МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
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 оцінювання якості застосунків, що створюються на платформі Flutter, та розробка програми для її реалізації.

Виконав: студент групи 6151м
_____Шебалін Н.О.
(підпис, ПІБ)

Керівник роботи:
____завідувач кафедри ПЗАС, проф.
(посада, науковий ступень вчене звання)
_____Приходько С.Б.
(підпис, ПІБ)

Миколаїв – 2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
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 Шебаліну Нікіті Олександровичу _____
(Прізвище, ім'я, по батькові)

1. Тема роботи: Математична модель для оцінювання якості застосунків, що створюються на платформі Flutter, та розробка програми для її реалізації _____
Керівник роботи завідувач кафедри ПЗАС, проф. Приходько Сергій Борисович _____
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. Апробація результатів досліджень**.** Публікації.); 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 Титульний слайд, Мета та завдання дослідження, Об'єкт та Предмет дослідження, Наукова новизна одержаних результатів, Практичне значення одержаних результатів, Особистий внесок здобувача, Апробація результатів досліджень та Публікації, Аналіз існуючих методів та моделей для оцінювання якості програмного забезпечення, Математична модель оцінювання якості Flutter-застосунків, Аналіз вимог до програмного забезпечення, Архітектура програмного забезпечення, Діаграма варіантів використання, Технічний проект програмного забезпечення, Вибір інструментів розробки, Реалізація та Тестування програмного забезпечення, Результати розробки, Висновки.

6. Консультанти розділів роботи

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

7. Дата видачі завдання 14.10.2024 р. _____

К А Л Е Н Д А Р Н И Й П Л А Н

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 _____ Шебалін Н.О.
(підпис) (ПІБ)

Керівник роботи _____ Приходько С.Б.
(підпис) (ПІБ)

РЕФЕРАТ

Шебалін Нікіта Олександрович

«Математична модель для оцінювання якості застосунків, що створюються на платформі Flutter, та розробка програми для її реалізації»

Кваліфікаційна (магістерська) робота на здобуття освітнього рівня магістра зі спеціальності 121 «Інженерія програмного забезпечення». Національний університет кораблебудування імені адмірала Макарова. Миколаїв, 2024 р.

Обсяг роботи: 120 стор., 9 табл., 19 рис., 43 використаних джерел, 5 додатків.

Актуальність теми: Дослідження охоплює актуальну проблему оцінювання якості програмного забезпечення, особливо для застосунків, розроблених на платформі Flutter. З огляду на зростаючу популярність Flutter для кросплатформної розробки, ця робота сприяє розвитку галузі, забезпечуючи знання, адаптовані до унікальних архітектурних особливостей платформи.

Мета та завдання дослідження: Мета дослідження полягає у підвищенні достовірності оцінювання якості застосунків на Flutter шляхом розробки спеціалізованої математичної моделі. Основними завданнями є аналіз існуючих методів оцінки якості, визначення специфічних метрик для Flutter-застосунків, створення моделі оцінки якості, розробка програмного забезпечення на основі цієї моделі та проведення тестування для перевірки достовірності.

Об'єкт дослідження: Об'єктом дослідження є процес оцінювання якості програмного забезпечення, розробленого на платформі Flutter.

Предмет дослідження: Предметом дослідження є математична модель для оцінювання якості застосунків на Flutter, що використовує метрики CBO (Coupling Between Object Classes), WMC (Weighted Methods per Class) та RFC (Response for Class) для прогнозування.

Методи дослідження: У роботі застосовано методи регресійного аналізу, математичної статистики, нормалізуючих перетворень і теорії ймовірностей для побудови та перевірки моделі. Емпіричне тестування проводилося на реальних даних Flutter-застосунків для підтвердження практичної придатності.

Наукова новизна: Удосконалено математичну модель для оцінювання якості Flutter-застосунків, яка, на відміну від існуючих моделей для Java та Kotlin застосунків, враховує інтервал значень метрик застосунків, що створюються на платформі Flutter, шляхом визначення нових параметрів моделі та застосування нормалізуючого перетворення у вигляді десяткового логарифму для прогнозування метрики RFC на основі метрик CBO та WMC, що забезпечує достовірне оцінювання якості програмного забезпечення, розробленого на даній платформі.

Практична значущість: Модель надає розробникам можливість ефективно оцінювати якість застосунків на Flutter, що сприяє кращому управлінню проектами, оптимальному використанню ресурсів та підвищенню стабільності програмного забезпечення, відповідно до галузевих стандартів та національних цілей ІТ-конкурентоспроможності.

Апробація результатів дослідження: Основні результати дослідження були представлені на V Всеукраїнській науково-практичній інтернет-конференції «Інформаційні технології: моделі, алгоритми, системи», організованій Національним університетом кораблебудування у жовтні 2024 року.

Публікації: За результатами дослідження опубліковано одну наукову статтю у матеріалах конференції.

Ключові слова: Оцінювання якості, Flutter, програмні метрики, математичне моделювання, кросплатформна розробка.

REVIEW

Shebalin Nikita

«A mathematical model for assessing the quality of applications created on the Flutter platform and development of a program for its implementation»

Qualification (master's) thesis for obtaining a master's degree in the speciality 121 «Software Engineering». National University of Admiral Makarov National University of Shipbuilding. Mykolaiv, 2024.

Scope of work: 120 pages, 9 tables, 19 figures, 43 references, 5 appendices.

Relevance of the topic of the work: The study addresses the pressing need for reliable software quality assessment, particularly for applications developed using the Flutter framework. Given Flutter’s growing popularity in cross-platform application development, this research contributes to the evolving landscape by providing insights specific to this platform’s unique architectural traits.

Purpose and objectives of the study: The study aims to enhance the accuracy of quality assessments for Flutter-based applications by developing a specialized mathematical model. Objectives include analyzing existing software quality assessment methods, identifying specific metrics for Flutter applications, designing a quality assessment model, implementing software based on this model, and conducting validation tests.

The object of the study: The object of the study is the quality assessment process for software developed on the Flutter platform.

