть бути RFC, CBO та WMC [1], які дозволяють аналізувати структуру, взаємозв'язки і складність коду, зокрема, щільність дефектів, тобто співвідношення кількості дефектів до розміру ПЗ, вираженого в кількості рядків коду або функціональних точок. Цей показник широко використовується в багатьох комерційних програмних системах.

Також, для визначення кількості дефектів програмного забезпечення застосовуються моделі, що базуються на регресійному аналізі та нормалізуючих перетвореннях випадкових величин.

**Актуальність даної теми дослідження** обумовлена швидким розвитком вимог до програмного забезпечення, зростанням складності застосунків та підвищенням очікувань користувачів. Використання математичних моделей для

визначення показників, таких як RFC, дозволяє знизити рівень суб'єктивності під час оцінювання, автоматизувати аналіз і забезпечити високий рівень точності та об’єктивності.

У рамках цієї роботи буде проведено огляд існуючих методів оцінювання якості, розроблено математичну модель оцінювання метрики RFC десктоп застосунків на мові C# і реалізовано програму для автоматизації цього процесу.

Результати можуть бути використані для оцінювання якості програмного забезпечення в різних галузях, що сприятиме ефективнішій розробці та використанню застосунків.

Таким чином, тема цієї роботи має значення як для теоретичного аналізу, так і для практичного застосування в індустрії розробки програмного забезпечення.

**Мета і завдання дослідження**. Метою є підвищення достовірності оцінювання метрики RFC від CBO та WMC для десктоп застосунків мовою програмування C# для визначення їх якості

Для досягнення поставленої мети потрібно виконати наступні завдання:

– провести огляд існуючих методів та математичних моделей для оцінювання якості програмного забезпечення;

– провести аналіз метрик оцінювання якості ПЗ, їх переваги та недоліки;

– зібрати дані для аналізу з реальних проєктів, створених на мові C# та виконати обчислення обраних метрик якості для цих проєктів;

– побудувати математичну модель, яка дозволяє оцінювати метрику RFC залежно від CBO та WMC;

– розробити програмне забезпечення для автоматизованого оцінювання метрики RFC та висновку щодо якості ПЗ на основі запропонованої математичної моделі;

– провести тестування системи на реальному проєкті;

– проаналізувати результати системи та оцінити достовірність математичної моделі на основі зібраних даних;

– надати рекомендації щодо використання розробленого інструменту в практичній діяльності.

**Об’єктом дослідження є** процес оцінювання метрики RFC десктоп застосунків на мові програмування C# для визначення їх якості.

**Предметом дослідження є** математичні моделі для оцінювання метрики RFC десктоп застосунків на мові C# для визначення їх якості.

**Методи дослідження**. Для досягнення поставлених цілей були використані підходи теорії ймовірностей, математичної статистики, побудови математичних моделей, регресійного аналізу, а також принципи об'єктно-орієнтованого програмування.

**Наукова новизна отриманих результатів** полягає у вдосконаленні математичної моделі, призначеної для оцінювання метрики RFC десктоп застосунків на мові C# для визначення їх якості. Це було досягнуто шляхом застосування нормалізуючого перетворення Бокса-Кокса, очистки значень метрик від викидів за допомогою квадрату відстані Махаланобіса, використання регуляризації для зменшення впливу мультиколінеарності, що дозволило підвищити достовірність оцінювання у порівнянні з іншими моделями.

**Практичне значення отриманих результатів** полягає у створенні програмного забезпечення, яке реалізує вдосконалену математичну модель для оцінювання метрики RFC, залежної від CBO та WMC, для десктоп застосунків на C# та визначає якість застосунків.

**Особистий внесок автора**. Кваліфікаційна робота була виконана самостійно. Усі результати, представлені в роботі, отримані автором особисто.

# 1 СУЧАСНИЙ СТАН ОЦІНЮВАННЯ ЯКОСТІ ЗАСТОСУНКІВ

Якість програмного забезпечення – це багатовимірне поняття, яке охоплює різні аспекти, починаючи від технічних характеристик і закінчуючи зручністю для кінцевого користувача. У сучасному світі вимоги до якості ПЗ постійно зростають через швидкий розвиток технологій, посилення конкуренції на ринку та високі очікування користувачів.

## 1.1 **Сучасні вимоги до якості програмного забезпечення**

Одним із базових документів для оцінювання якості ПЗ є стандарт ISO/IEC 25002:2024 [2], який визначає вісім ключових характеристик якості [3]:

– **функціональна придатність**: відповідність ПЗ своїм функціональним вимогам. Наприклад, система для обліку продажів повинна точно розраховувати суми та генерувати звіти;

– **продуктивність**: час відгуку та ефективність роботи системи під навантаженням. Системи, як от банківські застосунки, повинні забезпечувати швидке виконання транзакцій навіть у пікові години;

– **сумісність**: можливість інтеграції з іншими системами та платформами. В цьому випадку можна зрозуміти, що бухгалтерські програми, наприклад, повинні взаємодіяти з податковими сервісами;

– **зручність використання**: інтуїтивний інтерфейс і простота роботи. Для прикладу, популярні текстові редактори, такі як Microsoft Word, мають зрозумілий інтерфейс для широкого кола користувачів;

– **надійність**: стабільність роботи програми. Складні системи, які є критичними, повинні працювати без збоїв, але це також стосується і звичайних застосунків;

– **безпека**: захист даних і конфіденційності. Наприклад, мобільні банківські додатки повинні захищати дані користувача за допомогою багатофакторної автентифікації;

– **підтримка**: легкість модифікації та оновлення коду. В цьому випадку системи управління контентом (CMS) повинні дозволяти адміністратору швидко вносити зміни;

– **портативність**: можливість роботи на різних пристроях або платформах. Застосунки повинні працювати однаково добре на різних пристроях.

Модель якості програмного забезпечення наведена на рис. 1.1.

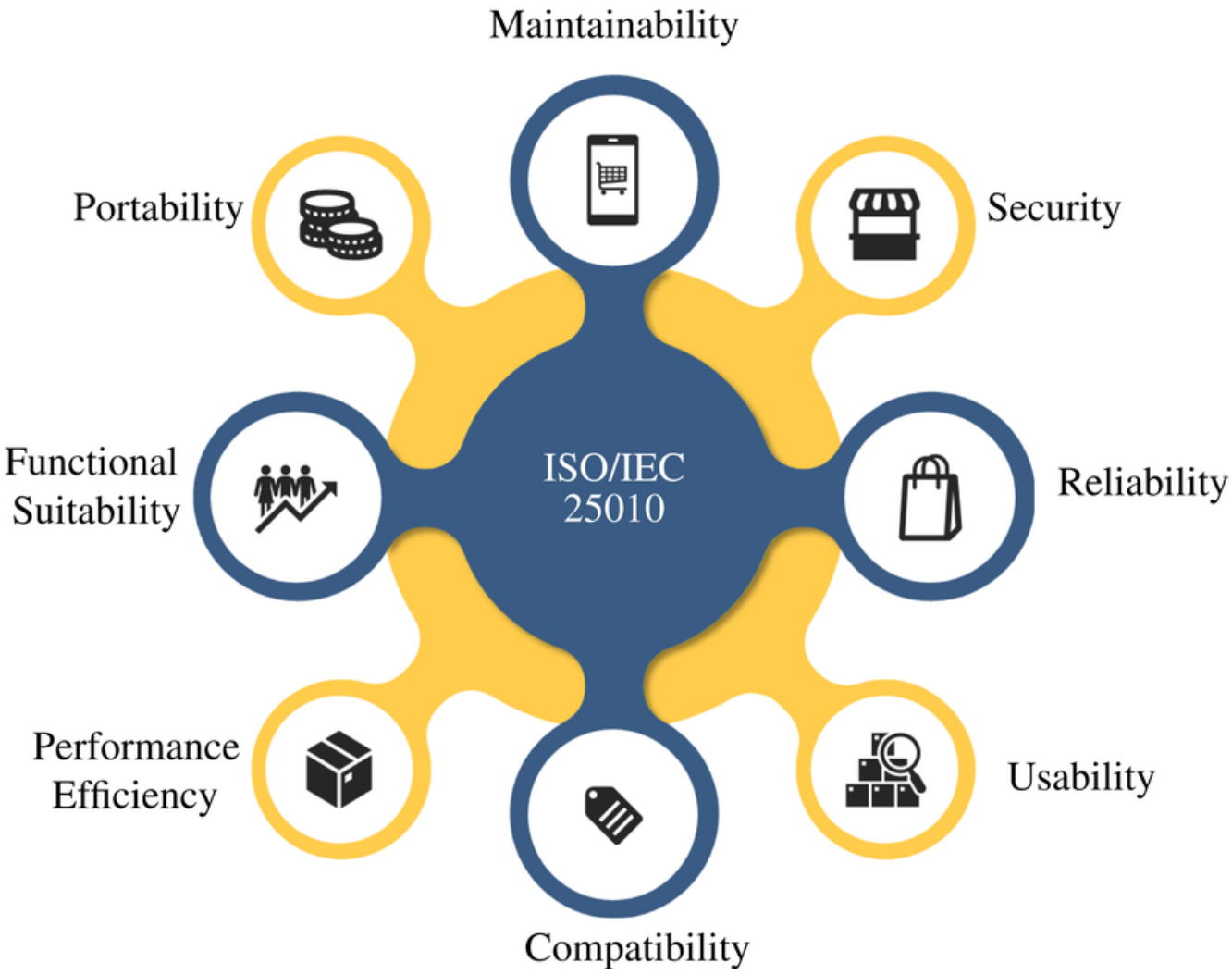


Рисунок 1.1 – Модель якості ПЗ за стандартом стандарт ISO/IEC 25002:2024 [4]

Ці вимоги застосовуються у багатьох сферах сучасного життя, наприклад у галузі фінансів системи, такі як SWIFT. У сфері охорони здоров'я медичні застосунки повинні бути функціонально надійними та захищати персональні дані пацієнтів згідно з GDPR. А ігрові платформи Steam чи Epic Games Store мають

високу вимогливість до сумісності та продуктивності.

Якість програмного забезпечення є одним із ключових показників його успішності та ефективності. Оцінювання якості таких застосунків є важливою задачею для забезпечення їхньої надійності, функціональності, продуктивності та інших аспектів, що визначені стандартом ISO/IEC 25010, а також з використанням нового стандарту ISO/IEC 25002:2024.

## 1.2 **Огляд та аналіз останніх публікацій з питань оцінювання якості програмного забезпечення**

Дослідження задачі оцінювання якості ПЗ набуло великої популярності за останні роки, досліджуються різні впливи параметрів математичних моделей, навантаження системи та метрики. Для більшого розуміння задачі, її проблематики та підходів вирішення, варто не тільки проаналізувати деякі дослідження, а і визначити характеристики.

Наприклад, дослідження [5] аналізує зв'язок між модульністю програмного забезпечення (згідно зі стандартом ISO 25010) і метриками Chidamber and Kemerer (CK Metrics) [6], такими як CBO, DIT, LCOM, WMC, RFC і NOC.

Модульність часто асоціюється з такими аспектами, як гнучкість та складність, однак досі не було науково доведеного кореляційного зв'язку між CK метриками та модульністю.

У межах цього дослідження автори застосовували кількісний підхід для аналізу залежностей між CK метриками та аспектами модульності за допомогою методів кореляції Пірсона і Спірмена [7]. Вони вручну обчислювали значення модульності на основі стандарту ISO 25010 та значення кожної метрики CK для заданого набору даних. Метою було виявити рівень взаємозв'язку між метриками і визначити, які з них впливають одна на одну.

Це дослідження демонструє, що певні метрики CK мають суттєвий вплив на модульність, але цей вплив не є однаковим для всіх метрик.

У дослідженні [8] фокусуються на оцінюванні якості програмного забезпечення з відкритим кодом, розробленого на мові Kotlin, яка набуває все

більшої популярності як для створення програмних інструментів, так і для компонентів мобільних застосунків.

Оскільки вибір якісного програмного забезпечення є критично важливим, автори дослідження звертаються до використання метрик, таких як середній RFC, середній CBO та середній WMC, для визначення якості.

Основна проблема, яку вирішує це дослідження, полягає у відсутності моделей та методів, що базуються на програмних метриках, для оцінювання якості відкритих додатків, створених на Kotlin. Для заповнення цього пробілу автори застосували техніку оцінювання якості за допомогою прогнозування та довірчих інтервалів нелінійної регресії. Регресійна модель була побудована з використанням середнього RFC як залежної змінної, а середнього CBO та середнього WMC – як незалежних змінних. Для точності прогнозування автори використали триваріантну трансформацію Бокса-Кокса.

Для моделювання було використано дані 52 відкритих програм, розроблених на Kotlin і розміщених на GitHub. За допомогою отриманих метрик побудовано прогнозні та довірчі інтервали для середнього RFC, що дозволило оцінити залежність якості програмного забезпечення від його структурних характеристик.

Результати дослідження підкреслюють можливість використання нелінійної регресії та програмних метрик для визначення якості відкритого програмного забезпечення. Запропонований підхід надає розробникам інструмент для об'єктивного аналізу коду і вибору високоякісного програмного забезпечення, сприяючи підвищенню стандартів у спільнотах відкритого коду.

Дослідження [9] присвячене використанню програмних метрик для визначення якості програмного забезпечення, де аналізується метрика RFC, яка використовується для оцінювання якості програмних систем на рівні застосунків. Традиційно оцінка базується лише на середніх значеннях метрик, що може бути недостатнім для точного аналізу. Водночас відомо, що RFC має залежність від інших метрик, таких як CBO і WMC.

Для вирішення цієї проблеми автори запропонували методику, яка дозволяє оцінювати якість програмного забезпечення за допомогою довірчих і прогнозних

інтервалів нелінійної регресії. Ця методика розширює підхід до оцінювання, вводячи залежність RFC від метрик CBO і WMC як предикторів. Методика включає класифікацію програмного забезпечення за рівнем якості залежно від розташування значення RFC у межах визначених інтервалів, а саме, якщо:

– значення RFC у межах довірчого інтервалу вказує на середню якість;

– значення RFC між кордонами довірчого та прогнозного інтервалу вказує на високу якість;

– значення RFC вище верхньої межі прогнозного інтервалу свідчить про низьку якість.

У дослідженні також наводиться приклад практичного застосування запропонованої методики для оцінювання якості відкритого програмного забезпечення, розробленого мовою Java, яка досі залишається однією з найпопулярніших мов програмування у світі.

Дослідники вивчають не тільки вплив метрик чи зв’язки між модулями, а і застосування різних інструментів, наприклад у дослідження [10] аналізується ефективність використання інструментів для неперервного оцінювання якості коду, таких як SonarCloud, для зменшення кількості дефектів у програмному забезпеченні.

Заявляється, що покращення якості коду через використання таких інструментів повинно приводити до меншої кількості дефектів і, відповідно, підвищення якості продукту з точки зору користувача. Проте, автори дослідження зазначають, що наявних доказів ефективності цих інструментів у реальних проєктах обмаль.

У роботі було досліджено п'ять відкритих програмних проєктів, які впровадили SonarCloud для оцінювання якості коду. Автори порівнювали частоту звітів про дефекти до і після впровадження SC. Для одного з проєктів було виявлено статистично значуще зниження кількості дефектів після впровадження. На рис. 1.2, можна побачити щотижневу кількість дефектів для кожного проєкту з часом, де помаранчеві лінії вказують на впровадження SonarCloud.

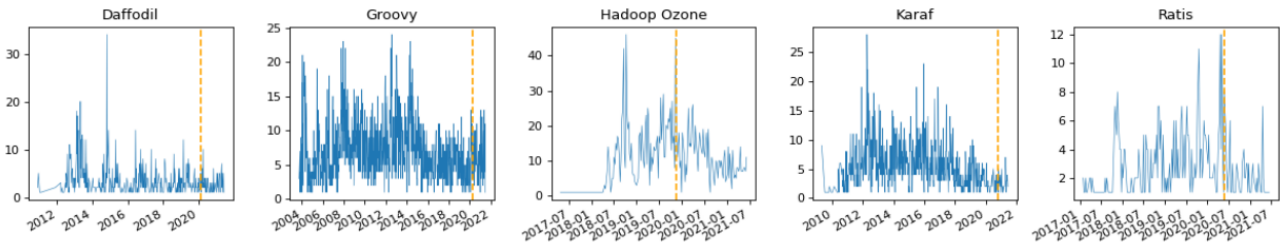


Рисунок 1.2 – Щотижнева кількість дефектів для кожного проєкту з часом з провадженням SonarCloud [10]

Однак подальший детальний аналіз показав, що це зниження, ймовірно, є випадковим і не пов'язане з використанням SC або дотриманням рекомендацій з покращення якості коду.

Загалом, результати дослідження не підтверджують гіпотезу про те, що впровадження інструментів CCQA, таких як SonarCloud, суттєво впливає на підвищення якості програмного продукту шляхом зменшення кількості дефектів. Це ставить під сумнів ефективність використання подібних інструментів як засобу забезпечення якості у реальних програмних проєктах.

Також, за допомогою метрик можна оцінити не тільки якість, а і стабільність системи. У дослідженні [11] увага націлена на стабільність об'єктно-орієнтованого дизайну, яка визначає, наскільки дизайн здатен протистояти змінам, що впливають на взаємозв'язки між класами. У випадку внесення змін до однієї з частин дизайну, ці зміни можуть викликати так званий ефект «ланцюгової реакції», що впливає на інші класи. Стабільність дизайну оцінюється логічною стабільністю (структурні зміни) та стабільністю продуктивності (поведінкові зміни).

Результати експерименту показали негативну кореляцію між метриками WMC, DIT, CBO, RFC та LCOM і логічною стабільністю класів. Це означає, що високі значення цих метрик, які вказують на більшу складність або сильну взаємозалежність між класами, знижують логічну стабільність дизайну. Однак для метрики NOC не було виявлено кореляції з логічною стабільністю.

Ці результати можуть бути використані для оптимізації структури дизайну, зменшення складності та підвищення стійкості до змін у майбутніх розробках.

Маючи таку велику кількість метрик та підходів для оцінювання ефективності, розміру чи якості ПЗ, звичайно постають питання про використання машинного навчання у задачі оцінювання, саме на цьому зосереджено дослідження [12].

Дослідження пропонує емпіричний підхід до оцінювання якості програмного забезпечення на основі характеристик якості, визначених стандартом ISO/IEC 25010. У дослідженні було застосовано алгоритми машинного навчання з підтримкою навчання для встановлення залежностей між метриками коду та показниками якості на основі історичних даних. Результатом стало створення моделі оцінювання якості програмного забезпечення, що базується на зібраних метриках коду з попередніх проєктів.

Для дослідження зібрано метрики коду та показників якості з 82 проєктів авіаційного вбудованого програмного забезпечення.

У межах цих кейсів було також проведено порівняння ефективності трьох алгоритмів машинного навчання для оцінювання якості:

– логістична регресія;
– алгоритм найближчих сусідів (k-Nearest Neighbor);
– штучна нейронна мережа (Back Propagation Neural Network).

Результати показали практичну здійсненність запропонованих метрик для оцінювання якості. Було також розроблено інструмент для управління даними проєкту та проведення оцінювання якості. Хоча отримані результати підтверджують ефективність методології, відгуки від інженерів-практиків вказують на необхідність подальшого вдосконалення для інтеграції підходу в реальні процеси розробки ПЗ.

Отже, дослідження підтвердило, що використання метрик і машинного навчання є перспективним підходом для оцінювання якості ПЗ, проте залишаються виклики, пов'язані з узгодженням методології з практичними потребами індустрії.

Попри наявність низки методів і метрик, що застосовуються для оцінювання якості, проблема полягає у відсутності єдиного підходу, який враховував би специфіку десктопних застосунків мовою програмування C#. Крім того, традиційні

методи часто зосереджуються на загальних аспектах, ігноруючи структурні характеристики та складність коду.

Існуючі підходи до оцінювання якості програмного забезпечення мають ряд обмежень, наприклад багато метрик зосереджуються лише на певних аспектах, не враховуючи повну картину. Метрики, такі як RFC, CBO та WMC, використовуються окремо, що обмежує їхню ефективність у комплексній оцінці.

Також, більшість моделей орієнтовані на веб- або мобільні застосунки і приділяється недостатня увага до особливостей архітектури та виконання десктопних програм на C#.

До проблеми також можна віднести використання регресійних моделей чи інших статистичних підходів, які часто потребують великих обчислювальних ресурсів та специфічної експертизи.

Для вирішення проблем необхідно:

– використовувати комбінований підхід, що включає аналіз коду, використання метрик і статистичних методів;

– побудувати математичну модель, яка враховує зв’язки між основними метриками ООП (RFC, CBO, WMC);

– розробити програмний продукт, який забезпечить автоматичний розрахунок цих метрик та інтеграцію моделі в процес розробки;

Для оцінювання якості будуть використовуватися такі метрики як RFC, CBO та WMC.

RFC відображає кількість методів, які викликає певний клас, що характеризує взаємозв’язки класу.

CBO відповідає за визначання залежності між класами, показуючи рівень зв’язності, а WMC обчислює кількість методів у класі, враховуючи їх складність.

Ці метрики забезпечують комплексний підхід до оцінювання структури та функціональності програмного забезпечення, дозволяючи враховувати як взаємозв’язки між класами, так і їхню індивідуальну складність.

Це дозволить створити універсальний інструмент для оцінювання якості, який відповідає сучасним вимогам і специфіці десктопних застосунків.

## 1.3 **Висновки до розділу 1**

Проведений аналіз підтвердив, що якість програмного забезпечення є критичним фактором для забезпечення успішної роботи десктопних застосунків. У сучасному світі високі вимоги до продуктивності, функціональності, надійності та зручності використання програмного забезпечення диктують необхідність використання ефективних методів і інструментів для оцінювання якості. Стандарт ISO/IEC 25010, як і ISO/IEC 250 забезпечує базу для класифікації ключових характеристик якості, таких як функціональна придатність, продуктивність, сумісність і безпека.

Загальні методики оцінювання часто не враховують специфіку об'єктно-орієнтованого програмування та особливості десктопних застосунків на мові C#. Це створює необхідність у нових підходах, що базуються на сучасних метриках, наприклад таких як RFC, CBO, WMC, які дозволяють оцінити структурну складність і взаємозалежність компонентів програми. Основними проблемами залишаються відсутність єдиного підходу до оцінювання та недостатня автоматизація процесу.

Кваліфікаційна робота спрямована на побудову математичної моделі, яка оцінює метрику RFC, залежну від CBO та WMC, і створення системи для автоматизованого аналізу якості десктопних застосунків.

# 2 ОГЛЯД ІСНУЮЧИХ МОДЕЛЕЙ ТА МЕТОДІВ ДЛЯ ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗБЕЧЕННЯ

Оцінювання якості програмного забезпечення є складним багатогранним процесом, який потребує застосування різних підходів залежно від специфікації, масштабів та мети проєкту. Основою будь-якого оцінювання є стандарти, метрики, методики та математичні моделі, які враховують як функціональні, так і нефункціональні характеристики програмного забезпечення.

## 2.1 Аналіз метрик та підходів оцінювання якості програмного забезпечення

Процес розробки програмного забезпечення є складним та багатовимірним. Будь-які недоліки в організації цього процесу можуть негативно позначитися на якості кінцевого продукту, знижуючи його надійність.

Для оцінювання таких параметрів розробники використовують систему показників, відомих як метрики. Метрики програмного забезпечення дозволяють об’єктивно вимірювати його якість, забезпечуючи інструмент для аналізу та удосконалення процесів створення ПЗ. Відповідно до стандартів, якість програмного забезпечення оцінюється за зовнішніми та внутрішніми характеристиками. Зовнішні характеристики визначають функціональні вимоги до продукту, а внутрішні пов’язані з архітектурою, кодом та процесами розробки.

**Емпіричні методи оцінювання.** Ці підходи базуються на зборі даних про продуктивність, стабільність і надійність програмного забезпечення у реальних умовах експлуатації. Вони включають аналіз відгуків користувачів, моніторинг системи у процесі її роботи та аналіз логів [13].

**Методи експертного оцінювання.** Ці підходи передбачають залучення експертів для оцінювання програмного забезпечення на основі їхнього досвіду. Експерти можуть використовувати чек-листи, стандарти та власні знання для визначення рівня відповідності програмного забезпечення вимогам. Після їх

оцінювання можливо створити систему на основі нечіткої логіки для оцінювання якості або характеристик ПЗ [14].

**Методи на основі машинного навчання та штучного інтелекту.** Останнім часом все більшого поширення набувають методи, які використовують алгоритми машинного навчання для прогнозування якості програмного забезпечення. Ці підходи базуються на аналізі великих обсягів даних, отриманих із попередніх проєктів, і дозволяють автоматизувати процес оцінювання. Нейронні мережі все частіше використовуються для класифікації програмного забезпечення за рівнем якості або для прогнозування ризиків [15].

**Методи автоматизованого аналізу коду.** Спеціальні інструменти, такі як SonarQube, Fortify, PMD, аналізують код на предмет вразливостей, відповідності стандартам кодування та структурної якості [16]. Ці методи дозволяють виявляти дефекти на ранніх стадіях розробки, забезпечуючи тим самим високу якість кінцевого продукту. Однак більшість таких інструментів є платними та потребують великих витрат на використання, наприклад ціна SonarQube стартує від 160 доларів на рік [17]. Водночас Fortify оголошує свою ціну після запиту, тобто підбирає індивідуально.

Також варто відзначити, що з попередніх досліджень помітно, що ефективність таких інструментів обмежена специфікою їх функціоналу.

Приклад оцінювання якості проєкту за допомогою SonarQube наведено на рис. 2.1.

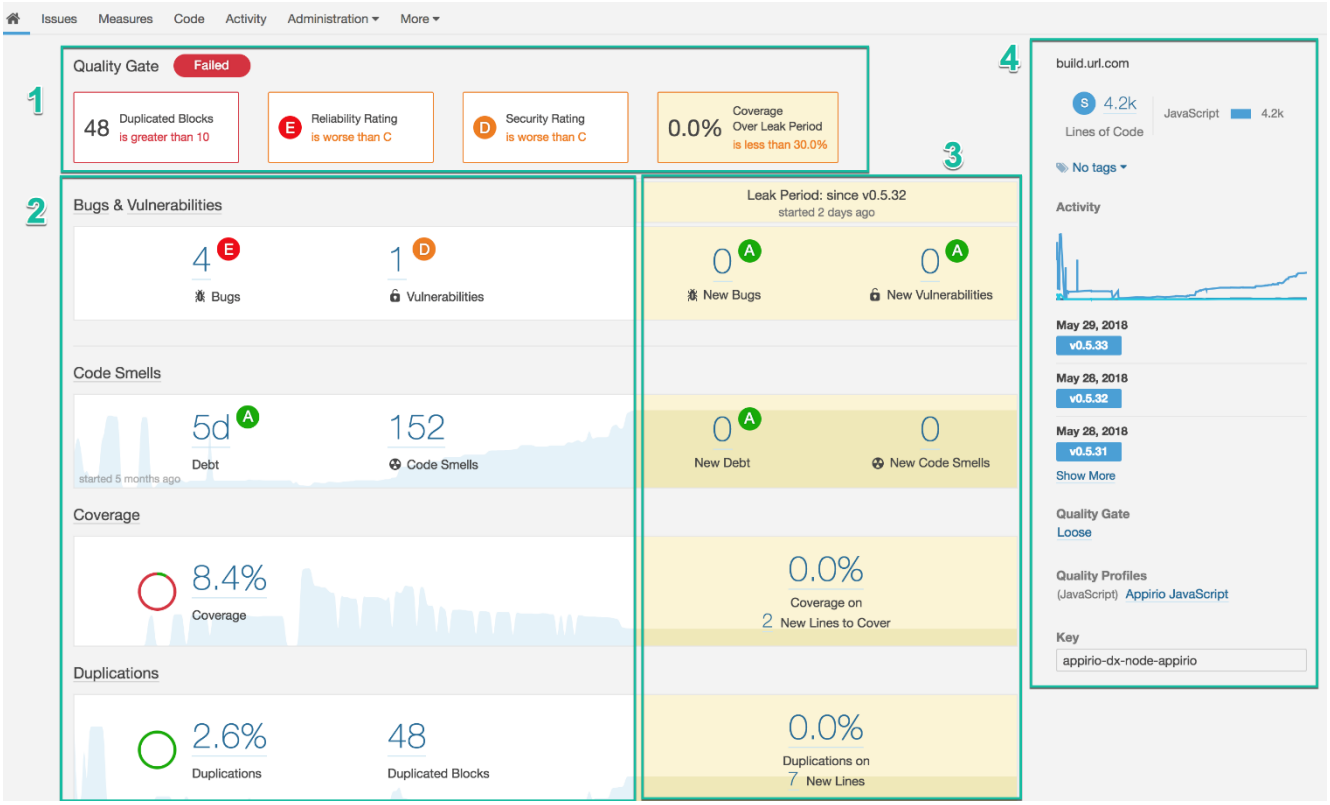


Рисунок 2.1 – Оцінювання якості за 4 параметрами в SonarQube [18]

Для кількісного оцінювання якості ПЗ визначаються відповідні атрибути, метрики, а також моделі оцінювання, які враховують час виконання компонентів, стабільність та ефективність їхньої роботи. Метрики, що застосовуються на етапах тестування або експлуатації, класифікуються як зовнішні, оскільки вони оцінюють параметри, які впливають на роботу продукту в реальних умовах.

ООП метрики програмного забезпечення є засобом для отримання кількісних показників його властивостей. Однією з важливих груп таких метрик є ті, що оцінюють якість розроблюваного продукту. Вони включають показники, які дозволяють розробникам зрозуміти загальний стан проєкту, виявити проблеми та визначити шляхи їх вирішення.

**Метрика щільності дефектів** є показником, який дозволяє оцінити кількість помилок у конкретному модулі відносно їхньої загальної кількості в проєкті. Цей показник дає змогу виявити проблемні ділянки системи та прийняти обґрунтовані рішення щодо їхньої оптимізації. Наприклад, якщо коефіцієнт щільності дефектів у модулі перевищує 0.2, це може свідчити про наявність проблем із вимогами,

недостатню кваліфікацію розробника чи технічну складність модуля. Використання цієї метрики забезпечує ефективне управління ризиками та поліпшення якості програмного забезпечення [19].

Щільність дефектів також можна вимірювати відносно розміру коду, наприклад, у рядках коду або функціональних точках. Вимірювання розміру за рядками коду має перевагу в простоті отримання даних за допомогою інструментів, інтегрованих у середовища розробки. Проте ця метрика залежить від мови програмування та інших технологічних особливостей.

**Метрики оцінювання ООП.** Крім основних метрик щільності дефектів, до аналізу якості ПЗ важливо включити ООП метрики оцінювання, які здатні працювати з архітектурними характеристиками коду [20, 21]:

**RFC** – визначає кількість методів, які можуть бути викликані об’єктом певного класу, враховуючи як власні методи класу, так і ті, що викликаються іншими класами. Чим вищий RFC, тим складніше стає тестування та підтримка класу, що може свідчити про недостатньо продуману архітектуру.

**CBO** – це показник, який вимірює зв’язність між класами. Високий рівень CBO вказує на сильну залежність між модулями, що ускладнює внесення змін до системи. Для досягнення якісного дизайну прагнуть до зменшення цього показника.

**WMC** – це метрика, що оцінює складність класу на основі кількості методів і їхньої ваги. Високий WMC може ускладнювати розуміння коду, підвищуючи ймовірність помилок та знижуючи загальну якість [22].

**DIT** – є метрикою, яка визначає глибину спадковості класу в ієрархії. Вона вимірює кількість класів, через які певний клас успадковується до найвищого рівня (зазвичай базового класу). Велике значення DIT свідчить про високе повторне використання коду, але може також ускладнити розуміння системи. Низьке ж значення DIT говорить про простішу структуру, але може свідчити про недостатнє використання переваг спадковості.

**NOC** – це метрика, яка визначає кількість класів, які безпосередньо успадковують певний клас. Вона показує, наскільки клас є базовим і скільки

підкласів використовують його функціональність. Високе значення NOC свідчить про те, що клас широко використовується як основа для інших класів, а низьке вказує на обмежене повторне використання

**LCOM** – це метрика, яка визначає рівень зв’язності методів у класі, тобто наскільки методи класу працюють над спільними змінними. Ця метрика має кілька переваг, зокрема, інформацію про кількість рядків коду в завершених проєктах чи модулях можна легко отримати за допомогою інтегрованих середовищ розробки (IDE) або спеціалізованих інструментів. Однак її недоліком є залежність від вибраної мови програмування та використовуваного середовища розробки.

Високе значення LCOM (низька зв’язність) означає, що клас може бути розділений на кілька менших класів для покращення модульності. У той час, як низьке значення (висока зв’язність) свідчить про те, що клас добре організований і фокусується на виконанні однієї задачі.

Підсумовуючи, високі значення метрик, таких як RFC, CBO та WMC, вказують на проблеми у підтримці, тестуванні та модифікації системи через її складність або сильну залежність між класами. DIT та NOC допомагають аналізувати використання спадковості, що впливає на повторне використання коду та його структурну ясність. LCOM показує рівень зв'язності методів у класі: низька зв'язність сигналізує про необхідність рефакторингу для покращення модульності.

Використання цих метрик допомагає розробникам підвищити якість дизайну програмного забезпечення, спрощуючи його підтримку та зменшуючи ймовірність помилок.

Для того, щоб обрати найкращий підхід для вирішення поставленої задачі, потрібно проаналізувати та виокремити конкретні переваги та недоліки кожного підходу. Спираючись на теоретичні знання та огляд останніх досліджень, було створено табл. 2.1, в якій описуються переваги, недоліки та найкраще застосування метрик та методів, таких як емпіричні методи оцінювання, методи експертного оцінювання, методи на основі ШІ, методи автоматизованого аналізу та ООП метрики.

Таблиця 2.1 – Порівняльна характеристика методів та метрик

| Метод/метрика | Переваги | Недоліки | Застосування |
|---|---|---|---|
| Емпіричні методи оцінювання | Реальна оцінка якості з даних. | Потребує часу для збору достатньої кількості даних. | Оцінювання стабільності в реальних умовах експлуатації. |
| Методи експертного оцінювання | Можливість оцінювання без реальних даних. | Суб'єктивність оцінок; складна реалізація систем (багато правил). | Швидкий аналіз на основі експертних знань. |
| Методи на основі машинного навчання та ШІ | Висока точність прогнозування; автоматизація оцінювання на основі історичних даних. | Потребує великого обсягу даних; високі вимоги до обчислювальних ресурсів. | Прогнозування ризиків і класифікація якості. |
| Методи автоматизованого аналізу коду | Виявлення дефектів на ранніх стадіях; автоматизація перевірок. | Залежність від інструментів; відсутність гнучкості. | Автоматична перевірка на відповідність стандартам. |
| ООП метрики | Чіткі кількісні показники для аналізу структури та якості. | Можливі складнощі у виборі оптимальних метрик та вибору математичної моделі. | Аналіз архітектури та коду на основі об'єктно-орієнтованих принципів. |

ООП метрики пропонують універсальний підхід до оцінювання архітектури

та коду на основі об'єктно-орієнтованих принципів. Ці метрики забезпечують кількісну оцінку складності, зв'язності та взаємодії компонентів системи, що є ключовим для забезпечення високої якості програмного забезпечення.

Це є найкращим підходом для оцінювання якості програмного забезпечення для нашої задачі, оскільки ООП метрики дозволяють аналізувати архітектуру, забезпечують об’єктивні показники і допомагають знаходити проблеми ще на етапі розробки.

Метрики якості програмного забезпечення надають розробникам можливість об’єктивно оцінити стан проєкту на будь-якому етапі життєвого циклу.

Обираючи серед існуючих метрик ООП, вибір падає на метрики RFC, CBO та WMC, ці метрики обрані через їхню здатність надавати кількісну оцінку ключових характеристик ООП-дизайну: зв’язності, складності та цілісності.

У той же час як DIT, NOC та LCOM дають інші корисні показники, наприклад рівень використання класу як базового.

Метрики RFC, CBO та WMC дають змогу оцінити якість архітектури, визначити слабкі місця та знайти способи їхнього покращення. Водночас використання метрики щільності дефектів допомагає ефективно управляти тестуванням і ризиками. Разом ці показники дозволяють створювати програмні продукти високої якості, які відповідають сучасним стандартам.

## 2.2 **Аналіз існуючих математичних моделей для оцінювання якості ПЗ**

Математичні моделі є основою для кількісного оцінювання якості програмного забезпечення. Вони дозволяють формалізувати процес оцінювання, враховуючи різноманітні аспекти, такі як структурна якість, продуктивність, надійність та інші характеристики.

**Регресійні моделі** широко використовуються для прогнозування залежностей між метриками програмного забезпечення та показниками його якості. Найбільш популярними є лінійні та нелінійні регресійні моделі. Вони описують залежність між незалежними змінними та залежною змінною.

Формула лінійної регресії 2.1 [23, 24]:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \varepsilon\,, \tag{2.1}$$

де Y – залежна змінна;

$X_1, X_2 \ldots, X_n$ – незалежні змінні;

$\beta_1, \beta_2 \ldots, \beta_n$ – коефіцієнти регресії;

$\varepsilon$ – похибка моделі.

Зазвичай при створенні лінійних регресійних моделей потрібно дотримуватись певних умов. Зокрема, залишки або залежна змінна мають відповідати нормальному розподілу.

Для складніших залежностей використовуються нелінійні регресійні моделі.

Для побудови нелінійних регресійних моделей можна використовувати нормалізуючі перетворення, до яких відноситься перетворення Бокса-Кокса, яке покращує точність прогнозування, формула 2.2 [25]:

$$Z_j = x(\lambda_j) = \begin{cases} \left(X_j^{\lambda_j} - 1\right)/\lambda_j, & \lambda_j \neq 0 \\ \ln(X_j), & \lambda_j = 0 \end{cases}, \tag{2.2}$$

де $Z_j$– це змінна Гауса;

$\lambda_j$ – параметр перетворення Box-Cox;

$j$ – метрика (CBO, RFC, WMC).

**Байєсові методи** дозволяють враховувати попередню інформацію про якість програмного забезпечення та комбінувати її з новими даними. Основна формула байєсового оновлення 2.3 [26]:

$$P(H \mid D) = \frac{P(D \mid H) * P(H)}{P(D)}, \tag{2.3}$$

де $P(H \mid D)$ – ймовірність гіпотези H (наприклад, що продукт має високу якість) за умови отриманих даних D;

$P(D \mid H)$ – ймовірність даних D за умови гіпотези H;

$P(H)$ – апріорна ймовірність гіпотези;

$P(D)$– загальна ймовірність даних.

Байєсові підходи часто застосовуються для оцінювання ризиків, пов’язаних із впровадженням змін у код.

Різні математичні моделі використовуються залежно від контексту. Регресійні моделі підходять для прогнозування метрик та аналізу залежностей. Ймовірнісні моделі корисні для оцінювання надійності та ризиків. Метрики ООП забезпечують структурний аналіз, а байєсові моделі дозволяють враховувати попередній досвід і прогнози. Використання комбінованого підходу, який інтегрує кілька типів моделей, є найбільш ефективним для забезпечення комплексного оцінювання якості програмного забезпечення.

Для оцінювання якості десктопних застосунків за основні метрик обрано CBO, RFC та WMC.

Так як не було знайдено моделей оцінювання метрики RFC десктоп застосунків на C#, було прийнято рішення побудувати двофакторну нелінійну регресійну модель.

## 2.3 **Висновки до розділу 2**

Огляд існуючих моделей і методів для оцінювання якості програмного забезпечення продемонстрував, що сучасні підходи включають широкий спектр методик, таких як емпіричні, статистичні, автоматизовані, а також засновані на штучному інтелекті. Метрики об'єктно-орієнтованого програмування, включаючи RFC, CBO, WMC, залишаються основою для структурного аналізу програмного забезпечення. Виявлено, що використання комбінованих підходів, які інтегрують різні методики, дозволяє отримати більш точну і комплексну оцінку якості.

Серед моделей для оцінювання якості ПЗ, виділяються регресійні, але використання лінійних моделей потребує виконання певних умов, наприклад нормального розподілу помилок, що справедливо лише в окремих випадках, через що і постає задача в створенні нелінійної регресійної моделі.

Підсумовуючи, визначено, що наступним кроком у дослідженні буде створення математичної моделі для оцінювання метрики RFC десктоп застосунків на C# залежної від метрик CBO та WMC.

# 3 МАТЕМАТИЧНА МОДЕЛЬ ДЛЯ ОЦІНЮВАННЯ ЯКОСТІ МЕТРИКИ RFC ДЕСКТОП ЗАСТОСУНКІВ НА C#

Аналіз наявних методів та моделей оцінювання якості програмного забезпечення не виявив таких, що могли б бути застосовані для оцінювання якості C# застосунків для десктоп.

## 3.1 **Збір метрик з десктоп проєктів на C#**

Збір даних з застосунків на C# виконується за допомогою інструмента NDepend.

NDepend є потужним статичним аналізатором для .NET проєктів. Його потрібно завантажити з офіційного сайту та інтегрувати у вашу IDE, наприклад, Visual Studio. Інструмент працює з існуючими проєктами .NET Framework або .NET Core/5+ [27].

NDepend дозволяє виконувати детальний статичний аналіз великих проєктів на C#, збираючи десятки метрик для класів, методів та збірок. Зокрема, метрики CBO, RFC та WMC допомагають виявити структурні недоліки коду.

За допомогою Dependency Matrix, Dependency Graph та Metrics View, NDepend забезпечує наочний спосіб аналізу взаємодій між класами та збірками. Візуалізація допомагає зрозуміти структуру проєкту та виявити «вузькі місця» з високою зв’язністю чи надмірною складністю.

Завдяки CQLinq можна автоматизувати збір конкретних метрик, створюючи запити для пошуку класів з високим CBO, RFC або WMC.

Також інтеграція з Visual Studio допомагає швидко виконувати аналіз проєкту.

Результат аналізу застосунку наведений на рис. 3.1.

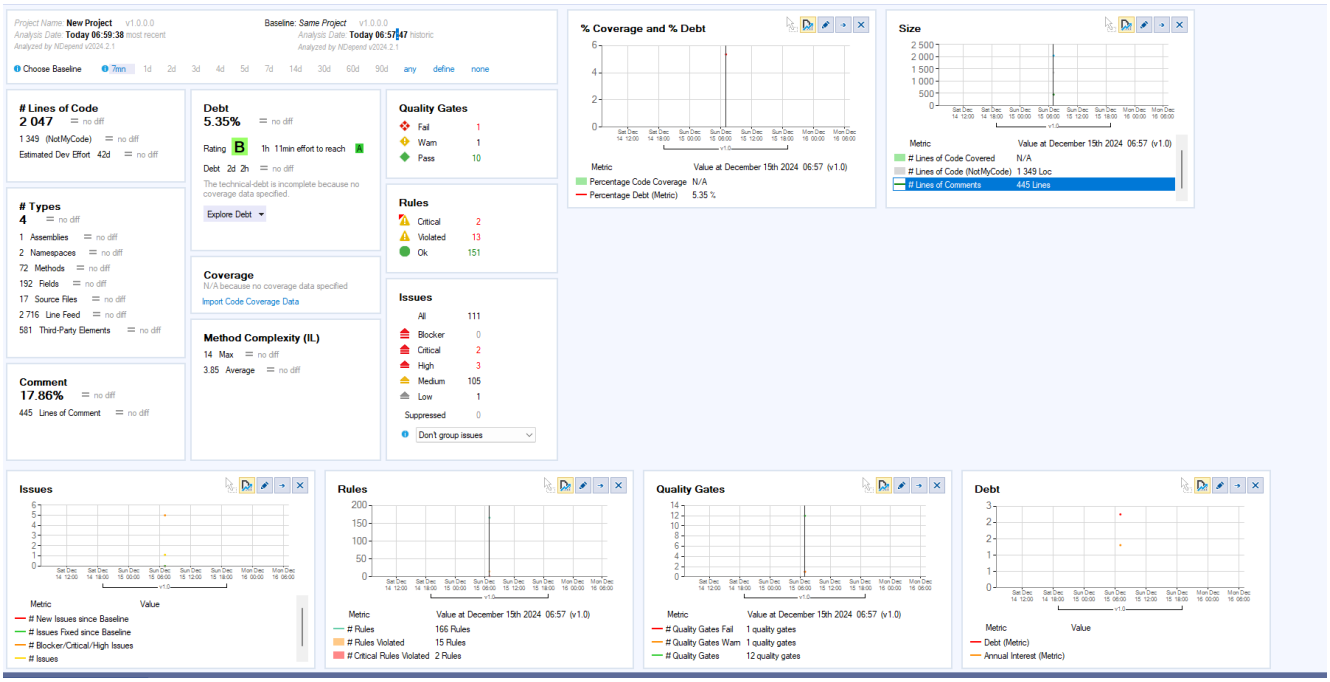


Рисунок 3.1 – Аналіз застосунку на C# за допомогою NDepend

Спочатку ми інтегруємо NDepend у Visual Studio після чого завантажуємо застосунки з GitHub.

Запускаємо аналіз і за допомогою CQLinq шукаємо необхідні нам метрики, будемо обирати лише точкові оцінки їх середнього значення по проекту. Збір значення CBO наведено на рис. 3.2.

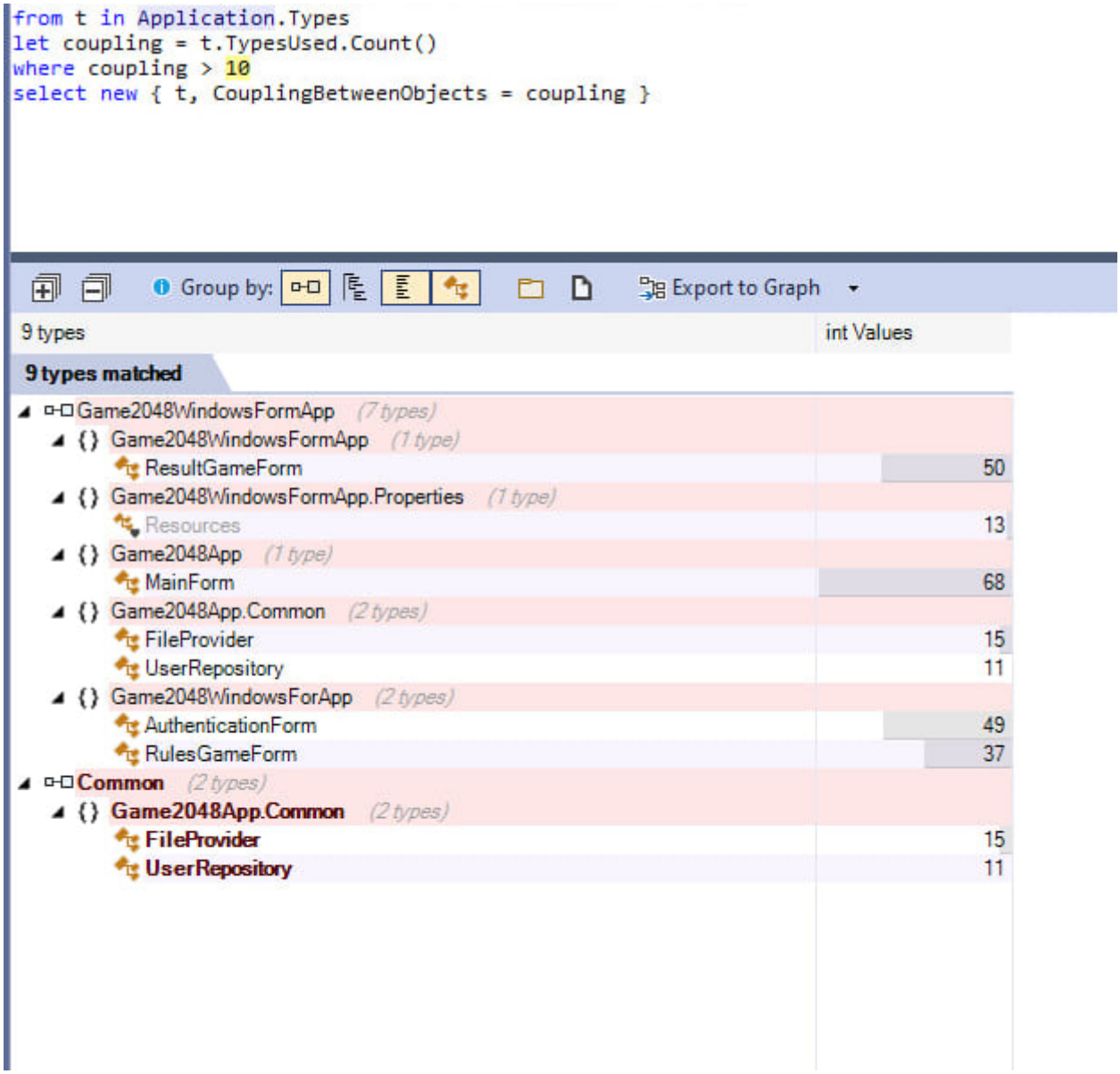


Рисунок 3.2 – Збір метрики CBO з застосунку на C#

Як було зазначено, CBO це метрика, яка визначає кількість залежностей одного класу від інших. Чим більше класів використовуються в одному класі, тим вищим є значення CBO. Високе значення показує сильну залежність від інших класів, що негативно впливає на гнучкість, тестування та підтримку програмного забезпечення.

Приклад збору метрики RFC та WMC наведено на рис. 3.3 та 3.4 відповідно.