Subject of the study: The subject focuses on a mathematical model for quality assessment of Flutter applications, utilizing the metrics CBO (Coupling Between

Object Classes), WMC (Weighted Methods per Class), and RFC (Response for Class) for predictive analysis.

Methods of the study: The research employs regression analysis, mathematical statistics, normalizing transformations, and probability theory for model construction and validation. Empirical testing was performed on actual data from Flutter applications to confirm practical applicability.

Scientific novelty of the results obtained: A mathematical model for assessing the quality of Flutter applications has been improved, which, unlike existing models for Java and Kotlin applications, takes into account the range of values of the metrics of applications created on the Flutter platform by defining new model parameters and applying a normalising transformation in the form of a decimal logarithm to predict the RFC metric based on the CBO and WMC metrics, which provides a reliable assessment of the quality of software developed on this platform

Practical significance of the obtained results: The model provides practical utility by enabling developers to assess Flutter app quality effectively. This leads to improved project management, resource allocation, and overall software stability, aligning with industry standards and national goals for IT competitiveness.

Testing of the research results: The primary findings were presented at the Fifth All-Ukrainian Scientific-Practical Online Conference “Information Technology: Models, Algorithms, Systems” held by the National University of Shipbuilding in October 2024.

Publications: The study has resulted in one academic publication within the conference proceedings.

Keywords: Quality assessment, Flutter, software metrics, mathematical modeling, cross-platform development.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CBO – Coupling Between Object Classes;

NOC – Number of Classes;

RFC – Response for Class;

WMC – Weighted Methods per Class.

ВСТУП

Актуальність теми дослідження зумовлена нагальною потребою в розробці ефективних методів оцінювання якості програмного забезпечення, особливо для сучасних платформ розробки. Оцінювання якості програмного забезпечення є критично важливою задачею, яка безпосередньо впливає на надійність, ефективність та підтримку програмних продуктів. Отже виникає задача, яка пов’язана з оцінюванням якості програмного забезпечення. Вона важлива для забезпечення стабільності, надійності та підтримки продуктів у розробці, особливо в умовах швидкого розвитку IT-технологій. Крім того, дослідження у сфері оцінювання якості програмного забезпечення мають важливе значення для наукової спільноти. Вони сприяють розвитку нових методів та інструментів, що можуть бути застосовані в різних контекстах і для різних фреймворків. Це, в свою чергу, сприяє загальному прогресу в галузі інженерії програмного забезпечення, допомагаючи розробникам створювати більш складні та якісні продукти. Для розв'язання цієї задачі розроблено різні підходи, методи та моделі [1-9]. Одним із методів визначення якості є метод [2], який базується на основі довірчих і прогнозних інтервалів нелінійної регресії для прогнозування RFC (Response for Class) використовуючи метрики CBO (Coupling Between Object Classes) та WMC (Weighted Methods per Class) [10-12].

З появою платформи Flutter [13], яка стала одним із провідних інструментів для створення кросплатформних додатків, виникла необхідність адаптації існуючих підходів до оцінювання якості. Фреймворк Flutter набув великої популярності завдяки можливості розробки кросплатформених додатків з однією базою коду із мовою програмування Dart [14]. Dart – це універсальна мова програмування з відкритим вихідним кодом, розроблена компанією

Google. Вона призначена для створення користувацьких інтерфейсів і особливо відома завдяки використанню з Flutter, інструментарієм UI Material Design [15], також створеним Google. Flutter має унікальну архітектуру, засновану на компонентному підході з використанням віджетів та реактивного програмування [16, 17], що суттєво впливає на характеристики програмного коду та значення метрик якості.

Дослідження показують [18], що Flutter-застосунки демонструють значно інші інтервали значень метрик порівняно з традиційними платформами Java та Kotlin. Зокрема, значення метрик CBO та WMC для Flutter-проектів систематично нижчі, що робить існуючі моделі оцінювання якості, розроблені для Java та Kotlin [19, 20], непридатними для оцінювання Flutter-застосунків. Особливо це стосується математичної моделі для прогнозування метрики RFC на основі метрик CBO та WMC, яка використовується для оцінки якості Java-застосунків та Kotlin-застосунків, але не враховує специфіку архітектури та особливості метрик Flutter-застосунків. Це підкреслює необхідність розробки спеціалізованої математичної моделі, яка враховуватиме особливості Flutter-застосунків та забезпечить достовірне оцінювання їх якості.

З огляду на викладене, розробка ефективних методів та інструментів для оцінювання якості програмного забезпечення, особливо такого, що розробляється за допомогою Flutter, є актуальною та важливою задачею. Це дозволить забезпечити високу точність прогнозів, підвищити ефективність управління проектами та сприяти загальному покращенню якості програмного забезпечення. Використання платформи Flutter для розробки застосунків стає дедалі популярнішим, і запропонована в дослідженні математична модель для оцінювання якості таких проектів сприятиме вдосконаленню процесу розробки. Це відповідає національним та міжнародним програмам з розвитку IT-індустрії, де пріоритетом є створення якісних програмних продуктів із оптимальним

використанням ресурсів і часу, що також важливо для України в умовах глобальної конкуренції на IT-ринку.

Мета дослідження полягає у підвищенні достовірності оцінювання якості застосунків, що створюються на платформі Flutter, шляхом удосконалення математичної моделі на основі метрик RFC, CBO та WMC, та розробці програмного забезпечення для її реалізації. Для досягнення мети потрібно вирішити наступні завдання:

– Проаналізувати існуючі підходи до оцінювання якості програмного забезпечення;

– Визначити специфічні метрики якості для застосунків, розроблених на Flutter;

– Розробити математичну модель для оцінювання якості програмних продуктів;

– Реалізувати програмне забезпечення для оцінювання якості на основі розробленої моделі;

– Провести тестування моделі та програмного забезпечення для забезпечення достовірності прогнозування.

Об'єктом дослідження є процес оцінювання якості програмного забезпечення, розробленого на платформі Flutter, та розробка програмного забезпечення для її реалізації.

Предметом дослідження є математична модель для оцінювання якості застосунків, створених на платформі Flutter, яка використовує метрики CBO та WMC для прогнозування RFC, та програмне забезпечення для її реалізації.

*Методи дослідження***.** У роботі застосовано методи регресійного аналізу, математичної статистики, нормалізуючих перетворень, а також методи теорії ймовірностей для побудови та оцінки достовірності моделі. Також проводилось

емпіричне тестування моделі на основі реальних даних Flutter застосунків для перевірки її практичної придатності.

Наукова новизна. Удосконалено математичну модель для оцінювання якості Flutter-застосунків, яка, на відміну від існуючих моделей для Java та Kotlin застосунків, враховує інтервал значень метрик застосунків, що створюються на платформі Flutter, шляхом визначення нових параметрів моделі та застосування нормалізуючого перетворення у вигляді десяткового логарифму для прогнозування метрики RFC на основі метрик CBO та WMC, що забезпечує достовірне оцінювання якості програмного забезпечення, розробленого на даній платформі.

Особистий внесок здобувача. Всі результати отримані самостійно. В роботі [18], яка опублікована у співавторстві з науковим керівником, автору належить: збір та обробка даних метрик з Flutter-застосунків, розробка скрипту для автоматизованого збору метрик, проведення порівняльного аналізу діапазонів метрик з Java та Kotlin застосунками, побудова нелінійної регресійної моделі та перевірці її якості.

Апробація результатів досліджень. Основні результати досліджень були оприлюднені на V Всеукраїнській науково-практичній інтернет-конференції «Інформаційні технології: моделі, алгоритми, системи» (30-31 Жовтня 2024), організованій Національним університетом кораблебудування імені адмірала Макарова.

Публікації. За результатами досліджень опубліковано одну наукову працю, а саме матеріали інтернет-конференції.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА МОДЕЛЕЙ ДЛЯ ОЦІНЮВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд програмних метрик які використовуються для оцінювання якості програмного забезпечення

Оцінювання якості програмного забезпечення є комплексним процесом, який вимагає застосування різноманітних метрик та методів аналізу. У сучасній практиці розробки програмного забезпечення використовується широкий спектр метрик, кожна з яких відповідає за оцінку певних аспектів якості програмного продукту.

Важливу роль в оцінюванні якості відіграють метрики складності коду. Цикломатична складність дозволяє оцінити складність потоку керування програми, вимірюючи кількість лінійно незалежних шляхів через програмний код. Когнітивна складність, у свою чергу, фокусується на оцінці того, наскільки код складний для розуміння людиною, враховуючи вкладеність структур керування та логічні переходи. Метрика Холстеда [21] представляє собою набір показників, що оцінюють складність програми на основі кількості операторів та операндів у коді.

Не менш важливими є метрики зв'язності та зчеплення програмного коду. Метрика CBO (Coupling Between Object Classes) вимірює ступінь зв'язку між класами через виклики методів, наслідування та використання змінних. LCOM (Lack of Cohesion of Methods) оцінює ступінь зв'язності методів всередині класу через спільне використання змінних екземпляра. DIT (Depth of Inheritance Tree) дозволяє оцінити глибину успадкування класу в ієрархії.

Розмір та структуру програмного забезпечення можна оцінити за допомогою таких метрик як LOC (Lines of Code) [22], що вимірює кількість рядків коду, NOC (Number of Children), яка показує кількість безпосередніх підкласів класу, та WMC (Weighted Methods per Class), що оцінює складність класу через суму складності його методів.

Особливу увагу слід приділити метрикам відповідальності та взаємодії компонентів. RFC (Response for Class) вимірює кількість методів, які можуть бути викликані у відповідь на повідомлення, отримане об'єктом класу. Набір метрик MOOD (Metrics for Object Oriented Design) включає фактори інкапсуляції, наслідування, поліморфізму та зчеплення, що дозволяє комплексно оцінити об'єктно-орієнтований дизайн програми.

Для оцінювання якості Flutter-застосунків було обрано комбінацію метрик RFC, CBO та WMC. Цей вибір обумовлений декількома важливими факторами. По-перше, дана комбінація дозволяє провести комплексну оцінку трьох ключових аспектів якості програмного забезпечення: складності взаємодії компонентів, зв'язності між класами та внутрішньої складності компонентів. По-друге, ці метрики добре адаптуються до специфіки Flutter-архітектури, яка базується на компонентному підході з використанням віджетів та особливостях реактивного програмування.

Практична значимість обраних метрик підтверджується наявністю значної кореляції між ними та показниками якості програмного забезпечення, а також можливістю їх автоматизованого обчислення. Важливо відзначити, що ці метрики мають чітку інтерпретацію для розробників, що полегшує їх практичне застосування. Крім того, існують успішні приклади використання даних метрик для оцінки якості Java та Kotlin проектів [19, 20], а також достатня теоретична база для їх адаптації до нових платформ.

Існують і альтернативні підходи до оцінювання якості програмного забезпечення. Метрики продуктивності [23], такі як час відгуку, використання

ресурсів та пропускна здатність, дозволяють оцінити ефективність роботи програми. Метрики надійності [24], включаючи частоту відмов, середній час між відмовами та доступність системи, фокусуються на стабільності роботи програмного забезпечення.

Проте для оцінки якості коду Flutter-застосунків комбінація RFC, CBO та WMC має суттєві переваги. Ці метрики безпосередньо відображають архітектурні особливості Flutter, дозволяють оцінювати якість на етапі розробки та надають кількісні показники для порівняння різних реалізацій. Вони забезпечують можливість комплексно оцінювати різні аспекти якості коду, враховувати специфіку архітектури Flutter, отримувати об'єктивні кількісні показники та підтримувати автоматизацію процесу оцінювання.

Подальші дослідження спрямовані на адаптацію цих метрик до особливостей Flutter-розробки та створення математичної моделі для їх ефективного використання в контексті оцінювання якості застосунків.

1.2 Особливості розробки застосунків на платформі Flutter

Flutter є сучасним фреймворком для кросплатформної розробки, розробленим компанією Google, який використовується для створення мобільних, веб та настільних додатків. Завдяки використанню Flutter розробники можуть створювати додатки з єдиною базою коду, що забезпечує економію часу і ресурсів. Однак особливості архітектури Flutter вносять специфічні вимоги до оцінювання якості, оскільки платформа суттєво відрізняється від традиційних підходів у Java та Kotlin.