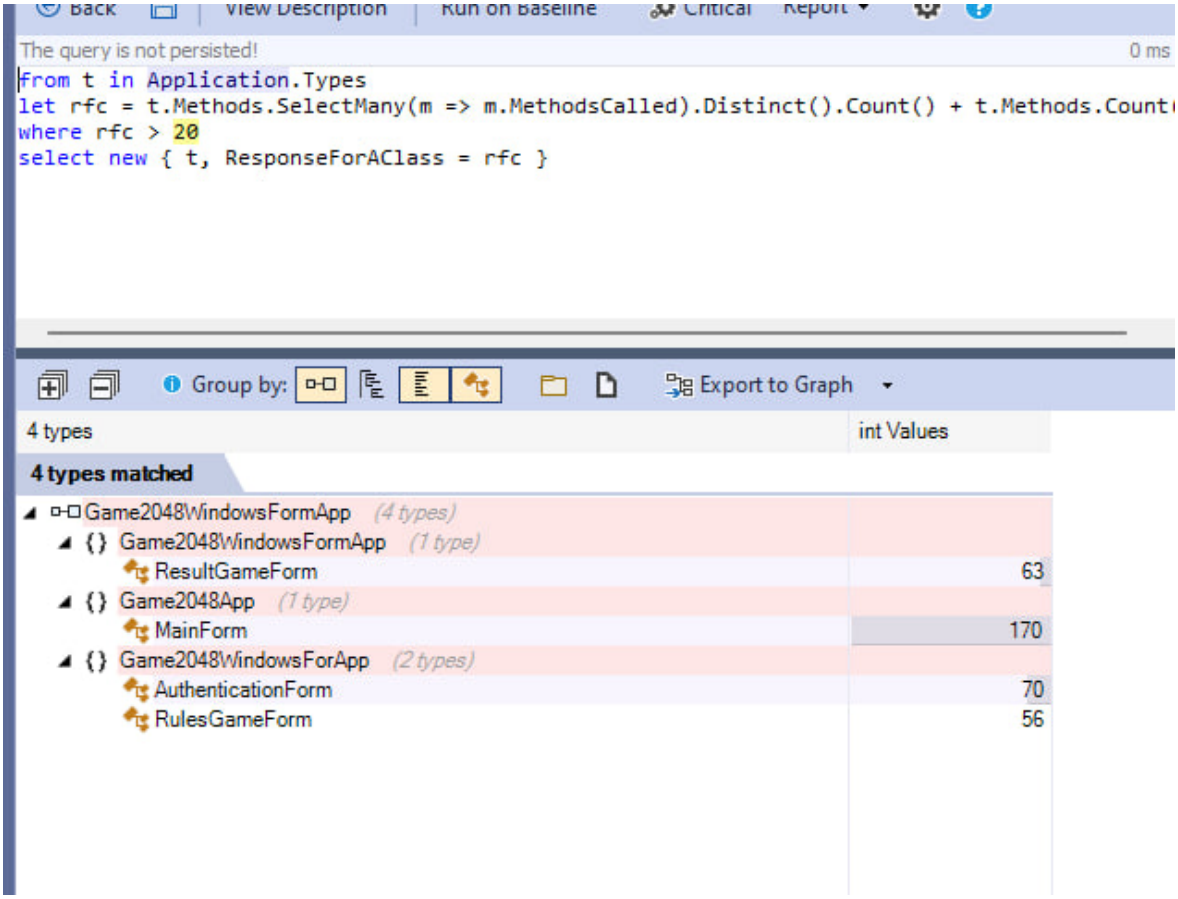


Рисунок 3.3 – Збір метрики RFC з застосунку на C#

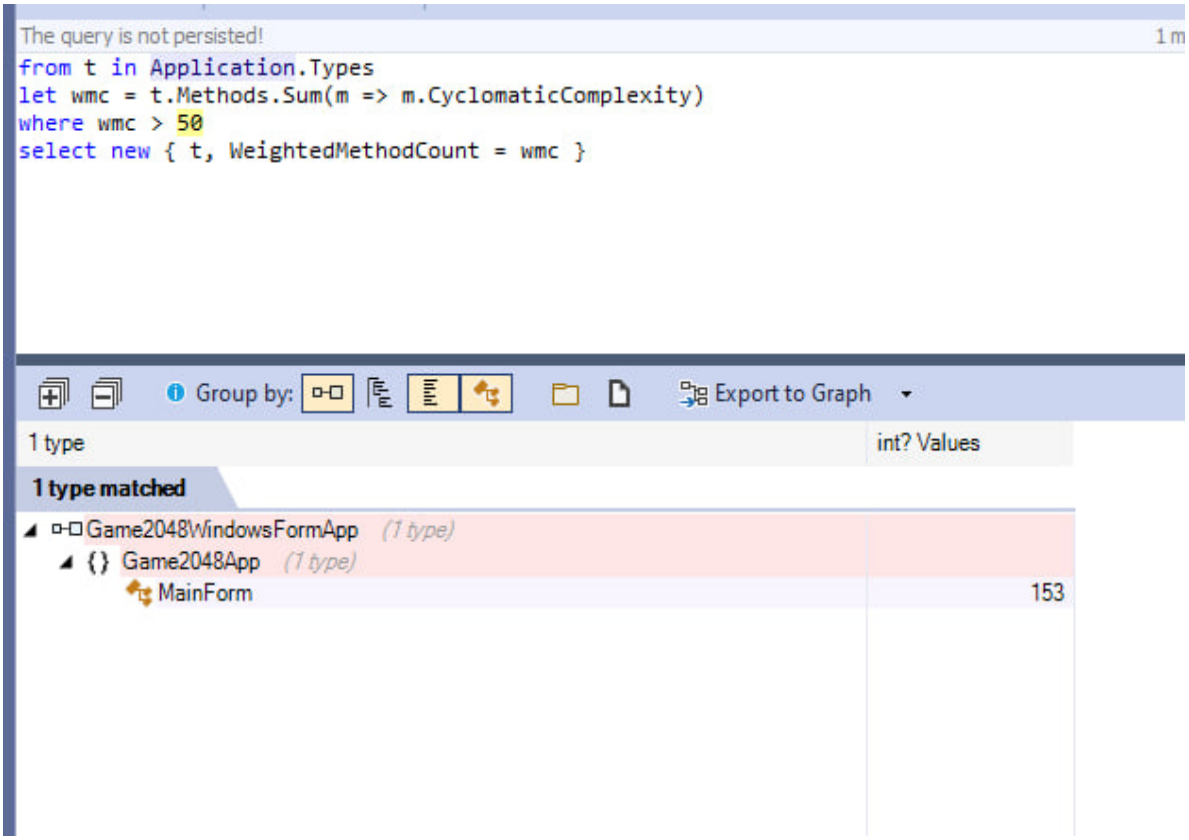


Рисунок 3.4 – Збір метрики WMC з застосунку на C#

Варто зауважити, що збір даних проводився з різних десктоп застосунків, тобто як великих по типу ігрових, так і середніх, як от менеджер часу.

Отже, такими методами вдалося зібрати метрики з 56 застосунків на C# (включно з різними версіями та релізами), які представлені в таблиці 3.1.

Таблиця 3.1 – Метрики CBO, RFC та WMC проєктів десктоп застосунків на C#

| **№** | **CBO** | **RFC** | **WMC** |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 1 | 29,891 | 39,982 | 17,632 |
| 2 | 7,706 | 5,435 | 12,138 |
| 3 | 23,247 | 15,839 | 27,652 |
| 4 | 12,306 | 18,342 | 22,128 |
| 5 | 18,947 | 25,634 | 31,206 |
| 6 | 26,532 | 34,783 | 16,844 |
| 7 | 5,891 | 12,476 | 10,012 |
| 8 | 13,872 | 9,361 | 14,572 |
| 9 | 20,354 | 30,877 | 21,314 |
| 10 | 11,563 | 19,905 | 28,437 |
| 11 | 15,344 | 23,123 | 33,014 |
| 12 | 35,214 | 49,876 | 42,108 |
| 13 | 9,265 | 6,145 | 10,357 |
| 14 | 14,721 | 10,543 | 18,991 |
| 15 | 19,387 | 32,141 | 29,451 |
| 16 | 7,536 | 9,445 | 12,678 |
| 17 | 10,349 | 8,722 | 11,305 |
| 18 | 22,138 | 14,871 | 25,348 |
| 19 | 17,655 | 28,902 | 36,211 |
| 20 | 5,915 | 20,137 | 24,556 |

Продовження табл. 3.1

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 21 | 38,297 | 45,368 | 40,623 |
| 22 | 11,084 | 7,652 | 10,995 |
| 23 | 27,451 | 35,661 | 29,714 |
| 24 | 5,992 | 5,334 | 10,452 |
| 25 | 13,275 | 19,452 | 20,803 |
| 26 | 36,408 | 42,755 | 44,682 |
| 27 | 6,259 | 9,601 | 13,213 |
| 28 | 18,325 | 15,021 | 27,442 |
| 29 | 25,732 | 33,095 | 38,221 |
| 30 | 9,888 | 5,891 | 14,792 |
| 31 | 39,365 | 46,201 | 58,334 |
| 32 | 20,573 | 29,376 | 16,009 |
| 33 | 16,455 | 21,098 | 27,535 |
| 34 | 8,007 | 8,559 | 14,662 |
| 35 | 31,589 | 40,102 | 34,417 |
| 36 | 21,302 | 25,778 | 23,997 |
| 37 | 10,891 | 15,341 | 12,537 |
| 38 | 14,639 | 10,222 | 17,291 |
| 39 | 26,201 | 14,336 | 22,449 |
| 40 | 5,205 | 9,121 | 13,777 |
| 41 | 9,467 | 18,334 | 19,981 |
| 42 | 7,205 | 11,534 | 15,925 |
| 43 | 34,388 | 48,753 | 39,559 |
| 44 | 6,108 | 5,962 | 10,212 |
| 45 | 12,446 | 20,201 | 25,559 |
| 46 | 17,559 | 16,994 | 27,108 |
| 47 | 22,405 | 10,448 | 15,003 |

Продовження табл. 3.1

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 48 | 8,552 | 14,267 | 19,902 |
| 49 | 11,782 | 7,604 | 12,552 |
| 50 | 36,847 | 42,558 | 45,979 |
| 51 | 24,109 | 12,346 | 28,403 |
| 52 | 28,013 | 17,284 | 34,278 |
| 53 | 5,873 | 5,504 | 10,908 |
| 54 | 18,201 | 6,223 | 12,955 |
| 55 | 7,683 | 9,312 | 11,521 |
| 56 | 37,205 | 22,122 | 15,894 |

Після збору даних, можна приступати до створення математичної моделі оцінювання метрики RFC.

### 3.2 **Нормалізація даних метрик CBO, RFC та WMC**

Для того, щоб створити математичну модель, яка дозволить оцінювати метрику RFC виконаємо нормалізуючі перетворення вхідних даних використовуючи тривимірне перетворення Box-Cox, яке можна представити за формулою 2.2, щоб зменшити вплив великих значень.

Популярність даного перетворення зумовлена тим, що воно:

– підвищує стійкість моделі до змінних з різними порядками величин;

– покращує точність оцінок, особливо за наявності нерівномірного розподілу даних;

– сприяє більш зрозумілій інтерпретації результатів.

Також, необхідно дослідити фактори на предмет мультиколінеарності [28]. Її присутність вказує на те, що незалежні змінні мають взаємозв'язок у рамках множинної регресійної моделі або демонструють високий рівень кореляції.

Наявність мультиколінеарності оцінюється за допомогою коефіцієнтів варіаційного інфляційного фактору (VIFs), які використовуються для аналізу

майбутніх предикторів (факторів) у моделі множинної лінійної регресії. Значення VIF більше ніж 10 часто сигналізує про наявність мультиколінеарності. Якщо значення VIF знаходяться у діапазоні від 1 до 5, то мультиколінеарність відсутня.

Для метрик CBO та WMC з табл. 3.1, значення коефіцієнтів дорівнюють 4,88 та 4.74 відповідно, що вказує на помірний рівень мультиколінеарності, це може вплинути на результати регресійного аналізу.

Додамо регуляризації, щоб зменшити вплив мультиколінеарності на модель. Використаємо Ridge Regression так як всі змінні є важливими. Він зменшує коефіцієнти змінних, пов'язаних з мультиколінеарністю, не видаляючи змінні з моделі. Формулу цієї регуляризації можна представити як 3.1 [29].

$$\mathrm{L}oss = RSS + \lambda \sum_{i=0}^{n} \beta_i^2 , \qquad (3.1)$$

де *Loss* – функція втрат;

*RSS* – помилка прогнозування (різниця між фактичними та прогнозованими значеннями);

$\lambda \sum_{i=0}^{n} \beta_i^2$ – регуляризаційний штраф.

Після застосування Ridge-регуляризації отримано такі коефіцієнти для предикторів: CBO: 1.8893, WMC: 0.3530.

Далі виконаємо нормалізацію та позбудемося викидів, для цього застосуємо квадрат відстані Махаланобіса, який може бути представлений формулою 3.2 [30]:

$$d_i^2 = (Z_i - \bar{Z})^T S_N^{-1} (Z_i - \bar{Z}) , \qquad (3.2)$$

де $Z_i$ – вектор значень (CBO, RFC, WMC) для одного рядка,;

$\bar{Z}$ – вектор вибіркових середніх;

$S_N^{-1}$– обернена коваріаційна матриця.

При цьому, матриця коваріаційної вибірки визначається наступним чином:

$$S_N = \frac{1}{N}\sum_{i=1}^{N}(Z_i - \bar{Z})(Z_i - \bar{Z})^T,$$

де $\bar{Z}$– середні арифметичні відповідних величин;

$Z_i$ – гаусівський випадковий вектор $Z = \{Z_1, Z_2, \dots, Z_m\}^T$ для кожної багатовимірної точки даних *i*.

Після відсіювання викидів, залишилось 55 з 56 рядків, тобто залишилися лише ті дані, які знаходяться в межах прийнятних відстаней.

Після цього у нас буде новий набір нормалізованих даних, з якими можна побудувати нелінійну регресію. Цей набір вхідних даних наведений в табл. 3.2.

Таблиця 3.2 – Нормалізовані метрики за допомогою перетворення Box-Cox

| № | CBO | RFC | WMC |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 1 | 4,188 | 3,662 | 2,115 |
| 2 | 2,312 | 1,687 | 1,911 |
| 3 | 3,817 | 2,748 | 2,34 |
| 4 | 2,926 | 2,893 | 2,232 |
| 5 | 3,523 | 3,223 | 2,397 |
| 6 | 4,011 | 3,525 | 2,091 |
| 7 | 1,974 | 2,511 | 1,799 |
| 8 | 3,088 | 2,227 | 2,013 |
| 9 | 3,625 | 3,407 | 2,213 |
| 10 | 2,842 | 2,974 | 2,353 |
| 11 | 3,227 | 3,121 | 2,422 |
| 12 | 4,436 | 3,88 | 2,53 |
| 13 | 2,549 | 1,809 | 1,819 |
| 14 | 3,17 | 2,345 | 2,154 |
| 15 | 3,556 | 3,447 | 2,37 |
| 16 | 2,283 | 2,236 | 1,936 |

Продовження табл. 3.2

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 17 | 2,695 | 2,157 | 1,87 |
| 18 | 3,746 | 2,685 | 2,298 |
| 19 | 3,423 | 3,342 | 2,464 |
| 20 | 4,565 | 3,787 | 2,515 |
| 21 | 2,786 | 2,027 | 1,854 |
| 22 | 4,061 | 3,549 | 2,374 |
| 23 | 1,997 | 1,669 | 1,824 |
| 24 | 3,028 | 2,951 | 2,201 |
| 25 | 4,487 | 3,728 | 2,556 |
| 26 | 2,05 | 2,252 | 1,959 |
| 27 | 3,476 | 2,695 | 2,336 |
| 28 | 3,965 | 3,476 | 2,488 |
| 29 | 2,635 | 1,767 | 2,021 |
| 30 | 4,608 | 3,804 | 2,666 |
| 31 | 3,64 | 3,358 | 2,064 |
| 32 | 3,324 | 3,031 | 2,338 |
| 33 | 2,361 | 2,138 | 2,016 |
| 34 | 4,271 | 3,665 | 2,441 |
| 35 | 3,69 | 3,229 | 2,272 |
| 36 | 2,762 | 2,716 | 1,929 |
| 37 | 3,162 | 2,314 | 2,126 |
| 38 | 3,992 | 2,649 | 2,239 |
| 39 | 1,823 | 2,199 | 1,982 |
| 40 | 2,578 | 2,892 | 2,18 |
| 41 | 2,226 | 2,434 | 2,061 |
| 42 | 4,4 | 3,857 | 2,503 |
| 43 | 2,019 | 1,779 | 1,81 |

Продовження табл. 3.2

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 44 | 2,941 | 2,988 | 2,302 |
| 45 | 3,415 | 2,817 | 2,331 |
| 46 | 3,763 | 2,336 | 2,029 |
| 47 | 2,445 | 2,644 | 2,178 |
| 48 | 2,867 | 2,021 | 1,93 |
| 49 | 4,506 | 3,724 | 2,568 |
| 50 | 3,87 | 2,501 | 2,353 |
| 51 | 4,091 | 2,834 | 2,44 |
| 52 | 1,971 | 1,7 | 1,849 |
| 53 | 3,466 | 1,821 | 1,948 |
| 54 | 2,308 | 2,221 | 1,881 |
| 55 | 4,521 | 3,077 | 2,06 |

Для побудови двофакторної нелінійної регресійної моделі розглянемо більш складну функціональну залежність, яка враховує нелінійні зв'язки між змінними. Залежною буде RFC, а незалежними CBO та WMC. Математична модель буде мати вигляд формули 3.3 [31]:

$$Y = [\,\hat{\lambda}_Y(\hat{Z}_Y + \varepsilon) + 1\,]^{1/\hat{\lambda}_Y}\ , \qquad (3.3)$$

де Y – цільова змінна;

$\hat{\lambda}_Y$ – параметр перетворень;

ε – випадкова величина (похибка), приймаємо нульову гіпотезу щодо нормальності розподілу залишків ε на рівні значимості 0,05;

$\hat{Z}_Y$ – результат лінійної моделі, яка має вигляд:

$$\hat{Z}_Y = \hat{b}_0 + \hat{b}_1 Z_1 + \hat{b}_2 Z_2$$

Значення параметрів $\hat{b}_0 = -3{,}2602$, $\hat{b}_1 = 0{,}2481$, $\hat{b}_2 = 2{,}5573$.

Параметр перетворень для метрик дорівнює $\hat{\lambda}_{RFC} = 0{,}0878;\ \hat{\lambda}_{CBO} = 0{,}1191;$ $\hat{\lambda}_{WMC} = -0{,}2246.$

Довірчий інтервал лінійної регресії визначається за формулою 3.4 [31]:

$$\hat{Z}_Y \pm\ t_{a/2,v} S_{Z_Y} \left\{\frac{1}{N} + (z_X^+)^T S_Z^{-1} (z_X^+)\right\}^{1/2}, \tag{3.4}$$

а інтервал передбачення лінійної регресії за формулою 3.5 [31]:

$$\hat{Z}_Y \pm\ t_{a/2,v} S_{Z_Y} \left\{1 + \frac{1}{N} + (z_X^+)^T S_Z^{-1} (z_X^+)\right\}^{1/2}, \tag{3.5}$$

де $\hat{Z}_Y$ – є результатом передбачення за лінійним регресійним рівнянням;

$z_X^+$ – це вектор з компонентами $Z_{1_i} - \bar{Z}_1, Z_{2_i} - \bar{Z}_2$ для *i* рядка;

$\bar{Z}_j = \frac{1}{N}\sum_{i=1}^{N} Z_{j_i}, j = 1{,}2;$

$S_{Z_Y}^2 = \frac{1}{v}\sum_{i=1}^{N}(Z_{Y_i} - \hat{Z}_{Y_i})^2;$

$v = N - k - 1;$

$S_z$ – це матриця 2x2, яка виглядає наступним чином:

$$S_z = \begin{pmatrix} S_{Z_1 Z_1} & S_{Z_1 Z_2} \\ S_{Z_1 Z_2} & S_{Z_2 Z_2} \end{pmatrix},$$

де $S_{Z_q Z_r} = \left[Z_{q_i} - \bar{Z}_q\right]\left[Z_{r_i} - \bar{Z}_r\right], q, r = \overline{1, k}.$

Інтервали нелінійної регресії отримуємо через зворотнє перетворення.

Для оцінювання якості отриманої двофакторної нелінійної регресійної моделі будемо використовувати критерії $R^2$, PRED(0,25) та MMRE.

$R^2$ – це коефіцієнт детермінації. Він вимірює частку варіації залежної змінної, яка пояснюється незалежною змінною за допомогою моделі. Значення $R^2$ варіюється від 0 до 1, де 1 означає, що модель ідеально пояснює залежність, а 0, що не пояснює зовсім. $R^2$ обчислюється за формулою 3.6 [32].

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y})^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \quad (3.6)$$

де $y_i$ – емпіричне значення $y$;

$\hat{y}$ – розрахункове значення y, середнє значення випадкової величини $\bar{y}$ можна представити як:

$$\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i ,$$

MMRE – це середнє відхилення передбачень моделі від реальних значень у відсотках. Низьке значення MMRE (менше 0,25) свідчить про непогану точність моделі. MMRE має наступний вигляд (формула 3.7) [33]:

$$\text{MMRE} = \frac{1}{N}\sum_{i=1}^{N} MRE_i , \quad (3.7)$$

де $N$ – загальна кількість значень у вибірці;

PRED(0,25) показує відсоток передбачень моделі, відхилення яких від реальних значень менше 25%. Високе значення (близьке до 1, тобто 100%) означає, що модель стабільно виконує точні передбачення. PRED(0,25) можна представити наступною формулою 3.8 [34].

$$\text{PRED}(0{,}25) = \frac{k}{N}, \quad (3.8)$$

де $k$ – кількість спостережень, для яких відносна помилка не перевищує 0.25;

$N$ – загальна кількість спостережень;

Розрахувавши ці метрики для цієї моделі отримуємо наступні результати:

– $R^2$ = 0,706, тобто 70,6%. Модель пояснює 70,6% варіації залежної змінної RFC що є високим результатом;

– MMRE = 0,269, що вказує на високу точність;

– PRED(0,25) = 0,642, більше половини передбачення мають похибку менше 25%.

Порівняємо нашу математичну модель оцінювання метрики RFC зі схожою моделлю для оцінювання метрики RFC для застосунків з відкритим кодом на мові програмування Java, за формулою 3.9 [35]:

$$Y = 10^{\varepsilon - 0{,}05477}\, X_1^{0{,}67226} X_2^{0{,}44154}, \qquad (3.9)$$

Використаємо в якості вхідних даних наші та зробимо порівняльний аналіз результатів у табл. 3.3.

Таблиця 3.3 – Порівняльний результат двофакторних моделей

| **Критерій** | $Y = 10^{\varepsilon - 0{,}05477}\, X_1^{0{,}67226} X_2^{0{,}44154}$ | $Y = [\,\hat{\lambda}_Y(\hat{Z}_Y + \varepsilon) + 1\,]^{1/\hat{\lambda}_Y}$ |
|---|---|---|
| $R^2$ | 0,192 | 0,706 |
| MMRE | 0,682 | 0,269 |
| PRED(0,25) | 0,367 | 0,642 |

Як видно з таблиці, наша модель суттєво краще враховує природу зв’язку між CBO, WMC та вихідною змінною RFC, вона надає значно вищий коефіцієнт детермінації (0,706 проти 0,192 у першої) та помітно знижує відносні похибки (MMRE зменшується майже в 3 рази, а частка успішних передбачень PRED(0,25) зростає майже в 2 рази). Це свідчить, що включення додаткового зсуву та застосування регуляризації дає моделі гнучкіші інструменти для корекції відмінностей між фактичною та прогнозованою величинами. Перша модель, що базується суто на десятковому логарифмі, виявилася надто обмеженою для адекватного опису залежності в цих даних, тому не змогла забезпечити достатнє пояснення зміни RFC.

Середня відносна помилка (MMRE) побудованої моделі становить 0,269

порівняно з 0,682 у першої моделі. Це свідчить про вищу точність другої моделі в прогнозуванні.

Також варто звернути увагу на показник PRED(0,25), який у побудованій моделі дорівнює 0,642, а у першій моделі – 0,367. Це означає, що друга модель значно краще відповідає критерію точності в межах похибки 0,25.

Таким чином, запропонована модель на основі нормалізації Box-Cox з регуляризацією є беззаперечно кращою, адже вона точніше описує досліджувану залежність і забезпечує вищу достовірність прогнозів.

Порівнюючи з минулими дослідженнями, за допомогою регуляризації та врахування мультиколінеарності, вдалося досягти кращих результатів використовуючи такий підхід до нелінійної регресивної математичної моделі

Використання тривимірного перетворення Box-Cox є гарним рішенням для нормалізації вхідних даних.

Варто звернути увагу, що використання нелінійних регресійних моделей у контексті оцінювання якості програмного забезпечення часто виправдане тим, що взаємозв’язки між емпіричними даними не завжди лінійні.

З позиції практичного застосування, така модель дає змогу краще розуміти, які конкретні зміни в архітектурі чи структурі коду варто зробити, щоб поліпшити якість.

## 3.3 **Висновки до розділу 3**

У цьому розділі було зібрано метрики 56 десктоп застосунків на мові C#, проведено їх нормалізацію та очищено від викидів. Побудовано нелінійну регресійну модель для оцінювання метрики RFC десктопних застосунків на C# залежної від метрик, таких як CBO, WMC, для аналізу впливу архітектури програмного забезпечення на його якість.

Дослідження показало, що запропонована двофакторна регресійна модель забезпечує високий рівень передбачуваності.

Результати моделювання дозволяють краще зрозуміти взаємозв’язок між архітектурними метриками та якістю програмного забезпечення, а також виявити

ключові аспекти, які потребують оптимізації для підвищення якості десктопних застосунків.

Нелінійна регресійна модель є основою для створення автоматизованої системи оцінювання метрики RFC в застосунках на C#, для визначення їх якості.

# 4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ ОЦІНЮВАННЯ ЯКОСТІ ЗАСТОСУНКІВ НА C#

Після проведеного дослідження, аналізу існуючих методів та створення математичної моделі оцінювання метрики RFC десктоп застосунків на C#, постала необхідність в створенні програми (системи) для оцінювання якості ПЗ на основі результатів математичної моделі.

Таке програмне забезпечення може використовуватись менеджерами та інженерами, які займаються розробкою програмних продуктів.

Воно повинно забезпечити користувачеві зручний, простий та інтуїтивно зрозумілий інтерфейс.

Таке програмне забезпечення буде корисним для менеджерів та інженерів, які займаються проєктуванням програмного продукту.

Програмне забезпечення повинно надати користувачу зручний у використанні, простий та інтуїтивно зрозумілий інтерфейс.

Для реалізації програмного продукту обрано мову програмування Python бібліотеки joblib, streamlit, matplotlib [36-38]

## 4.1 **Вимоги до програмного продукту для оцінювання якості десктоп застосунків на C#**

Нижче наведено технічні та функціональні вимоги до наведеного застосунку, враховуючи його призначення та можливості, які випливають з аналізу коду.

**Функціональні вимоги:**

– введення даних. Користувач може ввести значення метрик якості ПЗ (CBO, RFC, WMC) вручну через інтерактивні поля або завантажити CSV-файл із даними;

– нормалізація даних. Використовуються перетворення Бокса-Кокса вхідних значень для нормалізації даних;

– обчислення прогнозу якості ПЗ. На основі вхідних метрик застосунок

прогнозує певний показник якості ПЗ за допомогою побудованої математичної моделі, довірчих та прогнозних інтервалів;

– візуалізація результатів. Вікно зі значеннями довірчого та прогнозного інтервалів та значенням обчисленої метрики RFC;

– експорт результатів. Користувач може завантажити результати у форматі CSV, включно з вхідними метриками, прогнозом, інтерпретацією та розрахованими інтервалами.

**Вимоги до організації вхідних і вихідних даних.**

Вхідні дані містять:

– показники метрик CBO, RFC та WMC. Дані завантажуються з файлу у форматі CSV, який містить стовпці CBO, RFC та WMC або вводяться вручну.

Вихідні дані:

– інтерпретація якості ПЗ, на основі обчислення метрики RFC від CBO та WMC;

– довірчі та прогнозні інтервали, що характеризують невизначеність прогнозу;

– можливість збереження отриманих результатів у форматі CSV-файлу для подальшого аналізу чи документації.

**Вимоги до надійності ПЗ:**

– система повинна правильно обробляти вхідні дані та сигналізувати про помилки форматування чи відсутність потрібних стовпчиків у завантаженому CSV;

– при відсутності моделі або пошкодженому файлі моделі система має виводити відповідне повідомлення про помилку;

– повинна бути передбачена стійкість до некоректного вводу користувача (наприклад, виведення помилки, якщо введено від’ємні значення або дані поза очікуваним діапазоном).

**Вимоги до умов експлуатації:**

– система передбачається для експлуатації у стандартних умовах персонального чи локального серверного середовища;

– користувач повинен мати доступ до інтернет-браузера або можливість

запуску додатку за допомогою пакету Streamlit (локально або на віддаленому сервері);

– немає вимог до специфічних кліматичних чи екстремальних умов, оскільки застосунок є програмним та виконується у віртуальному середовищі.

**Програмна сумісність:**

– сумісність із сучасними веб-браузерами (Chrome, Firefox, Edge, Safari);

– сумісна при запуску з локального середовища за встановлено бібліотекою streamlit.

**Користувацьке програмно-апаратне забезпечення має відповідати наступним мінімальним вимогам:**

– наявність ПК або ноутбука зі встановленою ОС (Windows 10+, Linux, macOS) та підтримкою сучасного веб-браузера.

– мінімум двохядерний процесор з тактовою частотою 1.8 ГГц;

– оперативна пам’ять (RAM) мінімум 2 ГБ;

– жорсткий диск з достатньою кількістю вільного місця для зберігання результатів (не менше 100 Мб);

– встановлений інтерпретатор Python 3.x, а також можливість інсталяції необхідних бібліотек через pip, якщо система запускається локально.

– доступ до мережі Інтернет бажаний (для завантаження залежностей, оновлень та можливого віддаленого розгортання застосунку).

– монітор з роздільною здатністю не нижче 1280x720 для комфортної роботи з графічним інтерфейсом застосунку.

## 4.2 **Побудова діаграм варіантів використання ПЗ для оцінювання якості десктоп застосунків на C#**

Універсальна мова моделювання (UML) є загальноприйнятим стандартом позначень, призначеним для формалізованого визначення, візуалізації та документування об’єктно-орієнтованих моделей програмних систем. На відміну від мов програмування, UML не регламентує порядок виконання операцій і не містить інструкцій, які визначали б етапи розробки системи. Її основна мета

полягає у забезпеченні наочності структури та логіки програмного продукту, а також у полегшенні комунікації між розробниками та іншими зацікавленими сторонами. Оскільки UML усталено використовується в галузі програмної інженерії як стандартизований графічний засіб опису ПЗ, вона є доречним інструментом для моделювання розрахункових програм, зокрема, застосунку для оцінювання якості десктоп застосунків на мові C# [39].

Діаграми варіантів використання описують дії, які система здатна виконувати у відповідь на зовнішні стимули від користувачів або інших програмних компонентів. Таким чином, вони відображають очікувану функціональність системи та створюють основу для визначення вимог до її подальшого проєктування й реалізації [40].

Діаграма варіантів використання для оцінювання якості десктоп застосунків на мові C# наведена на рис. 4.1.

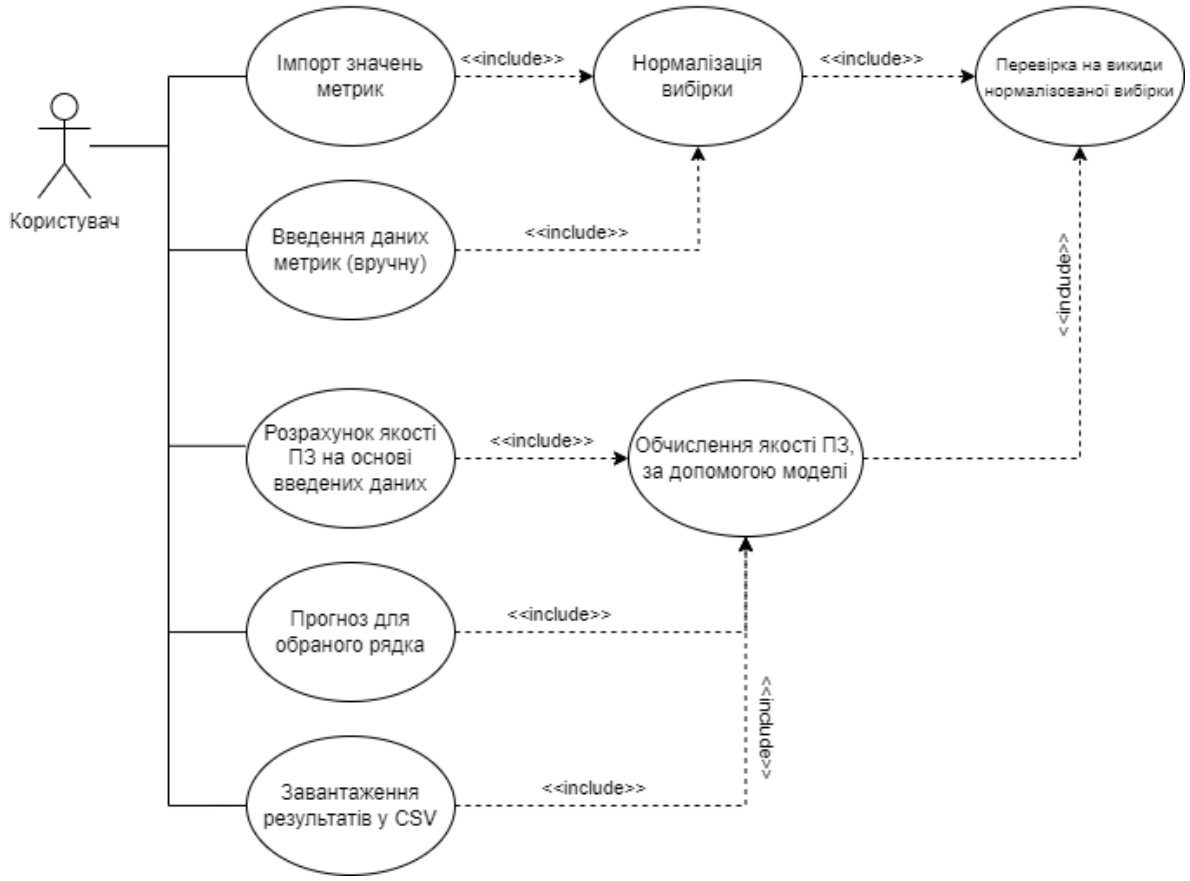


Рисунок 4.1 – Діаграма варіантів використання для оцінювання якості десктоп застосунків на мові C#

Діаграма варіантів використання є дуже важливою для проєктування системи та подальшої розробки. Наступним кроком необхідно розробити специфікації виявлених варіантів використання.

### 4.3 **Опис специфікацій варіантів використання ПЗ**

Для кожного з варіантів використання, які були зазначені у попередньому розділі на рисунку, слід підготувати специфікацію.

Кожна специфікація включає такі елементи як: стислий опис, передумови, основний сценарій подій, альтернативний сценарій подій та постумови.

Розуміння специфікацій варіантів використання ПЗ дає можливість більш детально розуміти процес побудови застосунку та його функціоналу. Іншими словами, специфікації варіантів використання – це детальний опис функціональних сценаріїв, що виступає опорою для проєктування, реалізації та тестування системи.

Ретельно підготовлені специфікації варіантів використання полегшують перехід від вимог до архітектури та коду, а також є основою для написання тест-кейсів. Специфікації варіантів використання сприяють кращому розумінню функціоналу системи. Специфікації варіантів використання наведені в таблицях 4.1 – 4.5.

Таблиця 4.1 – Опис специфікації варіанту використання «Імпорт значень метрик»

| **Складова** | **Опис** |
|---|---|
| Короткий опис | Імпорт вхідних даних метрик з csv файлу |
| Передумова | Відкриття сайту (запуск на сервері) |
| Потік подій | Користувач обирає вхідний файл csv, який містить CBO, RFC та WMC і завантажує його на сайт |
| Альтернативний потік подій | Повідомлення про помилку, якщо не вірний формат файлу або назва колонок |
| Постумови | Запускається процес розрахунку параметрів, нормалізації та видалення викидів |

Таблиця 4.2 – Опис специфікації варіанту використання «Введення даних метрик (вручну)»

| Складова | Опис |
|---|---|
| Короткий опис | Введення вхідних даних метрик у відповідні поля |
| Передумова | Відкриття сайту (запуск на сервері) |
| Потік подій | Користувач вручну вводить значення CBO, RFC та WMC у відповідні поля |
| Альтернативний потік подій | Повідомлення про помилку, якщо введено невірне значення |
| Постумови | Запускається процес нормалізації |

Таблиця 4.3 – Опис специфікації варіанту використання «Розрахунок якості ПЗ (вручну)»

| Складова | Опис |
|---|---|
| Короткий опис | Програма бере введені користувачем значення та передає їх у модель прогнозування. За допомогою побудованої моделі обчислює метрику RFC, визначає якість ПЗ, довірчий та прогнозний інтервали |
| Передумова | Введення вхідних даних вручну у відповідні поля та успішне виконання нормалізації |
| Потік подій | Користувач натискає на кнопку «Розрахувати якість ПЗ (вручну)», модель обчислює наближений показник якості ПЗ, довірчий та прогнозний інтервали |
| Альтернативний потік подій | - |
| Постумови | Вивід результатів |

Таблиця 4.4 – Опис специфікації варіанту використання «Прогноз для обраного рядка» у програмному забезпеченні

| Складова | Опис |
|---|---|
| Короткий опис | Програма передає дані з обраного рядка в побудовану математичну модель. |
| Передумова | Завантаження файлу з вхідними даними, їх нормалізація та видалення викидів. |
| Потік подій | При натисканні кнопки «Прогноз для обраного рядка», модель обчислює метрику RFC, довірчий та прогнозний інтервали |
| Альтернативний потік подій | Повідомлення про помилку значення, якщо вказати не існуючий номер рядка |
| Постумови | Вивід результатів |

Таблиця 4.5 – Опис специфікації варіанту використання «Завантаження результатів у CSV»

| Складова | Опис |
|---|---|
| Короткий опис | Програма формує CSV файл з даними про якість, оцінкою якості та значеннями інтервалів і починає завантаження файлу |
| Передумова | Виконання розрахунків за допомогою моделі для рядка або вручну |
| Потік подій | Користувач натискає на кнопку «Завантажити результати у CSV», модель збирає дані якості, показник якості, інтервали і конвертує ці дані в CSV файл, після чого починається завантаження |
| Альтернативний потік подій | - |
| Постумови | - |

Застосування специфікації варіантів використання на всіх рівнях моделювання дає змогу не лише досягти уніфікації позначень для відображення функціональності підсистем і системи загалом, а й стає потужним інструментом поступового та послідовного уточнення вимог до проєктованої системи.

Це досягається через методичний перехід від найвищого рівня абстракції – пакетів системи – до найдрібніших елементів, зокрема операцій класів. Такий підхід сприяє глибшому осмисленню внутрішніх взаємозв’язків, допомагає виявляти пропуски та суперечності у вихідних вимогах і забезпечує єдину уніфіковану базу для спільної роботи різних учасників проєктної команди (аналітиків, архітекторів, розробників, тестувальників) [41].

Деталізоване представлення функціональності системи полегшує її масштабування, модернізацію та документування, а також мінімізує ризики некоректного розуміння логіки системи на етапах проєктування та реалізації.

## 4.4 **Проєктування діаграм діяльності ПЗ**

Проєктування діаграми діяльності програмного забезпечення (ПЗ) є процесом створення графічної моделі, яка відображає послідовність дій, логіку виконання процесів, розгалуження й можливі паралельні потоки роботи всередині системи. Діаграми діяльності допомагають краще відображати сценарії використання дії [42].

Для демонстрації була підготовлена діаграма діяльності для варіанту використання «Імпорт значень метрик» (рис. 4.2).

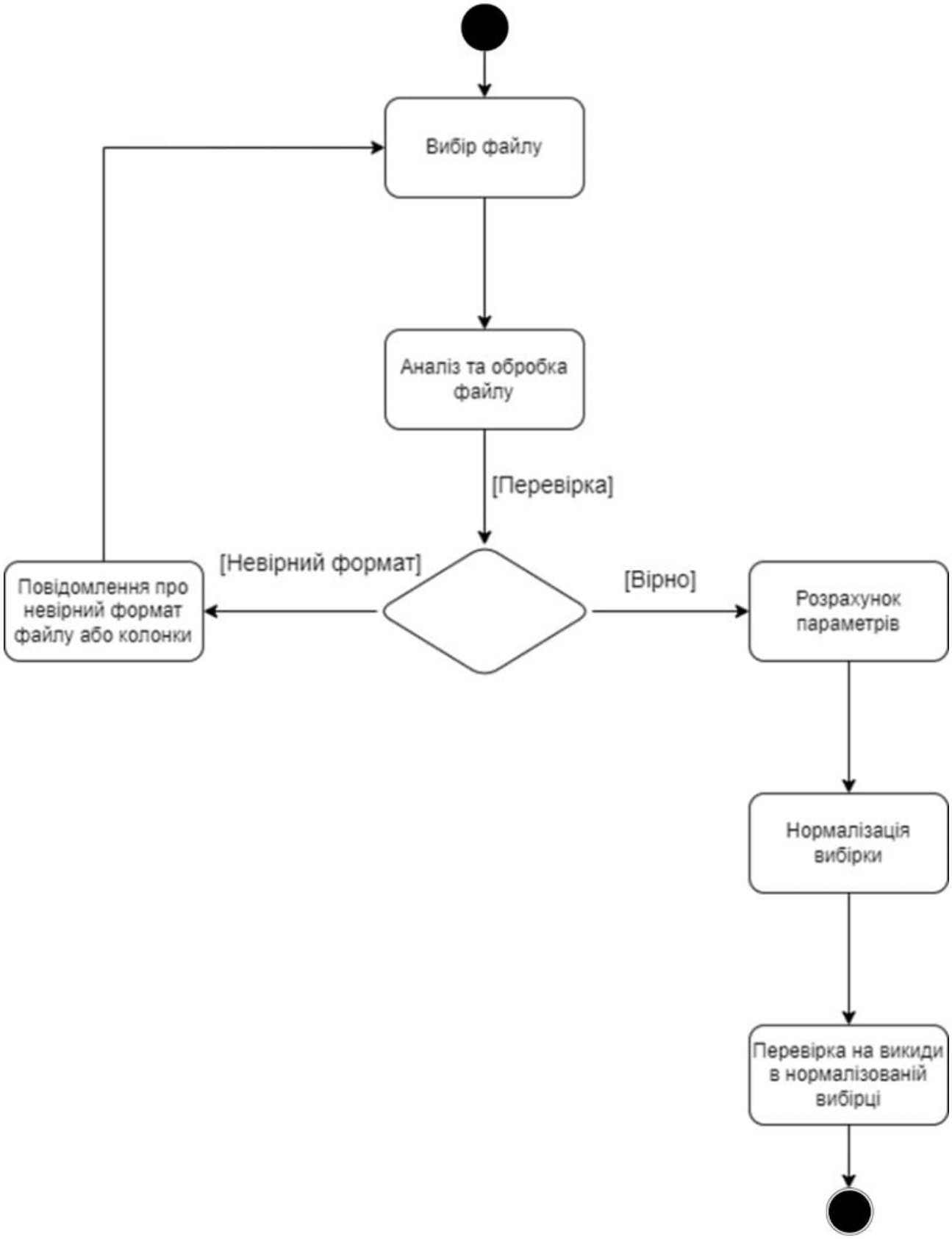


Рисунок 4.2 – Діаграма діяльності варіанту використання «Імпорт значень метрик»

## 4.5 **Проєктування інтерфейсу користувача для застосунку**

Процес проєктування інтерфейсу користувача (UI) для застосунку становить собою низку послідовних кроків, спрямованих на формування зручної, логічної та естетично привабливої взаємодії між користувачем і програмним продуктом. В основі цього процесу лежить розуміння потреб цільової аудиторії, аналіз функціональних вимог і дотримання принципів ергономіки та юзабіліті [43].

На етапі концептуального проєктування визначаються основні функціональні блоки застосунку та їхні взаємозв’язки, формуються принципи організації інформації на екрані. Результатом може стати створення інформаційної архітектури: ієрархічної структури меню, навігаційних шляхів, логіки переходів між екранами та розташування основних елементів керування.

Після цього проєктувальники переходять до розроблення прототипів. Спершу створюються низькорівневі прототипи (wireframes), які фокусуються на розташуванні елементів та логіці взаємодії без надмірної уваги до естетики. Ці чорнові ескізи дають змогу швидко оцінити зручність сценаріїв використання, виявити потенційні проблеми в навігації, структурі чи логіці роботи інтерфейсу [44].

На основі перевірених прототипів розробляються більш деталізовані інтерактивні макети, що наближаються до кінцевого вигляду інтерфейсу. Тут вже застосовуються принципи візуального дизайну, підбираються кольорові гами, шрифти, іконки та графічні елементи, що відповідають стилю бренду та естетичним вимогам. Кінцева мета – досягти максимальної зрозумілості інтерфейсу, зробити його візуально привабливим та інтуїтивним.

Обов’язковим етапом є тестування отриманих прототипів з реальними або потенційними користувачами. Під час такого оцінювання вимірюють легкість опанування інтерфейсу, зручність взаємодії з елементами керування, швидкість виконання типових завдань. На основі результатів випробувань інтерфейс удосконалюється: змінюють компонування, уточнюють підписи, усувають «сліпі плями» в навігації, покращують зчитуваність елементів [45].

Завершальним етапом є впровадження затвердженого дизайну та візуальних

специфікацій у реальну реалізацію. Дизайнери спільно з розробниками перевіряють відповідність застосунку визначеній концепції, його сумісність із різними пристроями та екранами, враховують потреби в адаптивному дизайні, а також забезпечують доступність для користувачів з особливими потребами.

Таким чином, процес проєктування інтерфейсу користувача – це ітеративний і багатокроковий підхід, спрямований на постійне покращення взаємодії між користувачем і застосунком. Поєднання користувацьких досліджень, теоретичних принципів юзабіліті, технічних обмежень та перевіреного на практиці підходу дає змогу створити інтерфейс, що забезпечує комфортну, зрозумілу та продуктивну роботу з програмним продуктом.

Оцінивши функціонал застосунку, було створено інтерфейс, який відповідає всім технічним та функціональним вимогам для системи оцінювання якості ПЗ.

На рис. 4.3 продемонстровано вікно завантаження вхідних даних з файлу.

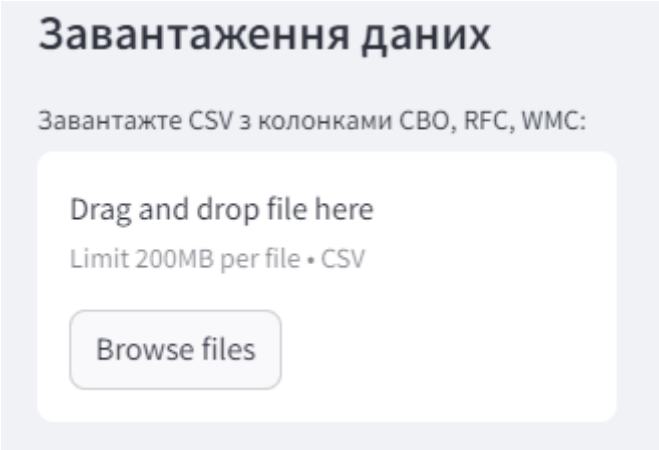


Рисунок 4.3 – Вікно завантаження вхідних даних з CSV файлу

Базовий дизайн системи починається зі стандартного вводу даних метрик (вручну) і продемонстрований на рис. 4.4.

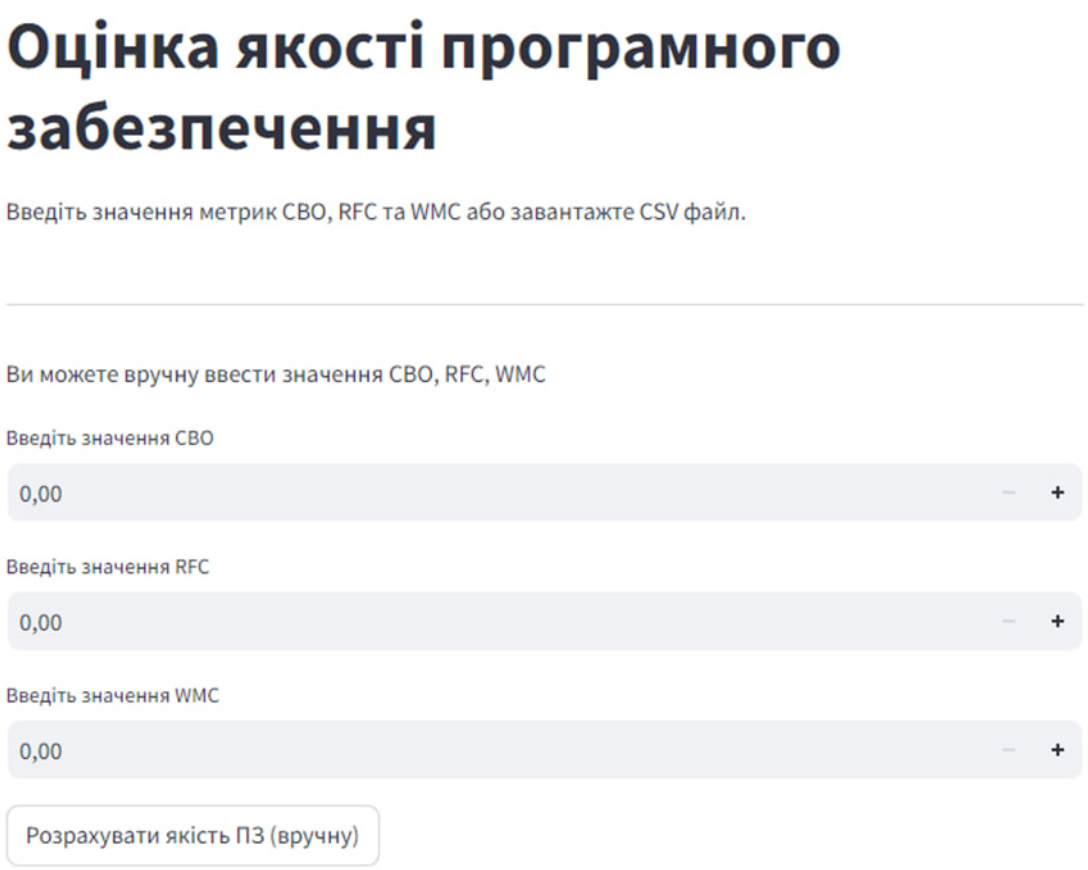


Рисунок 4.4 – Початкове вікно системи оцінювання якості ПЗ

Також важливо чітко повідомляти про помилки користувачів, для цього розроблені спеціальні повідомлення, наприклад повідомлення про невірне заповнення CSV завантаженого файлу (рис. 4.5).

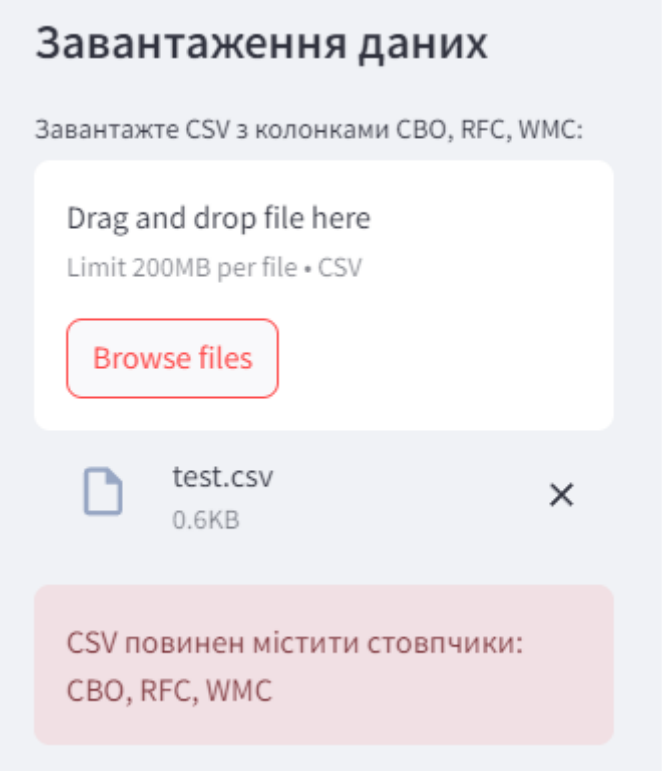


Рисунок 4.5 – Повідомлення про помилку відсутності назви стовпчиків

Основний розрахунку якості ПЗ з завантаженого файлу вхідних даних наведено на рис. 4.6, а на рис. 4.7 – кнопка завантаження результатів та рекомендації щодо покращення якості ПЗ.