Основною характеристикою Flutter є його віджет-орієнтована архітектура, що робить віджет головною структурною одиницею, яка відповідає за інтерфейс та взаємодію з користувачем [16]. Всі елементи, від простих кнопок до складних

структур, реалізовані через ієрархії віджетів. Такий підхід забезпечує високу модульність, проте ускладнює застосування традиційних метрик CBO та RFC, які орієнтовані на класичні об’єктно-орієнтовані системи з високим рівнем зв’язності між класами.

Flutter використовує мову Dart [14], яка підтримує як компіляцію Just-in-Time (JIT), що забезпечує швидкий цикл розробки завдяки "гарячому перезавантаженню" (hot reload), так і компіляцію Ahead-of-Time (AOT), яка підвищує продуктивність у production-середовищі. Dart надає засоби для об’єктно-орієнтованого програмування, але його синтаксис та функції дещо відрізняються від інших мов, що впливає на обчислення метрик WMC та CBO.

Flutter дотримується принципів реактивного програмування [17], де стан додатка зберігається в спеціальних об’єктах. Зміна стану автоматично оновлює інтерфейс, що забезпечується через механізми управління станом (State Management [25]) – такі як Provider, Bloc та інші. Це ускладнює застосування традиційних метрик для оцінювання складності, оскільки комунікація між компонентами реалізується через управління станом, а не прямі зв’язки між класами, що знижує значення CBO.

У Flutter асинхронність досягається за допомогою Future та Stream API [26], які забезпечують зручну обробку асинхронних операцій. Це важливо для архітектури додатків, особливо при взаємодії з мережею або базами даних, але також ускладнює тестування і збільшує складність додатка. Ця особливість вимагає адаптації метрик для врахування асинхронної взаємодії.

Flutter використовує рушій Skia для рендерингу [27], що дозволяє досягти продуктивності, близької до нативної, але вимагає ретельної оптимізації для забезпечення плавної роботи додатка. Водночас застосування віджетів в ієрархіях може збільшувати навантаження на рендеринг, що також варто враховувати при оцінюванні продуктивності і якості коду.

Flutter дозволяє використовувати бібліотеки та пакети з офіційного репозиторію pub.dev, що з одного боку прискорює розробку, але з іншого – ускладнює оцінку якості, оскільки залежність від зовнішніх бібліотек впливає на зв’язність та повторну використання компонентів. Це створює потребу у нових підходах до інтерпретації CBO у Flutter-додатках.

На основі аналізу особливостей Flutter стає зрозумілим, що архітектура фреймворку значно відрізняється від Java та Kotlin, де традиційні метрики CBO, RFC та WMC використовуються для оцінювання класів із високим рівнем зв’язності. В Flutter, через модульність віджетів і низький рівень зчеплення між ними, застосування цих метрик потребує адаптації. Наприклад, низьке значення CBO для Flutter-застосунків відображає не слабкість архітектури, а специфіку компонентної структури, що підкреслює необхідність розробки нової моделі для оцінки якості.

Таким чином, робота спрямована на удосконалення математичної моделі оцінювання якості для Flutter-застосунків, яка враховуватиме зазначені архітектурні особливості.

1.3 Аналіз існуючих моделей оцінювання якості програмного забезпечення

Оцінювання якості програмного забезпечення є критично важливим аспектом процесу розробки, що допомагає передбачити потенційні проблеми та оптимізувати використання ресурсів. На сьогоднішній день існує широкий спектр моделей оцінювання якості програмного забезпечення, кожна з яких має свої переваги та обмеження.

Ієрархічні моделі якості, такі як ISO/IEC 9126 [3] та ISO/IEC 25010 [4], пропонують стандартизований підхід до оцінки якості через ієрархію атрибутів.

Модель McCall [5] фокусується на трьох основних аспектах якості – операційних характеристиках, можливості до змін та адаптивності. Модель Boehm [6] розглядає якість через призму потреб користувача, забезпечуючи більш практичний підхід до оцінювання.

Метричні моделі представлені широким спектром підходів. Моделі складності, такі як метрики McCabe та Halstead [7], дозволяють оцінити структурну складність коду. Об'єктно-орієнтовані метрики, зокрема набір метрик Chidamber і Kemerer (CK) [8], спеціально розроблені для оцінки якості об'єктно-орієнтованого програмного забезпечення. Моделі розміру, включаючи SLOC та Function Points, надають кількісну оцінку обсягу програмного забезпечення.

Статистичні моделі включають як лінійні [1], так і нелінійні регресійні підходи [2]. Лінійні регресійні моделі, хоча і прості у реалізації та інтерпретації, часто не можуть адекватно описати складні залежності між метриками програмного забезпечення. Нелінійні регресійні моделі, включаючи степеневі, експоненціальні, логістичні та логарифмічні варіанти, забезпечують більш точний опис складних залежностей.

Моделі машинного навчання [9] представляють сучасний підхід до оцінки якості програмного забезпечення. Нейронні мережі здатні виявляти складні нелінійні залежності між різними характеристиками якості. Методи опорних векторів (SVM) особливо ефективні для класифікації якісних характеристик. Дерева рішень та випадкові ліси забезпечують добре інтерпретовані моделі для оцінки якості.

Для оцінювання метрик якості Flutter-застосунків було обрано математичну модель із використанням нелінійної регресії з кількох причин. Специфіка метрик якості програмного забезпечення, таких як RFC, CBO та WMC, вказує на нелінійний характер їх взаємозв'язків, який добре описується нелінійними регресійними моделями. Крім того, успішне застосування подібних

моделей для оцінювання метрик якості в інших мовах програмування, наприклад Java [19] та Kotlin [20], підтверджує ефективність цього підходу.