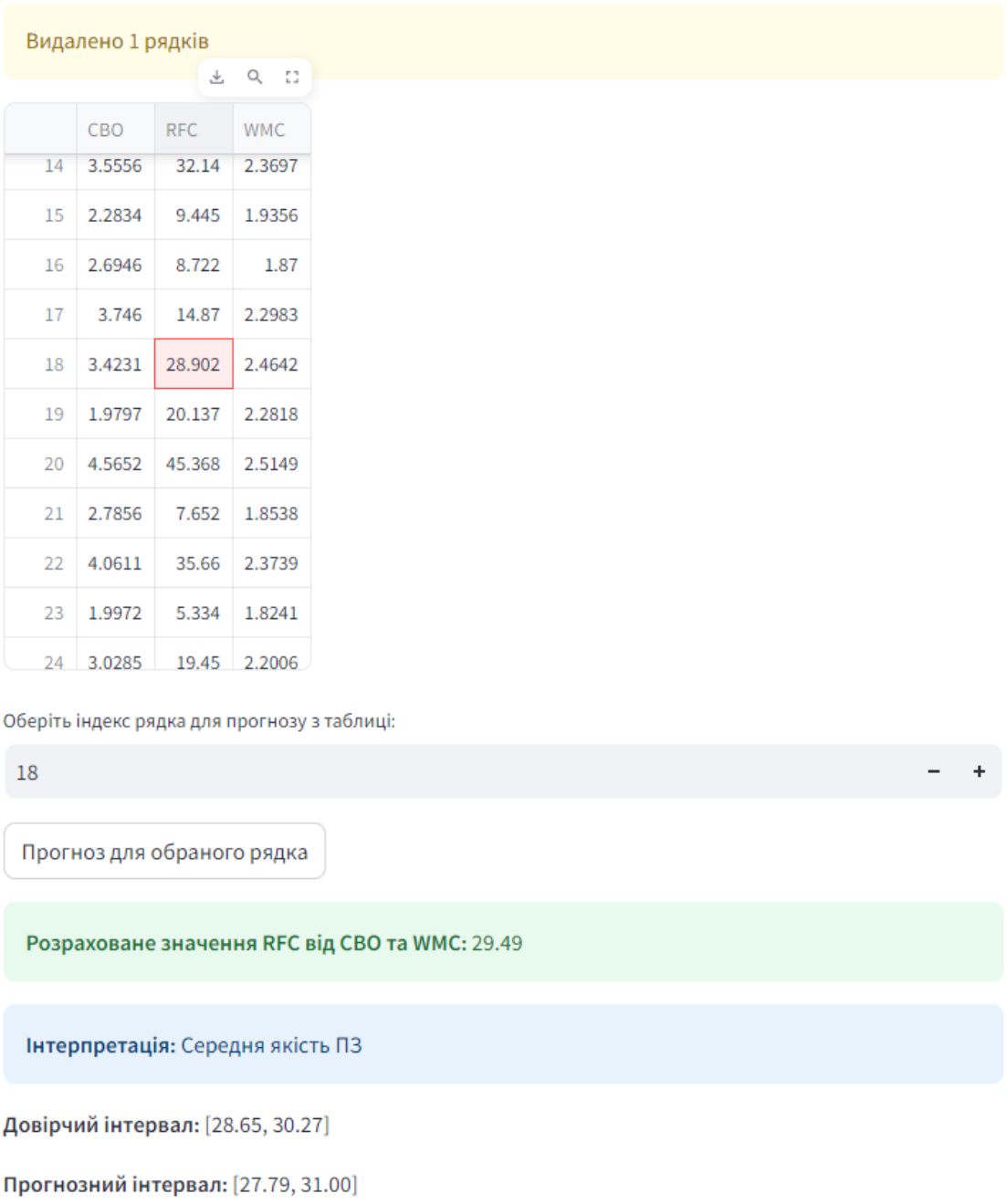


Рисунок 4.6 – Інтерфейс розрахунку рядка з звантаженої таблиці

Розраховане значення RFC наближено до фактичного, що говорить про високу достовірність обчислень за допомогою математичної моделі.

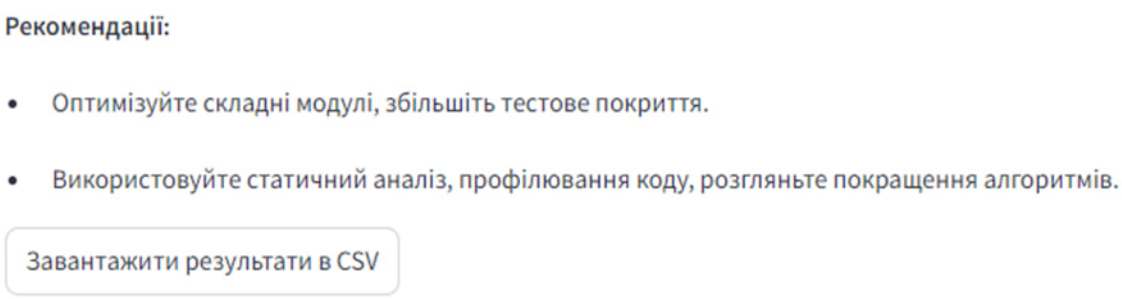


Рисунок 4.7 – Інтерфейс рекомендацій та завантаження результатів якості

Правильне проєктування інтерфейсу, дозволяє користувачам швидко та зручно орієнтуватись в ньому. Це додає привабливості застосунку.

Наступним кроком є розробка діаграми класів програмного забезпечення, що дозволить краще розуміти його майбутню структуру.

## 4.6 **Розробка діаграми класів програмного забезпечення**

Діаграма класів є одним із ключових засобів моделювання структурних аспектів програмної системи в межах мови UML. Її основне призначення полягає в систематичному відображенні статичної структури системи, що включає класи, їхні властивості (поля), методи (операції), а також асоціації, наслідування й інші відносини між цими класами. Завдяки цьому діаграма класів допомагає розробникам, аналітикам та іншим учасникам проєкту краще осмислити внутрішню логіку і композицію системи ще до написання програмного коду [46].

Основні цілі використання діаграми класів полягають у наступному:

– відображення структурних зв’язків між елементами системи;
– уточнення вимог і проєктних рішень;
– забезпечення платформи для комунікації;
– полегшення проєктування й рефакторингу;
– стандартизація підходу до проєктування.

Таким чином, діаграма класів – це інструмент, що допомагає систематизувати й формалізувати знання про статичну структуру програмного забезпечення, створити спільне інформаційне поле для всіх учасників розробки, а також підтримувати послідовний та раціональний проєктний процес.

Діаграма класів програмного застосунку для оцінювання якості наведена на рис. 4.8.

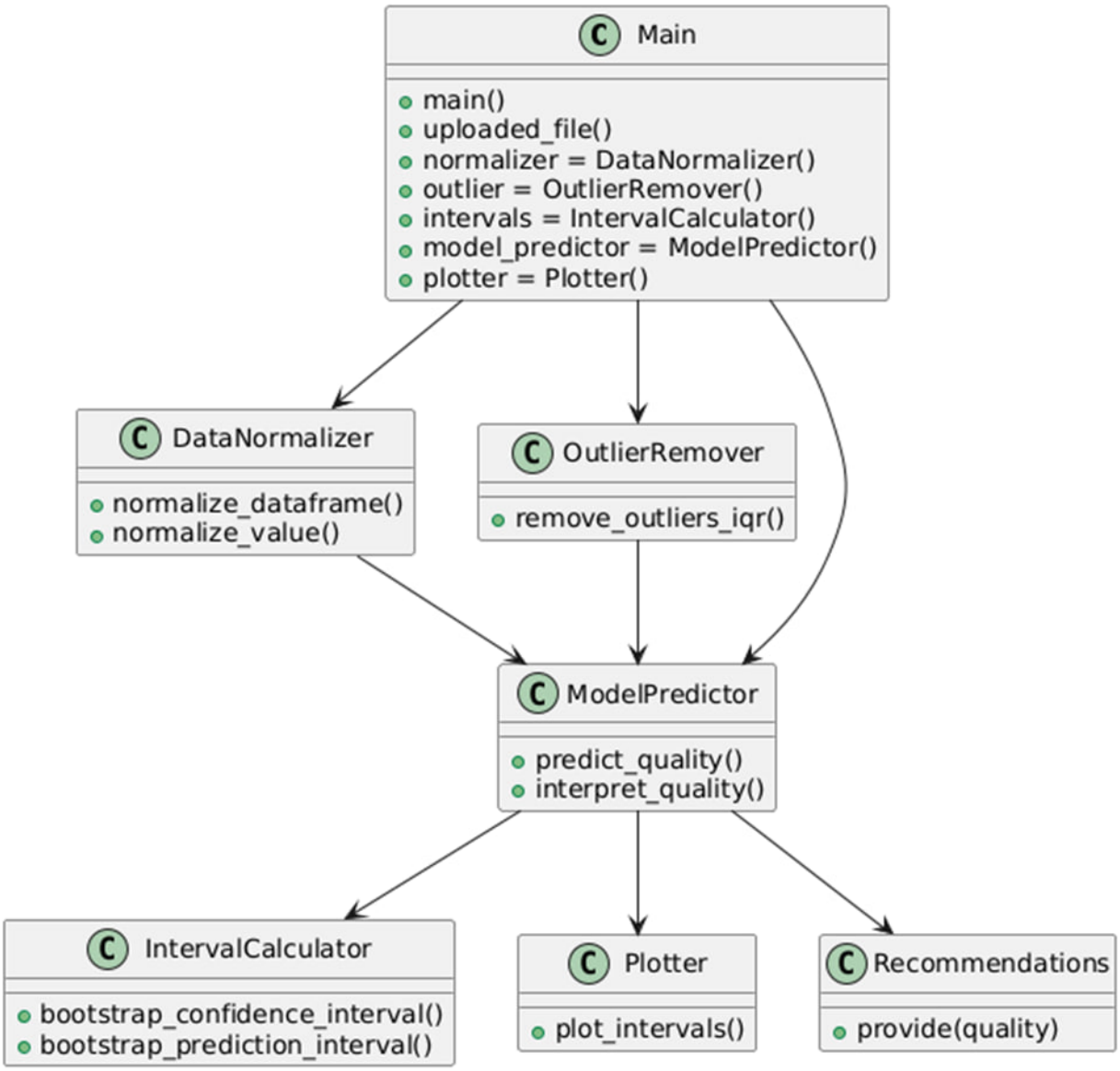


Рисунок 4.8 – Діаграма класів ПЗ для оцінювання якості десктоп застосунків на C#

На наведеній діаграмі класів UML зображено набір взаємопов’язаних класів, які спільно формують архітектуру програмної системи для оцінювання якості даних або результатів певного прогнозного моделювання. Кожен клас виконує окрему функціональну роль, а їхні взаємодії забезпечують послідовне перетворення вхідних даних, фільтрацію викидів, нормалізацію, побудову

моделей, розрахунок довірчих інтервалів та візуалізацію результатів.

У верхній частині перебуває клас Main, який слугує своєрідним «керівним центром» застосунку. Він ініціалізує та пов’язує між собою інші компоненти.

Це дозволяє аналізувати динамічну поведінку системи, виявляти потенційні проблеми у взаємодіях та оптимізувати структуру програмного забезпечення [47].

## 4.7 **Обґрунтування вибору технологій та реалізація застосунку**

Для реалізації програмного продукту було обрано мову програмування Python завдяки її популярності, простоті синтаксису та потужній екосистемі бібліотек для обробки даних, машинного навчання та розробки веб-застосунків. Python забезпечує ефективну інтеграцію різноманітних інструментів, що дозволяє швидко та зручно реалізувати складні обчислювальні завдання з мінімальними витратами часу.

Для реалізації інтерфейсу користувача була обрана бібліотека **Streamlit**. Бібліотека дозволяє легко створювати інтерактивні веб-застосунки на основі Python. Streamlit забезпечує зручний спосіб відображення даних, взаємодії з користувачем і генерації графічного інтерфейсу без необхідності у веб-програмуванні. У рамках проєкту Streamlit використовується для завантаження вхідних даних у форматі CSV, введення параметрів вручну, відображення результатів обчислень і візуалізації довірчих та прогнозних інтервалів.

Для збереження та завантаження побудованої математичної моделі у форматі .pkl обрана бібліотека **Joblib**.

Ця бібліотека дозволяє ефективно серіалізувати великі об'єкти, зокрема математичні моделі, що були побудовані і використовувати їх для подальшого обчислення без повторного навантаження на систему. Завдяки цьому значно знижується час виконання програми та підвищується продуктивність системи.

Для забезпечення візуалізації даних у вигляді діаграм та графіків обрано бібліотеку **Matplotlib**. Ця бібліотека є стандартом для побудови наочних графічних представлень, що допомагають краще інтерпретувати результати прогнозів і довірчі інтервали. У застосунку Matplotlib використовується для побудови

графіків, які показують прогнозовані значення разом із обчисленими довірчими та прогнозними інтервалами, що полегшує сприйняття інформації користувачем.

Додатково використано бібліотеку **Pandas**, яка є основним інструментом для роботи з табличними даними. Вона забезпечує завантаження, обробку та аналіз вхідних даних у форматі CSV.

Нижче наведено приклад коду побудови та зберігання математичної моделі.

```
file_path = "New_metrics.xlsx"
df = pd.read_excel(file_path)
cov = EmpiricalCovariance().fit(df_log)
mean = df_log.mean(axis=0)
def mahalanobis_distance(row):
    return mahalanobis(row, mean, cov.precision_)
df_log['Mahalanobis'] = df_log.apply(mahalanobis_distance, axis=1)
threshold = np.percentile(df_log['Mahalanobis'], 99)
cleaned_df = df_log[df_log['Mahalanobis'] <=
threshold].drop(columns=['Mahalanobis'])
num_remaining = cleaned_df.shape[0]
print(f"Залишилось програм після видалення викидів: {num_remaining} із
{df.shape[0]}")
target = 'RFC'
features = ['CBO', 'WMC']
X = cleaned_df[features].copy()
Y = cleaned_df[target].copy()
if (X <= 0).any().any():
    raise ValueError("Box-Cox трансформація вимагає позитивних значень для
всіх незалежних змінних.")
lambda_values = {}
X_transformed = X.copy()
for col in X.columns:
    X_transformed[col], lambda_val = boxcox(X[col])
    lambda_values[col] = lambda_val
pipeline = Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('ridge', Ridge(alpha=1.0))
])
pipeline.fit(X_transformed, Y)
Y_pred = pipeline.predict(X_transformed)
r2 = r2_score(Y, Y_pred)
mae = mean_absolute_error(Y, Y_pred)
model_info = {
    'pipeline': pipeline,
    'normalization_method': 'boxcox',
    'lambda_values': lambda_values
}
model_filename = "polynomial_ridge_pipeline.pkl"
joblib.dump(model_info, model_filename)
```

Розробка програмного продукту базується на об’єктно-орієнтованому підході, де окремі функціональні блоки системи реалізовані у вигляді класів: DataNormalizer для нормалізації даних, OutlierRemover для видалення викидів, ModelPredictor для прогнозування якості, IntervalCalculator для обчислення інтервалів і Plotter для візуалізації. Це дозволяє забезпечити розділення відповідальності, підтримуваність і масштабованість коду.

Таким чином, поєднання Python і вибраних бібліотек дозволило створити зручний, ефективний і надійний програмний продукт з інтуїтивно зрозумілим інтерфейсом користувача, розширеними можливостями для аналізу й візуалізації даних, а також функцією завантаження й збереження результатів у форматі CSV.

Повний код програми наведено в Додатку Б «Текст програми».

## 4.8 **Тестування розробленого програмного забезпечення**

Тестування розробленого програмного забезпечення є невіддільним етапом життєвого циклу проєкту, що дозволяє перевірити коректність роботи системи, виявити потенційні помилки та переконатися, що продукт відповідає функціональним і нефункціональним вимогам. Для тестування розробленого застосунку було використано ручне тестування, що охоплює різні аспекти роботи програми.

**Тестування функціональності.** Функціональне тестування було спрямоване на перевірку коректної роботи основних компонентів програми. Тестування охоплювало такі сценарії:

– завантаження користувачем вхідного файлу у форматі CSV. Було перевірено обробку коректних і некоректних файлів, включно з відсутністю обов'язкових стовпців (CBO, RFC, WMC). Програма успішно ідентифікує некоректний файл та виводить відповідне повідомлення про помилку;

– нормалізація даних. Тестування показало, що клас DataNormalizer правильно застосовує логарифмічну нормалізацію, а також коректно обробляє таблиці з нульовими або від’ємними значеннями;

– видалення викидів. Методи класу OutlierRemover успішно видаляють

аномальні значення для вказаних метрик про що свідчать тестові результати з реальними наборами даних (рис. 4.9);

– прогнозування якості. Клас ModelPredictor був протестований на працездатність моделі, завантаженої з файлу polynomial_model.pkl. Програма успішно виконує прогнозування для обраного рядка та повертає очікувані значення;

– обчислення довірчих та прогнозних інтервалів. Клас IntervalCalculator коректно обчислює інтервали, що було перевірено на тестових даних;


Рисунок 4.9 – Повідомлення про успішне видалення викидів

**Тестування на граничних значеннях.** Граничні тести були проведені для перевірки роботи програми з екстремальними значеннями параметрів:

– вхідні дані з дуже великими або дуже малими значеннями (CBO, RFC, WMC);

– відсутність значень у деяких рядках таблиці;

– порожній файл або файл без необхідних стовпців (рис. 4.10);

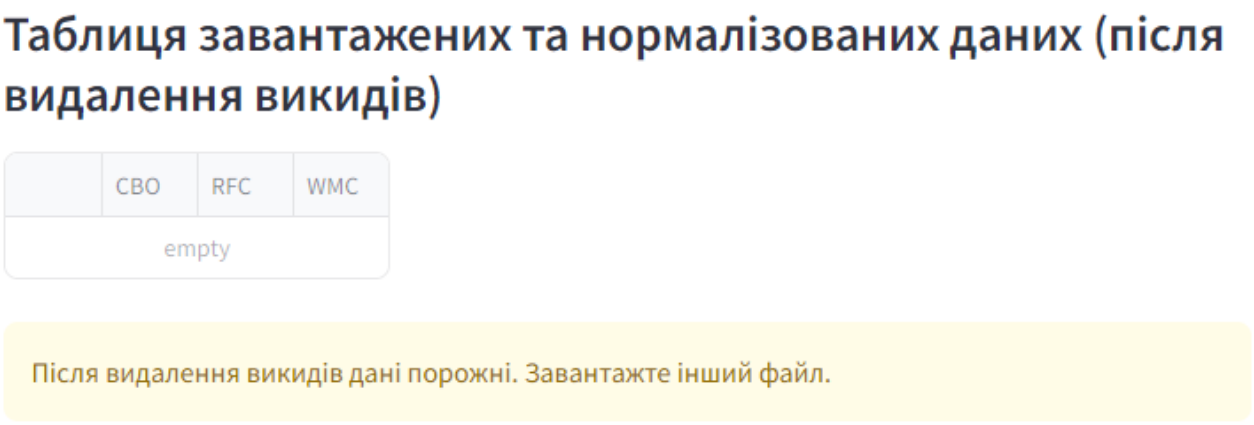


Рисунок 4.10 – Повідомлення про помилку через порожній файл

Тестування підтвердило, що система коректно обробляє ці випадки і виводить відповідні повідомлення або попередження.

Також було здійснено випробування програмного забезпечення згідно з програмою та методикою випробувань, яка викладена у Додатку Г.

Проведене випробування показало, що програмний застосунок відповідає вимогам Технічного завдання у Додатоку А.

Розроблений програмний продукт успішно пройшов усі етапи тестування, включно з функціональним, граничним і автоматизованим тестуванням. Програма є надійною, стійкою до помилок та готовою до використання користувачами для оцінювання якості програмного забезпечення.

## 4.9 **Висновки до розділу 4**

У четвертому розділі було розроблено програмне забезпечення для автоматизованого оцінювання якості програмних застосунків на основі двофакторної нелінійної регресійної математичної моделі, що оцінює метрику RFC за метриками CBO та WMC. Процес розробки включав вибір відповідних інструментів і технологій, реалізацію системи та тестування її функціональності та надійності.

Розробка системи здійснювалася мовою програмування Python, яка була

обрана завдяки її гнучкості, наявності потужних бібліотек та простоті інтеграції.

Для побудови графічного інтерфейсу застосунку використано Streamlit, що дозволяє створити інтуїтивно зрозумілий інтерфейс для кінцевого користувача. Додатково використані такі бібліотеки як Pandas (для обробки даних), Joblib (для завантаження моделі), Matplotlib (для візуалізації результатів) та інші.

Програмна архітектура була реалізована у вигляді окремих модулів, що відповідають за основні функціональні компоненти.

Результати тестування підтвердили працездатність програмного застосунку, його здатність обробляти дані, виконувати аналіз якості програмного забезпечення та надавати прогнози з високим рівнем достовірності.

Математична модель, що лежить в основі програми, продемонструвала свою ефективність для аналізу якості на основі обраних метрик (RFC, CBO, WMC). Вона забезпечує можливість прогнозування якості та надає інтерпретацію результатів, що дозволяє користувачам отримувати цінні аналітичні висновки про структуру та складність програмного забезпечення.

Таким чином, у результаті розробки було створено функціональний, надійний та зручний інструмент для оцінювання якості програмного забезпечення за допомогою метрик CBO, RFC та WMC на основі двофакторної нелінійної регресійної моделі. Проведене тестування підтвердило стабільну роботу системи й високу точність результатів, що робить розроблений продукт готовим до використання як у навчальних, так і у інших процесах.

# 5 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ РОЗРОБКИ ТА ВПРОВАДЖЕННЯ СИСТЕМИ ОЦІНЮВАННЯ ЯКОСТІ C# ЗАСТОСУНКІВ

Витрати на розробку включають прямі та непрямі витрати, необхідні для створення програмного продукту.

## 5.1 **Розрахунок витрат на створення та експлуатацію ПЗ для оцінювання якості застосунків на C#**

Програмне забезпечення, розробляє фахівець, який має щомісячний оклад у розмірі 19 000 грн. Додаткова заробітна плата для нього, в середньому становить 10% від основної, тобто 1 900 грн. Таким чином, загальна заробітна плата розробника на Python складає 20900 грн на місяць. При цьому вартість сучасного комп'ютера складає в середньому 22 000 грн.

Додаткові витрати на електроенергію становлять 4,32 грн/кВт·год, незалежно від обсягу споживання.

Витрати на допоміжні матеріали, які знадобяться під час розробки наведені в табл. 5.1.

Таблиця 5.1 – Витрати на допоміжні матеріали під час розробки ПЗ

| **Витрати** | **Сума, грн.** |
|---|---|
| Папір для принтера | 179,00 |
| Заправлення картриджа до принтера | 200,00 |
| Флеш-накопичувач «Goodram UCO2 64GB» | 260,00 |
| Непередбачені витрати | 300,00 |
| Загально | 939,00 |

Вартість розробки системи розраховується за наступною формулою:

$$С_{пр} = (В_{зп} + В_{сф} + В_{зг} + В_{е}) * Т + В_{м},$$

де Т – тривалість розробки, місяців;

$В_{зп}$ – основна і додаткова заробітна плата, грн.;

$В_{сф}$ – відрахування у соціальні фонд (38% від основної і додаткової заробітної плати), грн;

$В_{зг}$ – загальногосподарські витрати (10% від заробітної плати), грн;

$В_{е}$ – витрати на електроенергію;

$В_{м}$ – загальні витрати на матеріали.

Розрахуємо $В_{е}$ при споживанні 0,38 кВт, тривалості роботи на місяць в днях та робочих годинах 22 * 8 = 176 годин, вартості кіловат-години електроенергії 4,32 грн. складає:

$$В_{е} = 176 * 4,32 * 0,38 = 288,92 \text{ грн}$$

Загальні витрати, які потрібні для розробки ПЗ наведені в табл. 5.2.

Таблиця 5.2 – Загальні витрати під час розробки ПЗ

| **Витрати** | **Одиниця виміру** | **Кількість** |
|---|---|---|
| Тривалість розробки ПЗ | Міс. | 2,00 |
| Основна і додаткова заробітна плата | Грн. | 20900,00 |
| Відрахування у соціальні фонди | Грн. | 7942,00 |
| Загальногосподарські витрати | Грн. | 2090,00 |
| Витрати на допоміжні матеріали | Грн. | 939,00 |
| Витрати на електроенергію | Грн. | 288,92 |

Посилаючись на формулу 5.1, вартість програми розраховуємо наступним чином:

$$С_{пр} = (20900,00 + 7942,00 + 2090,00 + 288,92) * 2 + 939,00 = 63380,84$$

Тобто, вартість програми складає 63380.84 грн за весь період розробки.

Експлутаційні витрати для технічного засобу (комп’ютера) розрахуємо за як:

$$В_{зр} = А_{об} + В_{м} + В_{е},$$

де $А_{об}$ – амортизація відрахування, грн;

$В_{м}$ – річні витрати на матеріали, грн;

$В_{е}$ – витрати на електроенергію, грн.

Амортизація відрахування на устаткування складають 60% балансової вартості на рік:

$$А_{об} = 22\,000 * 0{,}6 = 13200{,}00 \text{ грн.}$$

У масштабах закладу річні витрати на основні і допоміжні матеріали (носії, папір, …) визначаються в розмірі 5% вартості основного устаткування:

$$В_{м} = 22\,000 * 0{,}05 = 1100{,}00 \text{ грн}$$

Річний обсяг робіт комп`ютера у годинах визначається наступним чином:

$$Ф_{м} = 262 * Т_{з},$$

де $Т_{з}$ – це середнє денне завантаження устаткування комп’ютера (близько 7 годин);

262 – середня кількість робочих днів на рік.

Отже, річний обсяг роботи комп`ютера складе 262 * 7 = 1834 годин.

Витрати на електроенергію $В_{е}$ при 1834 годинах роботи на рік складуть:

$$В_{е} = 1834 * 4{,}32 * 0{,}38 = 3010{,}70 \text{ грн.}$$

Експлуатаційні витрати для комп`ютера за рік складуть:

$$В_{зр} = 13200{,}00 + 1100{,}00 + 3010{,}70 = 17310{,}70 \text{ грн.}$$

Отже, на перший рік створення та експлуатації програмного забезпечення витрати наступні:

$$В_{се} = 63380{,}84 + 17310{,}70 = 80691{,}54 \text{ грн.}$$

## 5.2 **Аналіз економічної ефективності розробки та впровадження ПЗ**

Основним показником економічної ефективності функціонування програмного забезпечення (ПЗ) є його економічний ефект, який оцінюється як різниця між отриманими вигодами від використання ПЗ та витратами на його розробку, впровадження та підтримку.

Функціонування ПЗ має на меті зниження витрат на певні процеси. Наприклад:

– зменшення витрат на ручну обробку даних;

– скорочення часу на виконання завдань;

– оптимізація використання ресурсів.

Розрахуємо економічну ефективність використання ПЗ, грунтуючись на тому, що впровадження програмного продукту зменшує кількість потрібних працівників (за експертною оцінкою, приблизно одну особу), яких потрібно було б задіяти для обробки інформації.

Зарплата працівника на рік складає:

$$19000 * 12 * 1 = 228000 \text{ грн.}$$

Тобто, якщо річний економічний ефект розраховується як:

$$\mathrm{E_{рік}} = \Delta \mathrm{C_в} - \mathrm{E_в} * k \ ,$$

де $\Delta \mathrm{C_в}$ – вивільнені кошти після впровадження продукту (228 000 грн.) та мінус експлуатаційні витрати (17310,70 грн.);

$\mathrm{E_в}$ – одноразові витрати на впровадження продукту, тобто 80691,54 грн;

$k$ – це коефіцієнт ефективності, який дорівнює 0,6;

Прорахуємо це:

$$\mathrm{E_{рік}} = 228000{,}00 - 17310{,}70 - 80691{,}54 * 0{,}6 = 77998{,}66 \text{ грн.}$$

Термін окупності ПЗ в такому разі розрахуємо:

$$\mathrm{T} = \frac{\mathrm{E_в}}{\Delta \mathrm{C_в}} = \frac{80691{,}54}{210689{,}30} \approx 0{,}38 \ .$$

Що означає, строк окупності системи складає майже 4 місяці.

Іншими словами, через 4 місяці після впровадження програмного забезпечення всі витрати, вкладені у його розробку та впровадження, повністю компенсуються за рахунок отриманої вигоди (економії витрат або збільшення прибутку) та почне приносити чистий економічний ефект.

## 5.3 **Висновки до розділу 5**

В даному розділі було проведено підрахунки вартості створення та використання програмного забезпечення для оцінювання якості десктоп застосунків на C#. За результатами проведених обчислень, можна побачити, що ПЗ окупиться протягом першого року експлуатації та почне приносити економічну вигоду через 4 місяці. Можна зробити висновок, що програмне забезпечення є економічно вигідним та обґрунтованим з фінансової точки зору.

# 6 ОХОРОНА ПРАЦІ

Охорона праці – це цілісна система заходів, спрямованих на збереження життя, здоров'я та працездатності працівників у процесі їх трудової діяльності. Вона об'єднує правові, організаційні, соціально-економічні, санітарно-гігієнічні, технічні та профілактичні заходи, які регламентуються низкою нормативно-правових актів України. До найважливіших актів у сфері охорони праці належать Закон України «Про охорону праці», Закон України «Про загальнообов’язкове державне соціальне страхування від нещасного випадку на виробництві та професійного захворювання» (№ 1105 від 23.09.1999 р.), а також Порядок розслідування та обліку нещасних випадків, професійних захворювань і аварій на виробництві (№1112 від 25.08.2004 р.) [48].

Основна функція охорони праці полягає у створенні безпечних умов роботи шляхом мінімізації впливу шкідливих і небезпечних факторів на здоров'я людини. До основних завдань належать:

– забезпечення безпечності обладнання, технологічних процесів і робочих місць;

– навчання працівників методам безпечної праці та використанню засобів індивідуального захисту;

– контроль за дотриманням санітарно-гігієнічних норм і мікроклімату робочої зони.

Робоче місце – це конкретна зона, оснащена необхідним обладнанням для виконання трудових завдань. Умови праці визначаються сукупністю факторів виробничого середовища, включаючи температуру, освітлення, рівні шуму, електромагнітні випромінювання тощо.

## 6.1 **Аналіз шкідливих факторів в офісному приміщення з ЕОМ**

Небезпечні фактори можуть призвести до раптового погіршення здоров'я або травм. Прикладом є механічні ушкодження чи короткі замикання обладнання.

Шкідливі фактори, навпаки, впливають на організм тривалий час, що може призвести до професійних захворювань, наприклад, хронічних болів через випромінювання чи шум.

У процесі роботи працівників офісу, які використовують електронно-обчислювальні машини (ЕОМ), виділяють два основні типи негативних чинників, які розглядаються далі.

**Фізичні фактори**, що спричиняються впливом робочого середовища, зокрема електромагнітне випромінювання, освітленість, шум, температура та мікроклімат.

**Психофізіологічні фактори**, які охоплюють як фізичне навантаження, так і нервово-психічне перенапруження, що проявляється у вигляді статичної напруги, зорового перевантаження та емоційного виснаження.

До фізичних факторів відносяться:

– електромагнітне випромінювання;
– освітлення
– температура та вологість повітря
– шум

Розберемо ці фактори по окремості більш детально.

**Освітлення** є критичним фактором для підтримки зорової працездатності та загального комфорту працівника. Раціонально організоване світло не лише забезпечує чітке сприйняття об’єктів, а й позитивно впливає на психоемоційний стан людини, знижує втомлюваність і підвищує продуктивність праці.

Згідно з гігієнічними нормами ДСН 3.3.6.042-99, освітлення повинно відповідати наступним критеріям:

– яскравість повинна бути достатньою та рівномірною, не допускаючи утворення тіней чи відблисків;
– світло повинно мати спектр, максимально наближений до природного освітлення;
– оптимальна освітленість робочої поверхні у зонах розташування документів чи ЕОМ має становити 300–500 люкс;

– в якості джерела світла рекомендується використовувати люмінесцентні лампи з маркуванням ЛБ (білого світла).

**Електромагнітне випромінювання**, де ЕОМ є джерелом електромагнітних полів, які можуть впливати на організм людини. Хоча сучасні комп'ютери відповідають стандартам безпеки, тривала робота поруч з ними вимагає дотримання рекомендацій щодо безпечної експлуатації.

**Шум**. Одним із важливих чинників, що знижує концентрацію уваги, є шум. Тривала дія підвищеного шуму спричиняє порушення роботи центральної нервової системи, знижує слух, викликає головний біль, стомлюваність та дратівливість. Відповідно до нормативних документів ГОСТ 12.1.003-85 і СН 3222-85, допустимі рівні шуму для офісних працівників і користувачів ЕОМ наведено в таблиці 6.1.

Таблиця 6.1 – Допустимі рівні шуму для працюючих з ЕОМ

| Тип діяльності | Рівень шуму, дБА |
|---|---|
| Програмісти ЕОМ | 50 |
| Оператори в залах обробки інформації | 65 |
| Приміщення з шумними агрегатами ЕОМ | 75 |

Зниження рівня шуму досягається за рахунок застосування звукоізоляції обладнання, удосконалення технологічних процесів і регулярного технічного обслуговування пристроїв, що генерують шум.

**Мікроклімат** визначає умови перебування працівника та безпосередньо впливає на його самопочуття і продуктивність.

Самопочуття та працездатність людини значною мірою залежать від її теплового стану, який формується під впливом різноманітних фізичних чинників виробничого середовища. До них належать температура повітря, рівень вологості, швидкість повітряних потоків, атмосферний тиск та концентрація кисню. Недостатня кількість кисню в повітрі (менше ніж 12%) призводить до утруднення дихання, змушуючи людину дихати частіше. Такий стан може переноситися не

більше 30 хвилин. Перегрів організму найчастіше виникає через надмірне теплове випромінювання від нагрітих поверхонь. У відповідь на перегрівання активізуються захисні механізми організму: збільшується інтенсивність роботи серцево-судинної та дихальної систем, а також посилюється потовиділення, яке може досягати до 5 літрів за робочу зміну. При цьому відбувається втрата мінеральних солей та важливих вітамінів, що негативно позначається на фізіологічному стані людини.

Рівень вологості також відіграє ключову роль у процесі терморегуляції. Підвищена вологість повітря ускладнює віддачу тепла організмом і уповільнює охолодження тіла. З фізіологічної точки зору оптимальними умовами для роботи є вологість у діапазоні 40–60%.

Усі вищезазначені параметри – температура, вологість, рух повітря та інші характеристики – визначають так званий мікроклімат приміщення. Для приміщень, у яких здійснюється робота з електронно-обчислювальними машинами, встановлені відповідні норми мікроклімату, які наведені у таблиці 6.2

Таблиця 6.2 – Норми мікроклімату для приміщень з ЕОМ

| **Пора року** | **Категорія робіт** | **Температура, °С** | **Відносна вологість, %** | **Швидкість повітря, м/с** |
|---|---|---|---|---|
| Холодна | Легка – 1а | 22-24 | 40-60 | 0,1 |
| Холодна | Легка – 1б | 21-23 | 40-60 | 0,1 |
| Тепла | Легка – 1а | 23-25 | 40-60 | 0,1 |
| Тепла | Легка – 1б | 22-24 | 40-60 | 0,2 |

Як зазначено раніше, недотримання мікрокліматичних параметрів призводить до теплового перенавантаження або переохолодження організму, що активує адаптаційні функції і негативно впливає на роботу серцево-судинної та дихальної систем.

Електричний струм належить до особливо небезпечних виробничих факторів, оскільки його дія є раптовою і не завжди передбачуваною. Людина може

відчути дію змінного струму силою від 0,6 до 1,5 мА та постійного струму від 5 до 7 мА. Високі показники можуть викликати невідпускаючий ефект та серйозні ураження.

Для запобігання електротравматизму необхідно:

– використовувати заземлення обладнання;

– застосовувати пристрої захисного відключення (ПЗВ);

– регулярно перевіряти електричну ізоляцію проводів та пристроїв.

Таким чином, основними шкідливими факторами в офісних приміщеннях з ЕОМ є шум, освітлення, мікроклімат та небезпека електричного струму. Комплексний підхід до їх усунення забезпечує створення безпечних і комфортних умов праці, що позитивно впливає на продуктивність і здоров’я працівників.

## 6.2 **Заходи щодо зменшення впливу шкідливих факторів на працівника**

Шкідливі виробничі фактори, що виникають в офісних приміщеннях з електронно-обчислювальними машинами (ЕОМ), суттєво впливають на здоров'я та працездатність працівників. Для мінімізації їхнього негативного впливу необхідно впроваджувати комплекс заходів, що охоплюють технічні, організаційні, санітарно-гігієнічні та психофізіологічні аспекти.

**До технічних заходів** варто віднести такі, що спрямовані на усунення або значне зниження впливу фізичних чинників, таких як шум, освітлення, електромагнітні випромінювання та параметри мікроклімату:

– оптимізація системи освітлення відповідно до санітарно-гігієнічних норм (300–500 лк на робочій поверхні);

– використання люмінесцентних ламп із спектром, наближеним до природного світла;

– уникнення відблисків на екрані шляхом встановлення антивідблискових фільтрів та застосування матових поверхонь на меблях і обладнанні;

– максимальне використання природного освітлення, правильне

розташування робочих місць відносно вікон;

– звукоізоляція приміщень, використання шумоізоляційних матеріалів для стін, стель та підлог;

– застосування глушників та амортизаторів для обладнання, що генерує високий рівень шуму;

– встановлення комп'ютерів та периферійних пристроїв у спеціально обладнаних технічних приміщеннях або використання тихих систем охолодження (вентиляторів та блоків живлення);

– впровадження систем вентиляції та кондиціювання, що забезпечують підтримку температури повітря у межах 22–24°C у холодний період та 23–25°C у теплий;

– контроль за рівнем вологості на рівні 40–60% за допомогою зволожувачів або осушувачів повітря;

– зниження швидкості руху повітря до нормативних значень 0,1–0,2 м/с шляхом налаштування вентиляційних систем;

– використання сучасних ЕОМ із низьким рівнем випромінювання, що відповідають міжнародним стандартам безпеки;

– встановлення робочих місць на безпечній відстані від джерел електромагнітних полів (мінімум 50–70 см від моніторів);

– застосування заземлення обладнання та спеціальних екранів, що блокують випромінювання;

**Організаційні заходи** полягають у впровадженні ефективної організації робочого процесу для зменшення навантаження на працівників.

Спеціалісти вважають, що найбільш оптимальне положення для роботи за комп'ютером – це вертикальне або з легким нахилом. Крісло також може мати незначний нахил вперед. Організація робочого простору повинна враховувати відстань між робочими місцями: вона має становити не менше 2 метрів між моніторами в лінії зору та не менше 1,2 метра між їхніми бічними сторонами. Під робочим столом необхідно передбачити достатньо місця для ніг: висота має бути не меншою за 600 мм, ширина – 500 мм, глибина на рівні колін – мінімум 450 мм,

а на рівні витягнутих ніг – не менш як 650 мм.

Положення тіла часто відповідає напряму погляду, тому екрани, що розташовані занадто низько або під невідповідним кутом, можуть спричиняти сутулість. Оптимальна відстань від очей до екрана залежить від виду роботи і коливається в межах 40-70 см. Під час читання текстів рекомендується, щоб відстань до монітора була трохи більшою, ніж при читанні книги, приблизно 40-45 см. Очам слід періодично надавати відпочинок, заплющуючи їх на кілька секунд. Необхідно уникати фіксації погляду на одному рівні та користуватися великим шрифтом для текстів. Важливо, щоб монітор не мерехтів: частота оновлення повинна бути від 72 Гц і вище. Людям із чутливим зором бажано використовувати екрани з частотою оновлення від 85 Гц.

Не слід розташовувати монітор паралельно до вікна, оскільки інтенсивне світло може заважати зору та призводити до спотворення зображення. Якщо робоче місце знаходиться поруч із вікном, краще сидіти під прямим кутом до нього, щоб уникнути відблисків. Для зменшення низькочастотного електромагнітного випромінювання слід обирати монітори, що відповідають стандартам MPR II або TCO 92-95. За відсутності цих характеристик варто встановити захисний екран, який затримує до 99% випромінювання.

Крісло повинно підтримувати природну форму спини користувача. Важливо підібрати висоту крісла так, щоб уникнути надмірного тиску на стегна. Кут між спиною та стегнами зазвичай трохи більше 90°, оскільки більшість людей комфортніше сидять у злегка відхиленому положенні.

Клавіатура має бути розташована так, щоб не потрібно було тягнутися або нахилятися вперед. При зміні позиції тіла слід адаптувати положення як клавіатури, так і монітора. Допомогти може підставка, яка дозволяє працювати з мишею без зайвого напруження. Під час роботи руки повинні бути прямими в зап'ястях і зігнутими під кутом 90° у ліктях, а пальці – трохи зігнутими.

Слід обирати стіл такої висоти, яка забезпечує зручність користування клавіатурою. Якщо стіл занадто високий, можна підняти крісло або використовувати підставку для ніг. Якщо стіл занизький, його висоту можна

збільшити за допомогою підкладок під ніжки.

Щоб уникнути втоми, необхідно чергувати роботу за комп'ютером із відпочинком. Якщо робочий режим не дозволяє чергування, слід робити перерви кожні 45-60 хвилин тривалістю 10-15 хвилин. У нічний час перерви можна подовжити на 30%.

Також слід дотримуватися рекомендації щодо зміни робочої пози для зменшення статичних навантажень на опорно-руховий апарат і використовувати вправи для очей, щоб зняти зорове напруження.

**Санітарно-гігієнічні** рішення спрямовані на підтримку чистоти та забезпечення комфортних умов у приміщенні.

– регулярне провітрювання приміщень для забезпечення свіжого повітря;

– прибирання приміщення з використанням вологого методу для зменшення кількості пилу, що може осідати на екранах та обладнанні;

– встановлення очисників повітря, особливо в приміщеннях з високою концентрацією техніки.

**Психофізіологічний підхід** допомагає мінімізувати стресові та фізичні навантаження на працівників.

Для зменшення нервово-психічного напруження, варто впроваджувати короткі релаксаційні вправи під час перерв, створити сприятливий мікроклімат та використовувати інструменти для автоматизації рутинних завдань.

Запровадження комплексу заходів щодо зменшення впливу шкідливих факторів забезпечить безпечні та комфортні умови для роботи в офісних приміщеннях з ЕОМ. Використання сучасних технічних рішень, раціональна організація робочих процесів і дотримання санітарно-гігієнічних вимог значно знижують ризики для здоров’я працівників, покращують їх самопочуття та підвищують ефективність праці.

## 6.3 **Висновки до розділу 6**

У цьому розділі було ознайомлено з ключовими елементами охорони праці.

Охорона праці є важливим аспектом забезпечення безпеки та збереження здоров'я працівників, що працюють в умовах використання електронно-обчислювальних машин (ЕОМ). Було проаналізовано основні шкідливі фактори, які впливають на працівників у офісному середовищі, серед яких особливу увагу приділено фізичним (освітлення, шум, мікроклімат, електромагнітне випромінювання) та психофізіологічним факторам (зорове навантаження, статичні пози, нервово-психічне перенапруження). Кожен із цих чинників має потенційний негативний вплив на здоров'я та працездатність людини, а їх тривалий вплив може спричинити професійні захворювання, зниження продуктивності праці та загальне погіршення самопочуття.

Важливим етапом у розділі стало формування комплексу заходів для мінімізації впливу шкідливих чинників. Технічні рішення спрямовані на оптимізацію освітлення, шумоізоляцію приміщень, поліпшення параметрів мікроклімату та зниження рівнів електромагнітного випромінювання. Організаційні заходи передбачають ефективне планування робочих місць, дотримання правил ергономіки, впровадження регулярних перерв та режимів праці, що сприяє зменшенню статичних і зорових навантажень. Санітарно-гігієнічні аспекти забезпечують підтримку чистоти робочих зон, регулярну циркуляцію повітря та оптимальні умови для роботи. Психофізіологічний підхід дозволяє мінімізувати стресові фактори за рахунок створення комфортного робочого середовища та підтримки здоров'я працівників.

Усі запропоновані заходи спрямовані на створення безпечного, здорового та продуктивного середовища для працівників, які працюють з ЕОМ. Впроваження комплексних рішень, допоможе самопочуттю працівника та покращить його ефективність, що у свою чергу, підвищує продуктивність праці, знижує ризики професійних захворювань та створює умови для стійкого функціонування організації в цілому.

# 7 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Охорона навколишнього середовища (ОНС) – це система заходів, спрямованих на збереження та відновлення природних ресурсів, забезпечення екологічної безпеки та підтримання гармонійної взаємодії між суспільством і природою. Вона охоплює економічні, правові, гігієнічні, науково-технічні, біологічні та інші аспекти [49].

Основними завданнями ОНС є:

– забезпечення чистоти природних екосистем, включаючи повітря, воду та ґрунти;

– раціональне використання та відтворення природних ресурсів;

– запобігання та контроль забруднення довкілля;

– розробка та впровадження екологічно чистих технологій;

– підвищення екологічної свідомості населення.

Забруднення навколишнього середовища є однією з найактуальніших екологічних проблем сучасності. Діяльність людини, зокрема промислове виробництво, транспорт, сільське господарство та побутове споживання, призводить до викидів шкідливих речовин у повітря, воду та ґрунт, що негативно впливає на екосистеми та здоров'я людей.

За порушення у сфері охорони природи, незаконне використання природних ресурсів законодавством України передбачена адміністративна та кримінальна відповідальність.

## 7.1 **Забруднення довколишнього середовища приміщеннями з ЕОМ**

Офісні приміщення з електронно-обчислювальними машинами (ЕОМ) також роблять свій внесок у забруднення довкілля.

Одним з джерел забруднення є високе енергоспоживання.

Робота комп'ютерів, серверів та іншого офісного обладнання потребує значної кількості електроенергії, виробництво якої часто супроводжується

викидами парникових газів та іншими забрудненнями. Використання енергоефективних пристроїв та впровадження політики енергозбереження можуть зменшити цей вплив.

Також, можливим фактором можуть бути відходи електронного обладнання, це застаріла та непрацююча техніка містить небезпечні речовини, такі як свинець, ртуть та кадмій, які при неправильній утилізації можуть потрапляти в навколишнє середовище, забруднюючи його.

Правильна утилізація та переробка електронних відходів є необхідною для зменшення цього ризику.

Надмірне споживання паперу в офісах, також є негативним чинником, так як призводить до вирубки лісів та збільшення кількості відходів. Перехід на електронний документообіг та використання переробленого паперу можуть знизити негативний вплив на довкілля.

Для зменшення забруднення варто впроваджувати спеціальні заходи.

Концепція «Зелений офіс», є доволі популярною у сьогоденні, вона полягає в Впровадження екологічно дружніх практик в офісах, таких як енергозбереження, сортування відходів та використання екологічно чистих матеріалів.

Взагалі, забезпечення правильного збору та переробки застарілої техніки для запобігання потраплянню токсичних речовин у довкілля. В тому числі встановлення спеціальних контейнерів для збору батарейок та передача їх на переробку.

Впровадження цих заходів сприятиме зменшенню негативного впливу офісів з ЕОМ на навколишнє середовище та підвищенню екологічної свідомості працівників.

## 7.2 **Висновки до розділу 7**

У розділі, присвяченому охороні навколишнього середовища, було детально проаналізовано вплив інформаційних технологій, зокрема персональних комп'ютерів, на екологію. Розглянуто негативні наслідки виробництва, експлуатації та утилізації комп'ютерної техніки, які призводять до забруднення

довкілля токсичними речовинами та електронними відходами.

Окрім цього, висвітлено сучасні підходи до зменшення екологічного впливу ІТ-обладнання, такі як впровадження "зелених" інформаційних технологій, що передбачають енергоефективне використання ресурсів та екологічно безпечну утилізацію пристроїв. Також, було розглянуто практичні заходи для раціональної утилізації використаного обладнання без шкоди для природи, включаючи переробку та повторне використання компонентів, що сприяє зменшенню обсягів електронних відходів та збереженню природних ресурсів.

Загалом, розділ підкреслює важливість усвідомлення екологічних аспектів використання інформаційних технологій та необхідність впровадження сталих практик у сфері ІТ для збереження навколишнього середовища.

# ВИСНОВКИ

В кваліфікаційній роботі було вирішено задачу підвищення достовірності оцінювання метрики RFC від CBO та WMC для десктоп застосунків мовою програмування C# для визначення їх якості. Основним результатом дослідження є створення двофакторної нелінійної регресії для оцінювання метрики RFC від CBO та WMC, а також реалізації ПЗ для визначення якості десктоп застосунків на C#.

Мета досягнута, були вирішенні наступні завдання дослідження.

Було проаналізовано сучасні методи та математичні моделі для оцінювання якості програмного забезпечення. Знайдено, що існуючі підходи потребують вдосконалення з огляду на підвищення достовірності аналізу якості десктоп застосунків, а також, відсутні рішення та роботи пов’язані з оцінкою якості для десктоп застосунків мовою програмування C#.

Для досягнення результатів, було використано метод нормалізуючого перетворення Бокса-Кокса.

Застосовано регуляризацію для зменшення мультиколінеарності, після чого проведено очистку значень метрик від викидів за допомогою квадрату відстані Махаланобіса.

Побудовано двофакторну нелінійну регресійну модель для аналізу залежної метрики RFC від WMC та CBO, а також висновку щодо якості ПЗ. Створена математичну модель, дозволяє підвищити достовірність оцінювання метрики RFC для визначення якості програмного забезпечення порівняно з існуючою моделлю для застосунків на Java.

Побудована модель має наступні показники якості:

– $R^2$ = 0,706;
– MMRE = 0,269;
– PRED(0,25) має результат 0,642.

Розроблено програмний продукт, який реалізує побудовану модель та визначає якість програмного забезпечення. Використання програми забезпечує

швидкий і точний аналіз якості десктоп застосунків за ключовими показниками RFC, CBO та WMC.

Тестування програмного забезпечення підтвердило його ефективність та надійність. Програма успішно обробляє вхідні дані, видаляє викиди, обчислює прогнози та візуалізує результати

Розроблене програмне забезпечення далі може використовуватись у процесі розробки десктоп застосунків для автоматичного оцінювання якості програмного коду, а результати дослідження можуть бути впроваджені в інженерну практику для визначення якості програмних продуктів, а також використані у навчальному процесі для підготовки спеціалістів з інженерії програмного забезпечення чи у майбутніх дослідженнях.

Для подальшого покращення та досліджень, можна розширити модель шляхом додавання нових метрик якості. Збору більшого та різноманітнішого набору вхідних даних для розробки математичної моделі.

Один з варіантів майбутніх досліджень, це впровадження алгоритмів машинного навчання для підвищення точності прогнозування.

Також була розрахована економічна ефективність, яка показала, що розроблене програмне забезпечення окупається протягом 4 місяців після впровадження, забезпечуючи економічну вигоду.

В роботі, розглянуті вимоги до охорони праці та довколишнього середовища.

# СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team / R. Moser et al. Balancing Agility and Formalism in Software Engineering. Berlin, Heidelberg, 2008. P. 252–266. URL: https://doi.org/10.1007/978-3-540-85279-7_20 (date of access: 18.10.2024).

2. ISO/IEC 25010:2011. ISO. URL: https://www.iso.org/standard/35733.html (date of access: 18.10.2024).

3. Yenduri G., Gadekallu T. R. A review on soft computing approaches for predicting maintainability of software: State-of-the-art, technical challenges, and future directions. Expert Systems. 2023. URL: https://doi.org/10.1111/exsy.13250 (date of access: 18.10.2024).

4. 软件设计：推荐ISO/IEC 25010:2023标准，软件质量框架与设计指南-腾讯云开发者社区-腾讯云. 腾讯云 产业智变·云**启未来** - 腾讯. URL: https://cloud.tencent.com/developer/article/2426708 (date of access: 18.10.2024).

5. Yuniasri D., Rochimah S., Raharjo A. B. A Correlation Analysis Between ISO 25010 based Modularity and CK Metrics in Object-Oriented Software. 2020 International Conference on Advanced Science and Engineering (ICOASE), Duhok, Iraq, 23–24 December 2020. 2020. URL: https://doi.org/10.1109/icoase51841.2020.9436617 (date of access: 18.12.2024).

6. Racim B. Analyzing CK Metrics Results and Quality Assessment. Medium. URL: https://medium.com/@benkaddourmed54/analyzing-ck-metrics-results-and-quality-assessment-a70ba56534f0 (date of access: 18.10.2024).

7. Кларк Е. Кореляція в R: кореляційна матриця Пірсона та Спірмена. Guru99. URL: https://www.guru99.com/uk/r-pearson-spearman-correlation.html (дата звернення: 18.10.2024).

8. Prykhodko S., Prykhodko N. Estimating Quality of Open-Source Kotlin-Based Apps by the Confidence and Prediction Intervals of Nonlinear Regression for RFC Metric. 2023 IEEE 18th International Conference on Computer Science and Information

Technologies (CSIT), Lviv, Ukraine, 19–21 October 2023. 2023. URL: https://doi.org/10.1109/csit61576.2023.10324187 (date of access: 18.10.2024).