Серед різних типів нелінійних регресійних моделей було обрано логарифмічну модель на основі десяткового логарифму. Це рішення обумовлене здатністю такої моделі забезпечувати природне вирівнювання даних, зменшуючи вплив викидів та створюючи більш рівномірний розподіл значень. Логарифмічна трансформація ефективно справляється з позитивною асиметрією, часто властивою метрикам програмного забезпечення. Важливою перевагою обраної моделі є простота інтерпретації її коефіцієнтів, які мають чітке значення з точки зору відносних змін в оцінюваних величинах. Це робить модель особливо цінною для практичного застосування в процесі розробки програмного забезпечення. Десятковий логарифм був обраний як основа для нелінійної регресії з наступних причин:

– Природне вирівнювання даних: Десятковий логарифм дозволяє зменшити вплив викидів та "розтягнути" значення, які знаходяться ближче до мінімуму, що забезпечує більш рівномірний розподіл даних.

– Зменшення асиметрії та викривлень: Логарифмічна трансформація ефективно справляється з позитивною асиметрією, часто властивою метрикам програмного забезпечення, особливо у випадках, коли CBO і WMC мають незначні або проміжні значення.

– Інтерпретаційна простота: На відміну від степеневих та експоненціальних функцій, логарифмічна модель дозволяє отримати коефіцієнти, які легше інтерпретувати, оскільки зміни метрик у логарифмічній шкалі відповідають відносним змінам в оцінюваній величині RFC.

Таким чином, вибір десяткового логарифму для нелінійної регресійної моделі обумовлений його здатністю знижувати викривлення даних, забезпечуючи при цьому високу точність прогнозування без необхідності застосування надто складних функцій. Це дозволяє отримувати стабільні

результати при оцінюванні якості Flutter-застосунків та підвищити практичну цінність моделі для інженерів програмного забезпечення.

Існуючі моделі для Java [19] та Kotlin [20] базуються на припущеннях про структуру та організацію коду, які не завжди відповідають реаліям Flutter-розробки. Модель для Java-застосунків враховує більш складну систему успадкування та високу зв'язність між класами, що є типовим для enterprise-додатків на цій платформі. Дана модель побудована у роботі [19] на основі перетворення у вигляді десяткового логарифму. Модель має вигляд:

$$Y = 10^{\varepsilon+\hat{b}_0} X_1^{\hat{b}_1} X_2^{\hat{b}_2}, \quad (1.1)$$

де Y – метрика RFC, X_1 – метрика CBO, X_2 – метрика WMC, ε – гаусівська випадкова величина з нульовим математичним сподіванням та середньоквадратичним відхиленням, оцінка якого дорівнює 0,0654, $\hat{b}_0$, $\hat{b}_1$ та $\hat{b}_2$ є оцінками параметрів, які мають відповідно такі значення: -0,0548, 0,6723 та 0,4415.

Нелінійна регресійна модель для Kotlin-застосунків хоча і показує кращу відповідність через більш сучасний підхід до розробки, все ще не враховують специфіку Flutter, таку як широке використання композиції віджетів замість класичного успадкування та особливості управління станом через State-об'єкти. Дана модель побудована у роботі [20] також на основі перетворення у вигляді десяткового логарифму. Модель має аналогічний вигляд (1.1), де ε дорівнює 0,1128, $\hat{b}_0$, $\hat{b}_1$ та $\hat{b}_2$ є оцінками параметрів, які мають відповідно такі значення: -0,3069, 0,2286 та 1,3621.

Результати показали, що наявні підходи не можуть бути точно адаптовані для Flutter через суттєві відмінності в діапазонах значень метрик:

1. Метрика CBO (Coupling Between Object Classes):

– Flutter: [0,125, 1,46] – найнижчі значення серед усіх платформ;

– Java: [5,071, 22,417] – найвищі значення, що в 15-40 разів перевищують Flutter;

– Kotlin: [0,995, 4,724] – середні значення, ближчі до Flutter.

2. Метрика WMC (Weighted Methods per Class):

– Flutter: [2,5, 13,66] – середній діапазон;

– Java: [8,042, 71,837] – найширший діапазон, з суттєво вищою верхньою межею;

– Kotlin: [2,167, 13,020] – майже співпадає з Flutter.

3. Метрика RFC (Response for Class):

– Flutter: [0,28, 9,92] – найнижчі значення;

– Java: [7,306, 37,278] – найвищі значення, що в 26-37 разів перевищують Flutter;

– Kotlin: [2,118, 19,487] – середні значення.

Flutter-застосунки демонструють найнижчі значення зв'язності між класами (CBO) та кількості методів, що можуть бути викликані у відповідь на повідомлення (RFC). Java-застосунки мають найвищі показники за всіма метриками, Kotlin займає проміжне положення, але його показники ближчі до Flutter. Виявлені відмінності в діапазонах метрик підтверджують необхідність створення нової моделі для оцінювання RFC у Flutter-проектах.

Модель оцінювання якості [2] базується на використанні довірчих та прогнозних інтервалів для метрики RFC, що прогнозується на основі значень CBO та WMC. Використання довірчого інтервалу дозволяє визначити діапазон, у якому з високою ймовірністю перебуватимуть середні значення RFC, що надає інформацію про загальну якість програмного забезпечення. Прогнозний інтервал, своєю чергою, вказує на можливий діапазон варіацій RFC для нових даних, що забезпечує більш гнучкий та точний підхід до оцінювання якості окремих застосунків.

Цей підхід дозволяє класифікувати якість програмного забезпечення на високий, середній та низький рівні залежно від того, де розташоване значення RFC відносно меж довірчого та прогнозного інтервалів. Зокрема, якщо значення RFC знаходиться в межах довірчого інтервалу, це свідчить про середню якість застосунку. Знаходження значення RFC між межами довірчого і прогнозного інтервалів свідчить про високу якість, а вихід за верхню межу прогнозного інтервалу може вказувати на низьку якість програмного продукту.

Таким чином, проведений аналіз існуючих моделей оцінювання якості програмного забезпечення підтверджує необхідність створення нової моделі, спеціально адаптованої для оцінювання метрики RFC у Flutter-проектах, яка б враховувала специфіку платформи та реальні діапазони метрик, що спостерігаються у практичних застосунках.