9. Prykhodko S., Prykhodko N. A Technique for Detecting Software Quality Based on the Confidence and Prediction Intervals of Nonlinear Regression for RFC Metric. 2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT), Lviv, Ukraine, 10–12 November 2022. 2022. URL: https://doi.org/10.1109/csit56902.2022.10000532 (date of access: 18.10.2024).

10. Pfeiffer R.-H. The Impact of Continuous Code Quality Assessment on Defects. 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), Luxembourg, 27 September – 1 October 2021. 2021. URL: https://doi.org/10.1109/icsme52107.2021.00069 (date of access: 18.10.2024).

11. Elish M. O., Rine D. Investigation of metrics for object-oriented design logical stability. Seventh European Conference on Software Maintenance and Reengineering, Benevento, Italy. URL: https://doi.org/10.1109/csmr.2003.1192427 (date of access: 18.10.2024).

12. An Empirical study of Exploring Relevant Metrics to Assess Software Product Quality / Z. Song et al. 2020 7th International Conference on Dependable Systems and Their Applications (DSA), Xi'an, China, 28–29 November 2020. 2020. URL: https://doi.org/10.1109/dsa51864.2020.00023 (date of access: 18.10.2024).

13. Wright G. What is empirical analysis and how does it work?. WhatIs. URL: https://www.techtarget.com/whatis/definition/empirical-analysis (date of access: 21.10.2024).

14. Hrytsiuk Y. I., Ferneza O. R. Reflection of expert-based evaluation of software quality. Scientific Bulletin of UNFU. 2019. Vol. 29, no. 8. P. 152–158. URL: https://doi.org/10.36930/40290828 (date of access: 21.10.2024).

15. Lounis H., Gayed T. F., Boukadoum M. Machine-Learning Models for Software Quality: A Compromise between Performance and Intelligibility. 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence (ICTAI), Boca Raton, FL, USA, 7–9 November 2011. 2011. URL: https://doi.org/10.1109/ictai.2011.155 (date of access: 21.10.2024).

16. Gardini Miguel P. 20 Best Code Analysis Tools in 2024. The CTO Club. URL: https://thectoclub.com/tools/best-code-analysis-tools/ (date of access: 21.10.2024).

17. Plans & Pricing. Better Code & Better Software | Ultimate Security and Quality | Sonar. URL: https://www.sonarsource.com/plans-and-pricing/sonarqube/ (date of access: 21.10.2024).

18. Analyzing a Project with SonarQube - Amplify DX Documentation. Home - Amplify DX Documentation. URL: https://dx.appirio.com/quality-sonarqube/sonarqube-project-analysis/ (date of access: 21.10.2024).

19. Defect Density – Learn with examples. Best Test Management Software Tool for QA Testing – Tuskr™. URL: https://tuskr.app/learn/defect-density (date of access: 21.10.2024).

20. Sarkar S., Kak A. C., Rama G. M. Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software. IEEE Transactions on Software Engineering. 2008. Vol. 34, no. 5. P. 700–720. URL: https://doi.org/10.1109/tse.2008.43 (date of access: 21.10.2024).

21. Chidamber S. R., Kemerer C. F. A metrics suite for object oriented design. IEEE Transactions on Software Engineering. 1994. Vol. 20, no. 6. P. 476–493. URL: https://doi.org/10.1109/32.295895 (date of access: 17.12.2024).

22. Chidamber & Kemerer. Project Metrics Help - Chidamber & Kemerer object-oriented metrics suite. Aivosto. Analyze, Document and Flowchart Source Code. URL: https://www.aivosto.com/project/help/pm-oo-ck.html (date of access: 21.10.2024).

23. Simple Linear Regression. Fachbereich Geowissenschaften: Startseite. URL: https://www.geo.fu-berlin.de/en/v/soga-r/Basics-of-statistics/Linear-Regression/Simple-Linear-Regression/index.html (date of access: 22.10.2024).

24. Приходько С. Б., Макарова Л. М., Приходько Н. В., Пухалевич А. В. Методичні вказівки та завдання до виконання лабораторних робіт з дисципліни «Емпіричні методи програмної інженерії». Миколаїв: НУК, 2020. 60 с.

25. Fatima N. Understanding the Box-Cox Power Transformer. Medium. URL: https://medium.com/@noorfatimaafzalbutt/understanding-the-box-cox-power-transformer-3dbb6613a593 (date of access: 25.10.2024).

26. Dzingwa K. Bayesian Statistics: A Flexible and Principled Approach to Statistical Inference. Medium. URL: https://medium.com/@lomso.dzingwa/bayesian-statistics-a-flexible-and-principled-approach-to-statistical-inference-7f4d28bbc42c (date of access: 25.10.2024).

27. Getting Started. Getting Started with NDepend on Windows. URL: https://www.ndepend.com/docs/getting-started-with-ndepend (date of access: 26.10.2024).

28. Hayes A. Multicollinearity: Meaning, Examples, and FAQs. Investopedia. URL: https://www.investopedia.com/terms/m/multicollinearity.asp (date of access: 01.11.2024).

29. msoczi. Ridge Regression: Step by step introduction with example. Medium. URL: https://medium.com/@msoczi/ridge-regression-step-by-step-introduction-with-example-0d22dddb7d54 (date of access: 01.11.2024).

30. Mahalanobis Distance - Understanding the math with examples (python) - Machine Learning Plus. Machine Learning Plus. URL: https://www.machinelearningplus.com/statistics/mahalanobis-distance/ (date of access: 26.10.2024).

31. Prykhodko S. Evaluating Quality of Software Systems by the Confidence and Prediction Intervals of Regressions for RFC, CBO and WMC Metrics. WSEAS TRANSACTIONS ON SYSTEMS. 2024. Vol. 23. P. 322–330. URL: https://doi.org/10.37394/23202.2024.23.36 (date of access: 02.11.2024).

32. Shaun T. Coefficient of Determination ($R^2$) | Calculation & Interpretation. DataScience StackExchange. URL: https://www.scribbr.com/statistics/coefficient-of-determination/ (date of access: 26.10.2024).

33. Jørgensen M., Halkjelsvik T., Liestøl K. When should we (not) use the mean magnitude of relative error (MMRE) as an error measure in software development effort estimation?. Information and Software Technology. 2022. Vol. 143. P. 106784. URL: https://doi.org/10.1016/j.infsof.2021.106784 (date of access: 28.10.2024).

34. Методичні вказівки та завдання до виконання лабораторних робіт з дисципліни «Емпіричні методи програмної інженерії» / С. Б. Приходько, Л. М. Макарова, Н. В. Приходько, А. В. Пухалевич. – Миколаїв: НУК, 2023. – 56 с.

35. Позняков Р.О. Удосконалення регресійної моделі для оцінювання метрики RFC застосунків з відкритим кодом на Java та розробка програми для її реалізації / Р. О. Позняков ; наук. кер. С. Б. Приходько. – Миколаїв : НУК, 2022. – 108 с.

36. joblib.dump – joblib 1.4.2 documentation. Joblib: running Python functions as pipeline jobs – joblib 1.4.2 documentation. URL: https://joblib.readthedocs.io/en/stable/generated/joblib.dump.html (date of access: 15.11.2024).

37. Streamlit Docs. Streamlit documentation. URL: https://docs.streamlit.io/ (date of access: 16.11.2024).

38. fill_between(x, y1, y2) – Matplotlib 3.9.3 documentation. Matplotlib – Visualization with Python. URL: https://matplotlib.org/stable/plot_types/basic/fill_between.html#sphx-glr-plot-types-basic-fill-between-py (date of access: 20.11.2024).

39. What is Unified Modeling Language (UML)?. Ideal Modeling & Diagramming Tool for Agile Team Collaboration. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ (date of access: 20.11.2024).

40. Use case diagrams are UML diagrams describing units of useful functionality (use cases) performed by a system in collaboration with external users (actors). URL: https://www.uml-diagrams.org/use-case-diagrams.html (date of access: 23.11.2024).

41. Каграманова Ю. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. Dou. URL: https://dou.ua/forums/topic/40575/ (Дата звернення: 23.11.2024).

42. Activity Diagrams - Unified Modeling Language (UML) - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/ (date of access: 23.11.2024).

43. Puchkov D. UI Development Process: All You Should Know in 2024. Software Development Blog & IT Tech Insights | Django Stars. URL: https://djangostars.com/blog/ui-development-flow/ (date of access: 26.11.2024).

44. Joyshree. Foundation of User Interface and User Experience Series 02: The 6 Stages of UI Design. Medium. URL: https://medium.com/@shreejsk21/foundation-of-user-interface-and-user-experience-series-02-the-6-stages-of-ui-design-01ecde278f59 (date of access: 27.11.2024).

45. Stanislav K. UI Design Process: A Step-by-Step Guide for Designers. Medium. URL: https://medium.com/glow-team/ui-design-process-a-step-by-step-guide-for-designers-6cb806f87cfc (date of access: 27.11.2024).

46. Singh A. P. UML Class Diagram Explained with Examples. AlgoMaster Newsletter | Ashish Pratap Singh | Substack. URL: https://blog.algomaster.io/p/uml-class-diagram-explained-with-examples (date of access: 27.11.2024).

47. Sequence Diagram Tutorial â□□ Complete Guide with Examples | Creately. Creately. URL: https://creately.com/guides/sequence-diagram-tutorial/ (date of access: 27.11.2024).

48. Про охорону праці. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 01.12.2024).

49. Про охорону навколишнього природного середовища. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/1264-12#Text (дата звернення: 04.12.2024).

# ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ

Вступ

Повне найменування теми розробки: програмне забезпечення для оцінювання якості десктоп застосунків на C#.

Коротке найменування програми: Desktop C# Quality Calculator.

Коротка характеристика галузі застосування: програмне забезпечення застосовується для автоматизації обчислень з оцінювання десктоп застосунків, що створюються з використанням мови програмування C#, яке дозволяє підвищити ефективність, точність та зменшити витрати часу для аналітиків, менеджерів та інженерів, які займаються проєктуванням там розробкою програмного забезпечення.

1. Підстави для розробки

Розробка виконується на підставі завдання на кваліфікаційну роботу, що видане кафедрою ПЗАС ННІКНУП НУК ім. адмірала Макарова.

2. Призначення розробки

2.1 Функціональне призначення програми

Функціональним призначення ПЗ є автоматизація процесів:

– вводу статичних даних метрик;

– нормалізація вхідних вибірок за допомогою перетворення Бокса-Кокса та позбавлення викидів за допомогою квадрату відстані Махаланобіса;

– побудова математичної моделі;

– розрахунок метрики RFC від CBO та WMC;

– обчислення довірчого інтервалу та інтервалу прогнозування;

– надання рекомендацій, щодо покращення якості ПЗ;

– надання інтерпретації якості ПЗ на основі значення RFC.

2.2 Експлуатаційне призначення програми

Експлуатаційним призначенням даного програмного продукту є покращення та легкість обробки статичних даних про якість ПЗ, підвищення достовірності та полегшення оцінювання якості ПЗ десктоп застосунків на C#.

3. Вимоги до програмного забезпечення

3.1 Вимоги до функціональних характеристик

3.1.1 Вимоги до складу виконуваних функцій

Програмне забезпечення повинно виконувати наступні функції:

– можливість завантажити файл з вихідними даними метрик у форматі CSV;
– можливість вручну ввести значення метрик у відповідні поля;
– можливість замінити файл з вихідними емпіричними даними;
– розрахунок параметрів для вихідних вибірок емпіричних даних;
– перевірка вихідних вибірок даних на нормальність закону розподілу;
– нормалізація емпіричних даних за допомогою перетворення Бокса-Кокса;
– перевірка вибірок даних на наявність викидів за допомогою квадрату відстані Махаланобіса;
– оцінювання якості ПЗ за допомогою побудованої математичної моделі;
– обчислення довірчого інтервалу;
– обчислення інтервалу прогнозування;
– надання рекомендацій, щодо покращення ПЗ;
– експорт результатів у файл у форматі CSV.

3.1.2 Вимоги до організації вхідних та вихідних даних

Введення даних здійснюється за допомогою пристроїв введення даних (клавіатура або миша). Дані вводяться через завантаження файлу у вікні програми або за допомогою ручного вводу у відповідні поля.

Виведення даних здійснюється за допомогою пристрою виведення даних – монітору комп’ютера.

Вхідні дані:

- файл у форматі CSV який містить стовпці CBO, RFC, WMC;
- чисельні значення.

Вихідні дані:

- нормалізована вибірка даних у таблиці;
- інтерпретація якості ПЗ.
- значення залежної метрики RFC від CBO та WMC;
- розраховані довірчий інтервал та інтервал прогнозування

## 3.2 Вимоги до надійності

Програмний продукт повинний правильно обробляти вхідні дані та сигналізувати про помилки форматування чи відсутність потрібних стовпчиків у завантаженому CSV. При відсутності моделі або пошкодженому файлі моделі система має виводити відповідне повідомлення про помилку.

Повинна бути передбачена стійкість до некоректного вводу користувача (наприклад, виведення помилки, якщо введено від’ємні значення або дані поза очікуваним діапазоном).

## 3.3 Вимоги до умов експлуатації

Необхідний рівень підготовки користувачів для використання ПЗ: мінімальні навички в користуванні комп’ютером. Комп’ютер призначений для роботи в закритому опалювальному приміщені при наступних умовах навколишнього середовища:

- температура повітря від +10оС до +30оС;
- вологість повітря не більше 80;
- запиленість повітря не більше 0,75 мг/м2.

## 3.4 Вимоги до складу та параметрів технічних засобів

В склад технічних засобів повинен входити сервер, якщо застосунок

запускається локально, з наступними параметрами:

– процесор: Intel Celeron G4900, 3.1 GHZ, 2 ядра та більше;

– інтернет: 30 Мбит/с та більше.

– встановлений інтерпретатор Python 3.x, а також можливість інсталяції необхідних бібліотек через pip, якщо система запускається локально;

– оперативна пам’ять: мінімум 4Гб.

– жорсткий диск: для нормальної роботи системі потрібно 50Мб вільного дискового простору.

В склад технічних засобів користувача повинен входити комп’ютер, з мінімальними характеристиками, який підтримує встановлення програмного додатку Google Chrome. Робоча станція має мати наступні характеристики:

– мінімум двохядерний процесор з тактовою частотою 1.8 ГГц;

– оперативна пам’ять (RAM) мінімум 2 ГБ;

– жорсткий диск з достатньою кількістю вільного місця для зберігання результатів (не менше 500 Мб);

– монітор з роздільною здатністю не нижче 1280x720 для комфортної роботи з графічним інтерфейсом застосунку.

– інтернет: 30 Мбит/с та більше.

## 3.5 Вимоги до інформаційної та програмної сумісності

Для повноцінного функціонування системи необхідно використовувати:

– На сервері повинна бути встановлена ОС Linux

– На робочій станції повинна бути встановлена ОС (Windows 10+, Linux, macOS) та підтримкою сучасного веб-браузера.

## 3.6 Вимоги до маркування та пакування

Поширення програмного продукту на фізичних носіях не передбачається.

## 3.7 Вимоги до транспортування та зберігання

Вимоги до транспортування не висовуються у зв’язку з відсутністю фізичних носів.

4. Вимоги до програмної документації ,

Програмна документація повинна містити:

- технічне завдання;
- текст програми;
- опис програми;
- інструкцію користувача;
- програму і методику випробувань ПЗ.

5. Техніко-економічні показники

Економічна ефективність впровадження даного програмного забезпечення розрахована у розділі 5. Згідно з отриманими результатами обчислення, розробка, впровадження та експлуатація даного ПЗ є доцільною, а термін окупності становить 4 місяців. Тобто, через 4 місяці після впровадження програмного забезпечення всі витрати вкладені у його розробку та впровадження, повністю компенсуються за рахунок отриманої вигоди.

6. Стадії та етапи розробки

Стадії та етапи розробки представлені в таблиці А.1.

Таблиця А.1 – Стадії та етапи розробки проєкту

| **№** | **Стадії розробки ПЗ** | **Етапи роботи** | **Терміни виконання робіт** | |
|---|---|---|---|---|
| | | | **Початок** | **Кінець** |
| **1** | **2** | **3** | **4** | **5** |
| 1 | Технічне завдання | Обґрунтування необхідності розробки ПЗ | 08.09.24 | 11.09.24 |
| | | Постановка завдання | 11.09.24 | 15.09.24 |
| | | Затвердження технічних вимог | 15.09.24 | 25.09.24 |
| 2 | Ескізний проєкт | Розробка ескізного проєкту | 25.09.24 | 08.10.24 |
| | | Затвердження проєкту | 08.10.24 | 15.10.24 |

Продовження таблиці А.1

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 3 | Технічний проєкт | Проєктування | 15.10.24 | 25.10.24 |
| | | Розробка технічного проєкту | 25.10.24 | 01.11.24 |
| | | Затвердження технічного проєкту | 01.11.24 | 07.11.24 |
| 4 | Робочий проєкт | Розробка проєкту | 07.11.24 | 24.11.24 |
| | | Затвердження робочого проєкту | 24.11.24 | 26.11.24 |
| | | Розробка документації | 26.11.24 | 28.11.24 |

7. Порядок контролю та прийому

Контроль процесу аналізу та проєктування кожного компонента програмного забезпечення для оцінювання якості десктоп застосунків на C# здійснюється на всіх етапах розробки з урахуванням вимог, визначених у технічному завданні. Кожен етап створення програмного продукту має бути завершений у встановлені терміни та узгоджений із замовником.

Приймання готового програмного продукту оформлюється протоколом випробувань. Якщо під час перевірки виявлено помилки, складається акт із переліком дефектів, який підписується представниками замовника та розробника, а також затверджується керівництвом обох сторін. Розробник зобов'язаний усунути всі зазначені помилки протягом двох тижнів і повідомити замовника про готовність до повторного тестування.

# ДОДАТОК Б – ТЕКСТ ПРОГРАМИ

## Файл main.py

```
import streamlit as st
import pandas as pd
import io

from data_normalizer import DataNormalizer
from outlier_remover import OutlierRemover
from interval_calculator import IntervalCalculator
from model_predictor import ModelPredictor
from plotter import Plotter
from recommendations import Recommendations


def main():
    st.title("Оцінка якості програмного забезпечення")
    st.write("Введіть значення метрик CBO та WMC або завантажте CSV файл.")

    try:
        model_predictor = ModelPredictor("model_pipeline.pkl")
    except Exception as e:
        st.error(f"Не вдалося завантажити модель: {e}")
        return

    try:
        normalizer = DataNormalizer(lambda_values=model_predictor.lambda_values)
    except Exception as e:
        st.error(f"Не вдалося ініціалізувати DataNormalizer: {e}")
        return

    outlier_remover = OutlierRemover()
    interval_calculator = IntervalCalculator()
    plotter = Plotter()

    st.sidebar.title("Завантаження даних")
    uploaded_file = st.sidebar.file_uploader("Завантажте CSV з колонками CBO та WMC:",
type=["csv"])

    if uploaded_file is not None:
        try:
            data = pd.read_csv(uploaded_file)
            if data.empty:
                st.sidebar.error("Завантажений файл порожній. Будь ласка, завантажте файл з
даними.")
                return
            if not all(col in data.columns for col in ["CBO", "WMC"]):
                missing_cols = [col for col in ["CBO", "WMC"] if col not in data.columns]
                st.sidebar.error(
                    f"CSV файл повинен містити стовпчики: CBO та WMC. Відсутні: {',
'.join(missing_cols)}.")
                return
        except Exception as e:
            st.sidebar.error(f"Не вдалося завантажити файл: {e}")
            return

        try:
            data_normalized = normalizer.normalize_dataframe(data, ['CBO', 'WMC'])
        except ValueError as ve:
            st.error(f"Помилка нормалізації даних: {ve}")
            return
```

```
                try:
                    data_cleaned, removed_count =
outlier_remover.remove_outliers_iqr(data_normalized, ["CBO", "WMC"])
                except Exception as e:
                    st.error(f"Помилка при видаленні викидів: {e}")
                    return

                st.subheader("Таблиця завантажених та нормалізованих даних (після видалення
викидів)")
                if removed_count > 0:
                    st.warning(f"Було видалено {removed_count} рядків з викидами.")
                st.write(data_cleaned.head())

                if len(data_cleaned) > 0:
                    max_index = len(data_cleaned) - 1
                    selected_index = st.number_input("Оберіть індекс рядка для прогнозу з таблиці:",
                                                     min_value=0, max_value=max_index, step=1)
                    if st.button("Прогноз для обраного рядка"):
                        try:
                            cbo_row = data_cleaned.iloc[selected_index]["CBO"]
                            wmc_row = data_cleaned.iloc[selected_index]["WMC"]
                            cbo_orig = normalizer.denormalize_value(cbo_row, 'CBO')
                            wmc_orig = normalizer.denormalize_value(wmc_row, 'WMC')
                        except Exception as e:
                            st.error(f"Помилка денормалізації даних: {e}")
                            return

                        try:
                            result = model_predictor.predict_quality(cbo_orig, wmc_orig, normalizer)
                        except Exception as e:
                            st.error(f"Помилка прогнозування: {e}")
                            return

                        st.success(f"**Розрахована якість ПЗ для рядка {selected_index}:**
{result:.2f}")
                        quality = model_predictor.interpret_quality(result)
                        st.info(f"**Інтерпретація:** {quality}")

                        try:
                            ci_lower, ci_upper = interval_calculator.bootstrap_confidence_interval(
                                lambda c, w: model_predictor.predict_quality(c, w, normalizer),
                                cbo_orig, wmc_orig
                            )
                            pi_lower, pi_upper = interval_calculator.bootstrap_prediction_interval(
                                lambda c, w: model_predictor.predict_quality(c, w, normalizer),
                                cbo_orig, wmc_orig
                            )
                        except Exception as e:
                            st.error(f"Помилка при розрахунку інтервалів: {e}")
                            return

                        st.write(f"**Довірчий інтервал (95%):** [{ci_lower:.2f}, {ci_upper:.2f}]")
                        st.write(f"**Прогнозний інтервал (95%):** [{pi_lower:.2f}, {pi_upper:.2f}]")

                        Recommendations.provide(quality)

                        try:
                            results_df = pd.DataFrame([{
                                "CBO": cbo_orig,
                                "WMC": wmc_orig,
                                "Predicted_Quality": result,
                                "Interpretation": quality,
                                "CI_Lower": ci_lower,
                                "CI_Upper": ci_upper,
                                "PI_Lower": pi_lower,
                                "PI_Upper": pi_upper
                            }])
                            csv_buffer = io.StringIO()
                            results_df.to_csv(csv_buffer, index=False)
```

```
                    st.download_button(
                        label="Завантажити результати в CSV",
                        data=csv_buffer.getvalue(),
                        file_name="results.csv",
                        mime="text/csv"
                    )
                except Exception as e:
                    st.error(f"Помилка при збереженні результатів: {e}")

        else:
            st.warning("Після видалення викидів дані порожні. Завантажте інший файл.")

    st.write("---")
    st.write("Ви можете вручну ввести значення CBO та WMC")

    cbo = st.number_input("Введіть значення CBO", min_value=0.0, format="%.2f")
    wmc = st.number_input("Введіть значення WMC", min_value=0.0, format="%.2f")

    if st.button("Розрахувати якість ПЗ (вручну)"):
        if cbo <= 0 or wmc <= 0:
            st.error("Box-Cox трансформація вимагає позитивних значень для всіх метрик.")
        else:
            try:
                result = model_predictor.predict_quality(cbo, wmc, normalizer)
            except Exception as e:
                st.error(f"Помилка прогнозування: {e}")
                return

            st.success(f"**Розрахована якість ПЗ:** {result:.2f}")
            quality = model_predictor.interpret_quality(result)
            st.info(f"**Інтерпретація:** {quality}")

            try:
                ci_lower, ci_upper = interval_calculator.bootstrap_confidence_interval(
                    lambda c, w: model_predictor.predict_quality(c, w, normalizer),
                    cbo, wmc
                )
                pi_lower, pi_upper = interval_calculator.bootstrap_prediction_interval(
                    lambda c, w: model_predictor.predict_quality(c, w, normalizer),
                    cbo, wmc
                )
            except Exception as e:
                st.error(f"Помилка при розрахунку інтервалів: {e}")
                return

            st.write(f"**Довірчий інтервал (95%):** [{ci_lower:.2f}, {ci_upper:.2f}]")
            st.write(f"**Прогнозний інтервал (95%):** [{pi_lower:.2f}, {pi_upper:.2f}]")
            Recommendations.provide(quality)

            try:
                results_df = pd.DataFrame([{
                    "CBO": cbo,
                    "WMC": wmc,
                    "Predicted_Quality": result,
                    "Interpretation": quality,
                    "CI_Lower": ci_lower,
                    "CI_Upper": ci_upper,
                    "PI_Lower": pi_lower,
                    "PI_Upper": pi_upper
                }])
                csv_buffer = io.StringIO()
                results_df.to_csv(csv_buffer, index=False)
                st.download_button(
                    label="Завантажити результати в CSV",
                    data=csv_buffer.getvalue(),
                    file_name="results_manual.csv",
                    mime="text/csv"
                )
            except Exception as e:
```

```
                    st.error(f"Помилка при збереженні результатів: {e}")


if __name__ == "__main__":
    main()
```

## Файл data_normalizer.py

```
import numpy as np
import pandas as pd
from scipy.stats import boxcox

class DataNormalizer:
    def __init__(self, lambda_values=None):
        self.lambda_values = lambda_values or {}

    def normalize_dataframe(self, df: pd.DataFrame, columns: list):
        df_norm = df.copy()
        for col in columns:
            if (df_norm[col] <= 0).any():
                raise ValueError(f"Box-Cox трансформація вимагає позитивних значень для
колонки {col}.")
            df_norm[col], lambda_val = boxcox(df_norm[col])
            self.lambda_values[col] = lambda_val
        return df_norm

    def normalize_value(self, value, column):
        if column not in self.lambda_values:
            raise ValueError(f"Lambda value for column '{column}' is not available.")
        lambda_val = self.lambda_values[column]
        if lambda_val != 0:
            return (value ** lambda_val - 1) / lambda_val
        else:
            return np.log(value)

    def denormalize_value(self, value, column):
        if column not in self.lambda_values:
            raise ValueError(f"Lambda value for column '{column}' is not available.")
        lambda_val = self.lambda_values[column]
        if lambda_val != 0:
            return (value * lambda_val + 1) ** (1 / lambda_val)
        else:
            return np.exp(value)

    def denormalize_dataframe(self, df: pd.DataFrame, columns: list):
        df_denorm = df.copy()
        for col in columns:
            if col not in self.lambda_values:
                raise ValueError(f"Lambda value for column '{col}' is not available.")
            df_denorm[col] = df_denorm[col].apply(lambda x: self.denormalize_value(x, col))
        return df_denorm

    def get_lambda_values(self):
        return self.lambda_values
```

## Файл interval_calculator.py

```
import numpy as np

class IntervalCalculator:
    def __init__(self, n_samples=1000, alpha=0.05):
        self.n_samples = n_samples
        self.alpha = alpha

    def bootstrap_confidence_interval(self, prediction_func, cbo, wmc):
        preds = []
```

```
        for _ in range(self.n_samples):
            cbo_ = max(0, cbo + np.random.normal(0, 0.5))
            wmc_ = max(0, wmc + np.random.normal(0, 0.5))
            preds.append(prediction_func(cbo_, wmc_))
        lower = np.percentile(preds, 100 * self.alpha / 2)
        upper = np.percentile(preds, 100 * (1 - self.alpha / 2))
        return lower, upper

    def bootstrap_prediction_interval(self, prediction_func, cbo, wmc):
        preds = []
        for _ in range(self.n_samples):
            cbo_ = max(0, cbo + np.random.normal(0, 1.0))
            wmc_ = max(0, wmc + np.random.normal(0, 1.0))
            preds.append(prediction_func(cbo_, wmc_))
        lower = np.percentile(preds, 100 * self.alpha / 2)
        upper = np.percentile(preds, 100 * (1 - self.alpha / 2))
        return lower, upper
```

## Файл outlier_remover.py

```
import pandas as pd
from scipy.spatial.distance import mahalanobis
from sklearn.covariance import EmpiricalCovariance
import numpy as np

class OutlierRemover:
    def __init__(self):
        self.cov = None
        self.mean = None

    def fit(self, df: pd.DataFrame, columns: list):
        self.cov = EmpiricalCovariance().fit(df[columns])
        self.mean = df[columns].mean(axis=0)

    def remove_outliers_iqr(self, df: pd.DataFrame, columns: list, percentile: float = 99.0):
        if self.cov is None or self.mean is None:
            self.fit(df, columns)

        def mahalanobis_distance(row):
            return mahalanobis(row, self.mean, self.cov.precision_)

        df['Mahalanobis'] = df.apply(lambda row: mahalanobis_distance(row[columns]), axis=1)
        threshold = np.percentile(df['Mahalanobis'], percentile)
        cleaned_df = df[df['Mahalanobis'] <= threshold].drop(columns=['Mahalanobis'])
        removed = len(df) - len(cleaned_df)
        return cleaned_df, removed
```

## Файл model_predictor.py

```
import joblib
import numpy as np
import pandas as pd

class ModelPredictor:
    def __init__(self, path):
        self.params = joblib.load(path)
    def boxcox_transform(self, x, lam):
        x = np.array(x, dtype=float)
        if lam == 0:
            return np.log(x)
        else:
            return (x**lam - 1.0)/lam
    def boxcox_inverse(self, z, lam):
        z = np.array(z, dtype=float)
        if lam == 0:
            return np.exp(z)
```

```
                else:
                    val = z*lam + 1.0
                    mask = (val<=0)
                    out_ = np.full_like(val, np.nan)
                    out_[~mask] = val[~mask]**(1.0/lam)
                    return np.where(np.isnan(out_), 0, out_)
            def predict_rfc(self, cbo, wmc):
                b0 = self.params["b0"]
                b1 = self.params["b1"]
                b2 = self.params["b2"]
                l_rfc = self.params["lambda_rfc"]
                l_cbo = self.params["lambda_cbo"]
                l_wmc = self.params["lambda_wmc"]
                cbo_t = self.boxcox_transform(cbo, l_cbo)
                wmc_t = self.boxcox_transform(wmc, l_wmc)
                z = b0 + b1*cbo_t + b2*wmc_t
                eps = 0.05
                rfc_pred = self.boxcox_inverse(z+eps, l_rfc)
                rfc_pred = np.where(rfc_pred<0, 0, rfc_pred)
                return float(rfc_pred)
            def interpret_quality(self, val):
                if val<7:
                    return "Низька якість ПЗ"
                elif val<30:
                    return "Середня якість ПЗ"
                else:
                    return "Висока якість ПЗ"
```

## Файл plotter.py

```
        import matplotlib.pyplot as plt
        import streamlit as st

        class Plotter:
            @staticmethod
            def plot_intervals(prediction, ci_lower, ci_upper, pi_lower, pi_upper):
                fig, ax = plt.subplots(figsize=(8, 6))
                categories = ['Прогноз']
                ax.plot(categories, [prediction], 'ro', label='Прогноз')
                ax.fill_between(categories, [ci_lower], [ci_upper], color='r', alpha=0.2,
label='Довірчий інтервал (95%)')
                ax.fill_between(categories, [pi_lower], [pi_upper], color='b', alpha=0.2,
label='Прогнозний інтервал (95%)')
                ax.set_title("Прогноз якості ПЗ з довірчим та прогнозним інтервалами")
                ax.set_ylabel("Оцінка якості (умовні одиниці)")
                ax.set_xlabel("Результат оцінки")
                ax.legend()
                ax.grid(True, linestyle='--', alpha=0.7)
                st.pyplot(fig)
                st.write("**Червона точка** – точковий прогноз.")
                st.write("**Червона заштрихована зона** – довірчий інтервал (95%).")
                st.write("**Синя заштрихована зона** – прогнозний інтервал (95%).")
```

## Файл recommendations.py

```
        import streamlit as st

        class Recommendations:
            @staticmethod
            def provide(quality):
                if quality == "Низька якість ПЗ":
                    st.write("**Рекомендації:**")
                    st.write("- Проведіть рефакторинг коду (зменшіть складність, покращіть
архітектуру).")
                    st.write("- Додайте автоматизовані тести, застосуйте статичний аналіз коду.")
                    st.write("- Перегляньте залежності між класами та методами, зменшіть
зв'язність.")
```

```
                    st.write("- Використовуйте інструменти перевірки код-стайл (наприклад, Pylint,
Black).")
                elif quality == "Середня якість ПЗ":
                    st.write("**Рекомендації:**")
                    st.write("- Оптимізуйте складні модулі, зменшіть їхню складність.")
                    st.write("- Збільшіть тестове покриття (юніт-тести, інтеграційні тести).")
                    st.write("- Використовуйте статичний аналіз та профілювання коду для виявлення
потенційних проблем.")
                    st.write("- Покращте алгоритми та структури даних, якщо це можливо.")
                else:
                    st.write("**Рекомендації:**")
                    st.write("- Підтримуйте поточний рівень якості, впровадьте CI/CD для
автоматизації тестування.")
                    st.write("- Розгляньте додаткові інструменти для моніторингу та аналізу якості
коду.")
                    st.write("- Залучіть більше автоматизованих тестів для підтримки високої
якості.")
                    st.write("- Розгляньте можливість масштабування проєкту або застосування нових
патернів проєктування.")
```

# ДОДАТОК В – ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМИ

Програма для прогнозування залежної метрики RFC від CBO та WMC і визначення якості десктоп застосунків, створених на мові C#, яка розроблялась у рамках кваліфікаційної роботи, запускається через сервер за допомогою команди streamlit run main.py. Також програму можна розвернути на хостінгу.

Після запуски програми на сервері або розгорненні її на хостінгу, можна перейти на веб ресурс або локальний домен (наприклад http://localhost:8501/) З'явиться початкове вікно програми, зображене на рис. В.1.

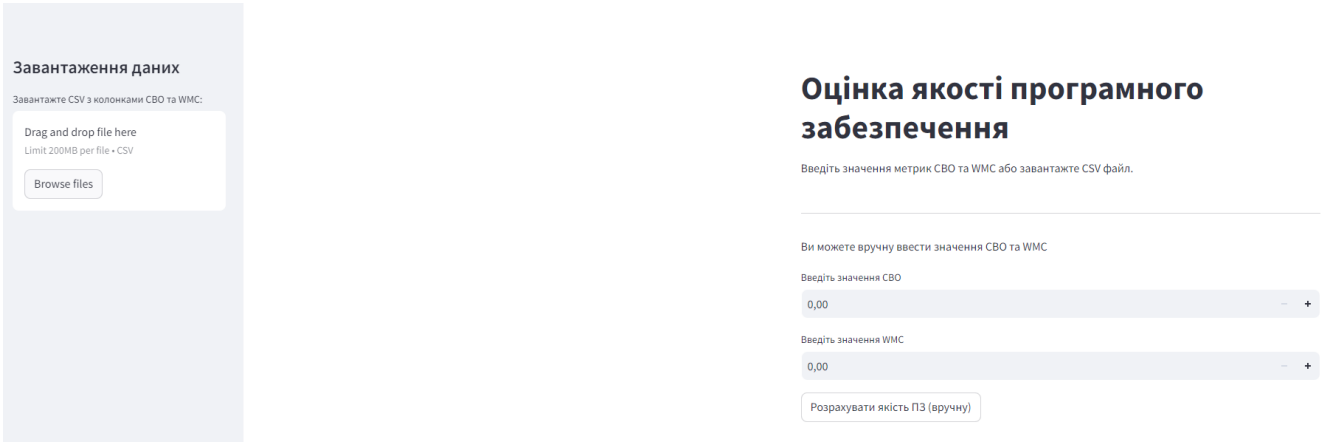


Рисунок В.1 – Початкове вікно застосунку

Для того, щоб завантажити файл для обробки необхідно натиснути на кнопку «Browse files», також можна ввести дані метрик вручну у відповідному вікні. Після того, як буде натиснута кнопка «Browse files», відкриється вікно вибору файлу ОС.

Будуть відображатись файли лише підтримуваного формату (CSV).

Файл повинен містити колонки RFC, CBO та WMC та не бути порожнім.

Для того, щоб імпортувати обраний файл необхідно натиснути на кнопку «Імпорт» або ЛКМ двічі швидко натиснути на файл, після чого файл буде імпортовано.

Після того, як файл буде завантажено, метрики нормалізуються за допомогою перетворення Box-Cox, з'явиться вікно для вибору індексу рядка, щоб почати

прогнозування та розрахунок якості ПЗ. При обиранні рядка та натисканні кнопки «Прогноз обраного рядка», буде розраховані довірчий інтервал, інтервал прогнозування, значення RFC на основі побудованої математичної моделі, визначення якості ПЗ та надання рекомендації щодо покращення якості ПЗ (рис. В.2).

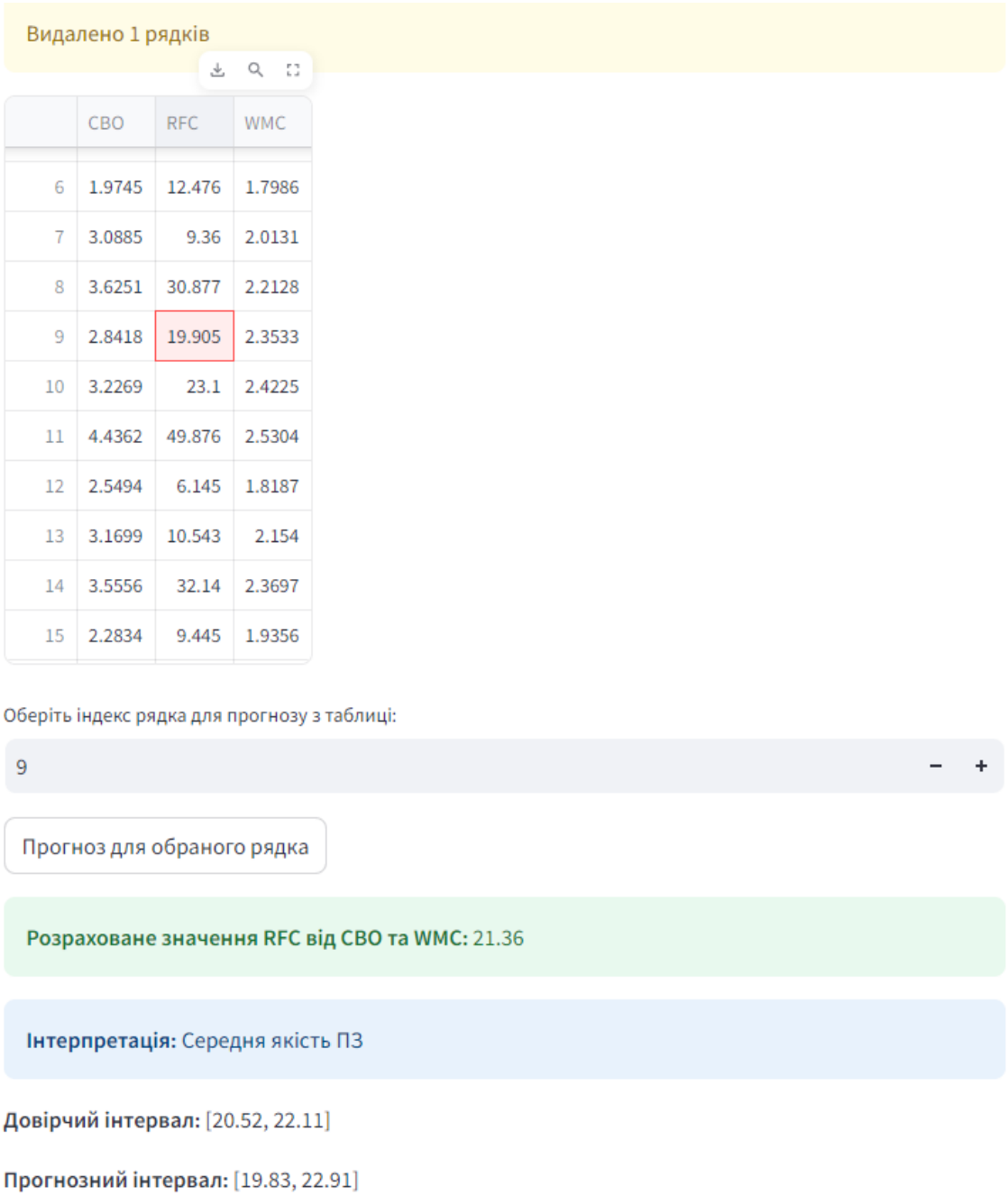


Рисунок В.2 – Вікно результатів прогнозування за обраним рядком

Для того щоб виконати експорт отриманих результатів у CSV файл, необхідно натиснути кнопку «Завантаження результатів в CSV», після чого автоматично почнеться завантаження файлу «result.csv» (рис. В.3).

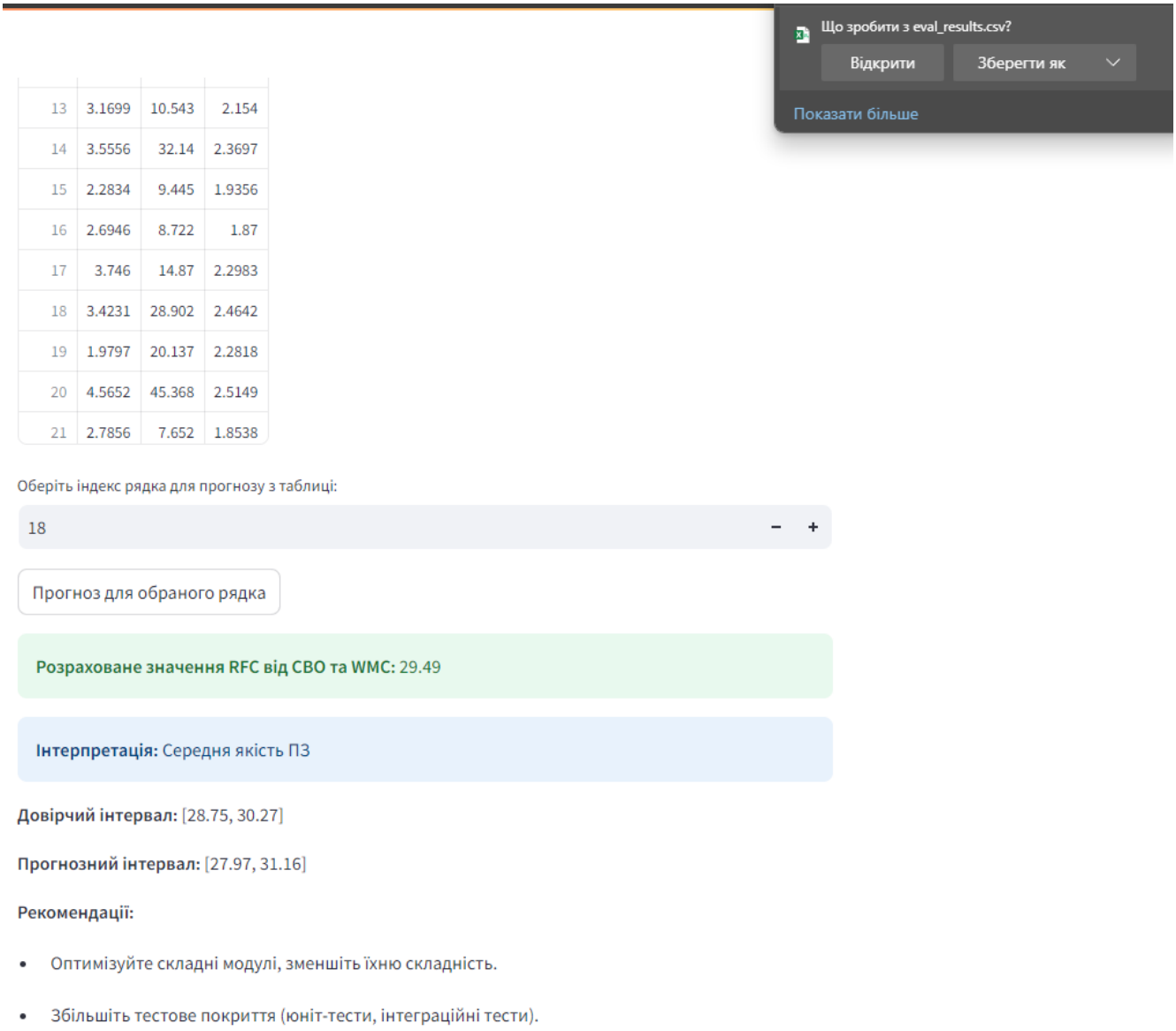


Рисунок В.3 – Завантаження результатів у файл CSV

При виконанні ручного введення метрик, процес відбувається так само, але необхідні метрики потрібно ввести в поля «Ви можете вручну ввести значення CBO та WMC».

# ДОДАТОК Г – ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ

Програма для оцінювання якості десктопних застосунків на C#, створена в рамках цієї роботи, пройшла серію тестувань. Усі виявлені дефекти програмного забезпечення були задокументовані та усунені до моменту введення програми в експлуатацію.

Тестування проводилося на обладнанні у конфігурації, визначеній у технічному завданні (Додаток А). У подальшому наведено перелік тестів, що виконувались для виявлення недоліків програми, описано послідовність їх виконання та проаналізовано реакцію програмного забезпечення на кожен із них.

**Випробування на завантаження порожнього файлу.**

У головному вікні програми натиснули кнопку «Browse files». Після цього був вибраний файл з пустими значеннями та отримано повідомлення про те, що файл є порожнім і проханням завантажити інший файл. (рис Г.1). Випробування пройдено успішно.

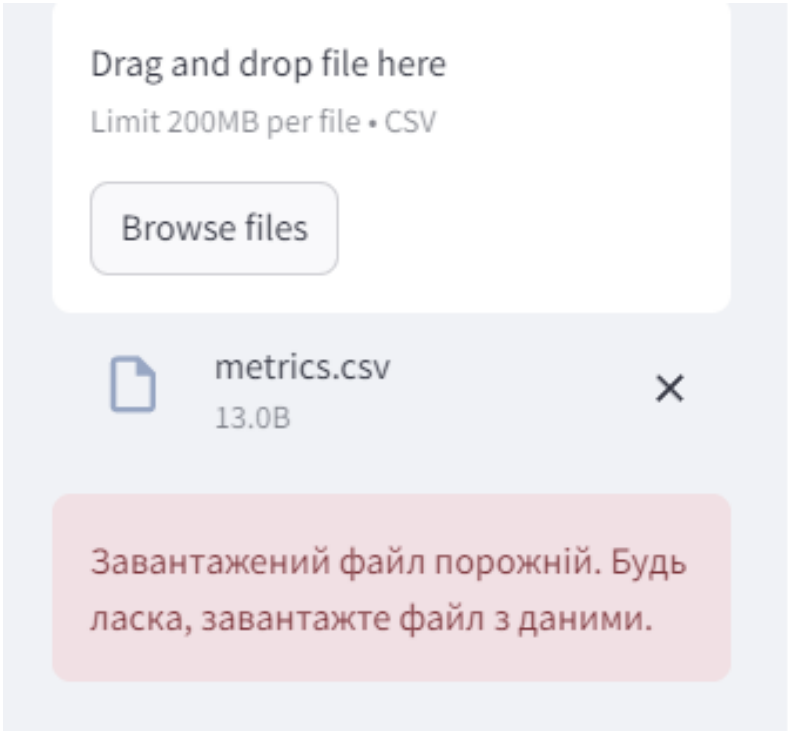


Рисунок Г.1 – Завантаження порожнього файлу

**Випробування на завантаження коректного формату файлу.**

У головному вікні програми натиснули кнопку «Browse files». Файл завантажується без проблем та одразу виводить нормалізовані значення до таблиці. Випробування пройдено успішно.

**Випробування на введення некоректного значення у поля вводу.**

У головному вікні програми в полях для введення метрик, при спробі ввести 0 значення, отримуємо помилку «Box-Cox трансформація вимагає позитивних значень для всіх метрик». При введені негативних значень, повідомлення про те, що не можна вводити значення нижче 1. Випробування пройдено успішно.

**Випробування на введення некоректного значення для індексу рядка.**

Після завантаження файлу з головного вікна програми та спробі обрати індекс рядка вище за той, що залишився у вибірці, отримуємо повідомлення, що значення має бути менше або дорівнювати (кількості рядків у таблиці).

Випробування пройдено успішно (рис. Г.2).

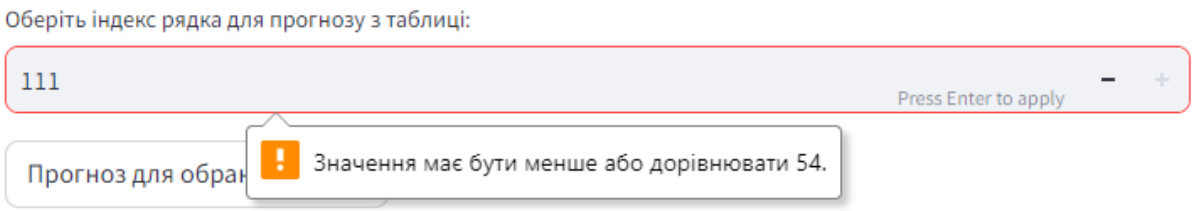


Рисунок Г.2 – Введення неіснуючого значення індексу рядка

Таким чином, усі проведені тестування підтвердили очікувану реакцію програми на дії користувача. Це дозволяє зробити висновок, що програмне забезпечення функціонує коректно, запобігаючи некоректним діям з боку користувача та, у разі виникнення проблем, надає відповідну інформацію про можливі способи їх усунення.

# ДОДАТОК Д – ОПИС ПРОГРАМИ

Програма для оцінювання якості десктоп застосунків мовою програмування C# була розроблена у рамках кваліфікаційної роботи з використанням мови програмування Python та інтегрованого середовища розробки PyCharm.

Вона орієнтована на підприємства, що займаються розробкою програмного забезпечення, для автоматизації оцінювання якості продуктів, оптимізації бізнес-процесів і обробки статистичних даних.

Метою програми є:

– спрощення бізнес-процесів для підприємств-розробників ПЗ;

– покращення ефективності обробки даних;

– підвищення достовірності результатів аналізу;

– автоматизація процесів оцінювання якості програмного забезпечення.

Програма забезпечує автоматизацію наступних процесів:

– введення статистичних даних вручну або за допомогою завантаження файлу CSV;

– нормалізація вибірок за допомогою перетворення Box-Cox;

– видалення викидів за допомогою квадрату відстані Махаланобіса;

– побудова математичної моделі;

– використання побудованої двофакторної нелінійної регресійної моделі для визначення якості ПЗ за допомогою прогнозування залежної метрики RFC від CBO та WMC;

– розрахунок довірчого та прогнозуючого інтервалів;

– надання рекомендацій, щодо покращення ПЗ, що оцінюється.

Для реалізації програмного забезпечення використовувалися:

– мова програмування: Python (з використанням бібліотек pandas, numpy, scipy, matplotlib, streamlit);

– середовищем розробки є IDE PyCharm;

– графічний інтерфейс реалізовано із використанням бібліотеки streamlit, як веб застосунок.

Велика увага приділена дизайну користувацького інтерфейсу:

– простота використання для кінцевого користувача;

– використання стандартних графічних елементів та кольорових схем для поліпшення сприйняття даних;

Перед впровадженням програми було проведено тестування:

– функціональне тестування, а саме перевірка на відповідність функціональним вимогам;

– навантажувальне тестування, де була оцінена робота програми при обробці великих масивів даних або невірних даних.

Усі виявлені недоліки були зафіксовані, виправлені та перевірені повторно. Результати тестування показали повну готовність програми до впровадження та ефективного використання у промислових умовах.

Розроблена програма є ефективним інструментом для підприємств, що займаються розробкою програмного забезпечення. Вона забезпечує автоматизоване визначення якості десктоп застосунків на мові C#, надаючи можливість швидко й достовірно аналізувати показники.