2 РОЗРОБКА МАТЕМАТИЧНОЇ МОДЕЛІ ДЛЯ ОЦІНЮВАННЯ ЯКОСТІ FLUTTER-ЗАСТОСУНКІВ

2.1 Збір та попередня обробка даних метрик Flutter-проектів

Для побудови математичної моделі для оцінювання якості Flutter-застосунків були зібрані дані зі 100 Flutter-застосунків з відкритим кодом, розміщених на GitHub. Метрики RFC, CBO та WMC на рівні застосунку були отримані за допомогою скрипту на Python (https://github.com/shebnik/flutter_metrics) та інструменту Dart Code Metrics [28]. З кожного проекту було отримано наступні відносні значення метрик в залежності від кількості класів:

RFC – це залежна змінна Y, яку потрібно спрогнозувати за допомогою моделі. CBO – незалежна змінна X_1, та WMC – незалежна змінна X_2 на основі яких буде здійснюватися прогнозування залежної змінної Y.

Для перевірки багатовимірних даних на нормальність був застосований тест Мардія [29], що враховує багатовимірну асиметрію та ексцес. Тест Мардія обчислює два основні параметри Асиметрія та Ексцес. Асиметрія оцінюється на основі квадрату відстані Махаланобіса [30]:

$$d_i^2 = \begin{bmatrix} Y_i - \bar{Y} \\ X_{1i} - \bar{X}_1 \\ X_{2i} - \bar{X}_2 \end{bmatrix}^T S_N^{-1} \begin{bmatrix} Y_i - \bar{Y} \\ X_{1i} - \bar{X}_1 \\ X_{2i} - \bar{X}_2 \end{bmatrix}, \quad (2.1)$$

де S_N – коваріаційна матриця:

$$S_N = \frac{1}{N}\sum_{i=1}^{N} \begin{bmatrix} Y_i - \bar{Y} \\ X_{1i} - \bar{X}_1 \\ X_{2i} - \bar{X}_2 \end{bmatrix} \begin{bmatrix} Y_i - \bar{Y} \\ X_{1i} - \bar{X}_1 \\ X_{2i} - \bar{X}_2 \end{bmatrix}^T . \quad (2.2)$$

У таблиці 2.1 представлено набір даних, де CBO – змінна X_1, WMC – змінна X_2, RFC – змінна Y, D^2 – розрахований квадрат відстані Махаланобіса.

Таблиця 2.1 – Набір даних

#	URL	RFC	CBO	WMC	D^2
1	2	3	4	5	6
1	https://github.com/shebnik/BingNUOS-Admin	5,4744	0,1923	10,5000	6,6294
2	https://github.com/shebnik/poprey_app	5,5354	0,1616	7,9192	1,4363
3	https://github.com/shebnik/PersonalFinanceApp	7,3125	0,2250	7,7000	1,6808
4	https://github.com/nafishahmeddev/fintracker	3,7111	0,2444	5,9556	1,1880
5	https://github.com/robert-muriithi/Game-Flix	3,0314	1,1220	3,0348	9,3964
6	https://github.com/fregie/pho	16,4706	0,5882	18,7206	9,2556
7	https://github.com/RivaanRanawat/flutter_twitter_clone	4,4098	0,1967	6,2131	0,8783
8	https://github.com/HypeTeqSoftware/FitnessApp	5,6515	0,2879	4,8485	2,2945
9	https://github.com/R8bert/busappschedule	3,0000	0,3571	4,0714	0,6164
10	https://github.com/AbdQader/flutter_grocery_app	3,3220	0,2881	4,6780	0,6206
11	https://github.com/Sameera-Perera/Flutter-TDD-Clean-Architecture-E-Commerce-App	2,3881	1,0699	2,4755	8,8146
12	https://github.com/Roaa94/wallet_app_workshop	2,7500	0,2250	3,5500	0,9716
13	https://github.com/mohamadayash22/flutter-movie-app	2,4494	0,3034	3,0056	0,9413
14	https://github.com/mbcorona/flutter_cars_app	3,1667	0,2500	2,8333	1,4904
15	https://github.com/aymanattieh77/fashion-app	3,0945	0,3727	3,6063	0,6516
16	https://github.com/AbdQader/flutter_ecommerce_app	3,1957	0,1957	4,4783	0,9738
17	https://github.com/iManYarahmadi/TravelApp	1,9091	0,1364	2,4091	1,7949
18	https://github.com/flutter/pinball	1,6846	0,3128	2,0051	1,4021
19	https://github.com/AmirBayat0/Flutter-Travel-App	2,2381	0,3333	2,5714	1,0940
20	https://github.com/tanvir-robin/weather-app	5,9677	0,3226	5,0645	2,4647
21	https://github.com/MasteerRui/WeatherApp-	8,5769	0,4615	6,8462	5,8708

	Flutter				
22	https://github.com/NobodyForNothing/blood-pressure-monitor-fl	6,8413	0,3386	10,8466	2,9715
23	https://github.com/tusharhow/movie-app	2,6250	0,2500	2,5000	1,3971
24	https://github.com/SamarthMovaliya/ecommerce_app_flutter_firebase	3,4688	0,4688	4,7500	0,6057

Продовження таблиці 2.1

1	2	3	4	5	6
25	https://github.com/Saffar-Org/saffar_app	3,0407	0,5366	4,5528	1,2209
26	https://github.com/dudecoderr/Electronics_E-Commerce_App_in_flutter	2,6667	0,3333	4,3333	1,0277
27	https://github.com/dicky7/ngandika_app	3,6408	0,6204	3,9388	1,4426
28	https://github.com/suleymangunes/TradingView-app	2,2532	0,1013	2,8101	1,8757
29	https://github.com/Re4ch-Jay/Meditation_App	2,8750	0,1250	2,5000	2,2555
30	https://github.com/darkmoonight/Rain	4,3011	0,2796	9,6452	7,9978
31	https://github.com/wwwwww3q/social_app	6,6859	0,4660	7,4241	0,2925
32	https://github.com/EmirBashiri/Flutter-E-commerce-X	1,7267	0,5467	2,4333	1,9971
33	https://github.com/alireza4585/Flutter-Shopping-App	3,2759	0,4483	3,2414	1,0173
34	https://github.com/antonkarliner/timer-coffee	9,7673	1,4752	9,8960	15,6157
35	https://github.com/zaaii/my_comic	2,0426	0,5000	2,3830	1,5701
36	https://github.com/Hamad-Anwar/Task-Sync-Pro-V2	4,8448	0,2069	3,8966	2,7547
37	https://github.com/nightFuryman/flutter-feed-app	5,4571	0,4571	7,5714	0,3377
38	https://github.com/dreautall/waterfly-iii	7,4476	0,3991	11,6328	3,2147
39	https://github.com/IsaiasCuvula/quote_generator	3,0196	0,0523	3,7386	2,0327
40	https://github.com/dstark5/Openlib	4,1957	0,2391	6,3696	0,9811
41	https://github.com/javedahm3d/instagram-clone-using-flutter-and-firebase	3,3158	0,3421	4,0263	0,5204
42	https://github.com/Ruslanbek0809/Yummify	5,2857	0,7755	5,4490	2,3770
43	https://github.com/adityar224/FlutterMarkdownEditor	8,2727	0,2727	9,5455	1,2568
44	https://github.com/cem256/manga_clean_arch	2,3304	0,2087	3,1478	1,2023
45	https://github.com/Shashank02051997/Agent X-Flutter	3,3158	0,4211	3,3158	0,9271
46	https://github.com/riccardocescon/clean_chess	3,8899	0,5505	4,0298	1,0406
47	https://github.com/kronos-et-al/MensaApp	4,7753	0,9955	6,2809	5,3896
48	https://github.com/kaina404/FlutterDouBan	5,6973	0,5747	7,2107	0,3603
49	https://github.com/simplezhli/flutter_deer	5,2277	0,3508	6,7754	0,1143
50	https://github.com/iampawan/FlutterWhatsAp	1,5455	0,2727	1,9091	1,5100

	pClone				
51	https://github.com/Predidit/Kazumi	7,4366	0,2676	9,0845	0,7950
52	https://github.com/canxin121/app_rhyme	12,0263	0,7684	12,4632	4,8952
53	https://github.com/Predidit/oneAnime	8,2170	0,2642	10,3585	1,2509
54	https://github.com/kangpeiqin/bilivideo_down	5,7541	0,1967	5,6393	1,7646

Продовження таблиці 2.1

1	2	3	4	5	6
55	https://github.com/dagmawibabi/ScholArxiv	7,0857	0,4000	7,9143	0,3986
56	https://github.com/Ferry-200/coriander_player	6,3709	0,3286	7,7136	0,2158
57	https://github.com/lyming99/wenznote	14,0101	0,5263	17,8421	6,1731
58	https://github.com/JHubi1/ollama-app	20,4500	0,5000	26,6500	21,1618
59	https://github.com/mirarr-app/mirarr	10,1333	0,3500	12,5167	2,0271
60	https://github.com/KNKPA/KNKPAnime	7,5676	0,2703	7,1081	2,7615
61	https://github.com/mu-fazil-vk/FluxTube	4,8231	0,5238	7,4354	1,2063
62	https://github.com/raj457036/copycat_clipboard	3,2616	0,1512	6,5407	3,3669
63	https://github.com/yaoxieyoulei/my_tv	6,0000	0,3146	5,9213	1,1534
64	https://github.com/K3vinb5/Unyo	9,1319	0,3194	10,6458	1,4502
65	https://github.com/zetaloop/keyviz	6,3419	0,4444	9,6496	1,5905
66	https://github.com/fanglu0411/sgs	9,9240	0,5447	13,6550	2,7765
67	https://github.com/brandonp2412/Flexify	7,0761	0,8696	8,0543	2,7149
68	https://github.com/uiYzzi/Yolx	11,5517	0,2759	13,4828	3,8845
69	https://github.com/Flutterando/yuno	4,6019	0,4757	5,6602	0,2032
70	https://github.com/poppingmoon/aria	9,7293	1,4622	15,3831	21,3625
71	https://github.com/Gambley1/flutter-instagram-offline-first-clone	3,2875	0,5000	5,0600	1,0590
72	https://github.com/hm21/pro_image_editor	6,9725	0,7879	11,8595	7,0250
73	https://github.com/JICA98/DailyAL	13,6362	0,3803	18,3357	7,7101
74	https://github.com/maelchiotti/LocalMaterialNotes	5,0450	0,2252	7,5045	1,1424
75	https://github.com/mahdinazmi/Spotify-Clone-With-Flutter	2,2532	0,2785	2,4810	1,2022
76	https://github.com/Sivan22/otzaria	7,9542	0,6260	9,0840	0,7847
77	https://github.com/VGVentures/io_crossword	2,1347	0,2489	2,9452	1,1622
78	https://github.com/Hamad-Anwar/Flutter-Music-Player-App-With-BLoc	2,7571	0,4429	3,1429	0,9423
79	https://github.com/VaibhavCodeClub/learn	3,9352	0,3056	3,9815	0,8286
80	https://github.com/kananinirav/Indian-IPTV-App	5,8750	0,6250	4,8750	2,8507
81	https://github.com/mahdinazmi/Ecommerce-App-With-Flutter	2,7353	0,2588	2,6588	1,2952
82	https://github.com/AquaWallet/aqua-wallet	3,9360	0,1919	6,1721	1,3366
83	https://github.com/kra-mo/sly	6,6923	0,5769	10,5000	2,7811
84	https://github.com/easybangumiorg/EasyMyGo	4,7091	0,4424	5,3030	0,2603
85	https://github.com/jacky4631/sfb	10,0074	0,7098	11,7024	1,8977
86	https://github.com/surtecha/karman	8,5824	0,2527	9,6593	1,6992
87	https://github.com/rohitsangwan01/uni_control_hub	5,2714	0,0643	7,0071	1,8843

Продовження таблиці 2.1

88	https://github.com/Predidit/BiliNeo	14,5865	0,4856	15,8125	7,2769
89	https://github.com/JGeek00/linkdy	2,8205	0,0256	6,7265	5,9465
90	https://github.com/onlyGuo/chatgpt_desktop	5,2000	0,1200	5,4600	1,7884
91	https://github.com/chicring/Themby	3,9424	0,1605	5,9218	1,2989
92	https://github.com/GalenBlabla/Hiddify-with-V2board	4,6328	0,1563	7,2184	1,7577
93	https://github.com/thelooter/labplus_for_gitlab	6,2639	0,1715	7,2559	1,1673
94	https://github.com/sunarya-thito/shadcn_flutter	6,3994	1,0137	10,3155	7,9586
95	https://github.com/brandonp2412/FitBook	12,0483	0,9614	13,5700	5,3993
96	https://github.com/judemont/reciper	5,6129	0,4516	5,7742	0,5793
97	https://github.com/mkobuolys/ai-photo-scavenger-hunt	2,1304	0,7391	2,8261	3,4250
98	https://github.com/mumu-lhl/Ciyue	4,1296	0,5000	4,7963	0,4750
99	https://github.com/paulocode/multimesh	15,2977	0,8795	13,9864	14,5698
100	https://github.com/yubo725/flutter-osc	9,1455	0,5273	7,6727	5,5709

Формула для асиметрії та ексцесу Мардіа [29] виглядає наступним чином:

$$\beta_{1,k} = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left[(\mathrm{X}_i - \bar{\mathrm{X}})^T \mathrm{S}_N^{-1}(\mathrm{X}_j - \bar{\mathrm{X}})\right]^3, \quad (2.3)$$

$$\beta_{2,k} = \frac{1}{N}\sum_{i=1}^{N}\left[(\mathrm{X}_i - \bar{\mathrm{X}})^T \mathrm{S}_N^{-1}(\mathrm{X}_i - \bar{\mathrm{X}})\right]^2, \quad (2.4)$$

де $\mathrm{X} - k \times 1$ вектор випадкових величин, $\mathrm{X} = (X_1, X_2, \dots, X_k)^T$,

$\mathrm{S}_N -$ зміщена вибіркова коваріаційна матриця X.

Для $\beta_{1,k}$ потрібно обчислити тестову статистику [29]:

$$\frac{N}{6}\beta_{1,k}, \quad (2.5)$$

де наближений розподіл χ^2 з $k(k+1)(k+2)/6$ ступенями свободи та рівнем значущості $\alpha = 0{,}005$.

Для $\beta_{2,k}$ у якості тестової статистики $z_{2,k}$ використовуємо $1-\alpha$ квантиль нормального закону розподілу з математичним сподіванням $k(k+2)$ та дисперсією $8k(k+2)/N$.

Щоб перевірити відхилення розподілу від нормального, потрібно порівняти значення статистики (2.5) з квантилем $\chi^2_{(k+1)(k+2)/6,\alpha}$, а $\beta_{2,k}$ з $z_{2,k}$. Якщо значення (2.5) більше за квантиль $\chi^2_{(k+1)(k+2)/6,\alpha}$ або, якщо значення $\beta_{2,k}$ більше за $z_{2,k}$, то багатовимірний розподіл даних не є нормальним.

Значення асиметрії Мардія (2.3) було обчислено як 109,6260, і порівняно з критичним значенням 25,1882. Розраховане значення ексцесу Мардія (2.4) становить 23,9017, і воно порівняне з критичним значенням 17,8217.

Результати тесту показали, що розподіл даних не відповідає гаусівському [31], оскільки асиметрія та ексцес більше за критичне значення. Дані мають значну асиметрію і не відповідають нормальному розподілу. Лінійна регресія не підходить для моделювання, оскільки її припущення про лінійність та нормальний розподіл залишків не виконуються в цій вибірці. У зв'язку з цим було вирішено застосувати нелінійну регресійну модель із нормалізуючим перетворенням у вигляді десяткового логарифму, що дозволяє достовірно описати взаємозалежності між метриками.

Значення асиметрії Мардія (2.3) для нормалізованих даних було обчислено як 22,7641, і порівняно з критичним значенням 25,1881. Розраховане значення ексцесу Мардія (2.4) становить 16,0161, і воно порівняне з критичним значенням 17,8216.

Використання десяткового логарифму як нормалізуючого перетворення допомогло врахувати асиметричний розподіл, що забезпечило рівномірній розподіл даних. Далі потрібно знайти і вилучити викиди із вихідних даних:

1. Проводимо нормалізацію даних за допомогою десяткового логарифмування значень метрик.

2. Розраховуємо квадрат відстані Махаланобіса (2.1) на нормалізованих даних.

3. Розраховуємо коваріаційну матрицю S_N для 3 змінних (2.2).

4. Знаходимо зворотну матрицю ${S_N}^{-1}$.

5. Для кожного спостереження і розраховуємо квадрат відстані Махаланобіса на нормалізованих даних (2.1).

6. Обчислюємо тестову статистику (TS) для кожного спостереження [32]:

$$T_i = (N-k)Nd_i^2/\big((N^2-1)k\big), \tag{2.6}$$

7. Обчислюємо квантіль розподілення Фішера $F_{\text{крит}}$ як [33]:

$$F_{\text{крит}} = F(\alpha, 3, n-3), \tag{2.7}$$

де α – рівень значущості 0,005,

3 – кількість ступенів вільності,

n – розмір вибірки.

8. Якщо $T_i > F_{\text{крит}}$, то і–те спостереження є викидом і його вилучаємо з вибірки.

9. Повторяємо крок 2 – 8 поки не буде вилучено усі викиди.

10. Далі потрібно побудувати рівняння нелінійної регресії (1.1) та розрахувати інтервал прогнозування який обчислюється як:

$$10^{\left(\hat{Z}_Y \pm t_{\frac{\alpha}{2},\nu} S_{Z_Y}\left\{1+\frac{1}{N}+(\mathbf{z}_X^+)^T \mathbf{S}_Z^{-1}(\mathbf{z}_X^+)\right\}^{\frac{1}{2}}\right)}, \tag{2.8}$$

де $\hat{Z}_Y$ – прогнозне значення, отримане з рівняння лінійної регресії для нормалізованих даних;

$Z_1 = \log_{10}(X_1)$ та $Z_2 = \log_{10}(X_2)$ – нормалізовані значення метрик CBO та WMC відповідно;

$t_{\alpha/2,v}$ – квантиль розподілу Стьюдента з рівнем значущості $\alpha/2$ та $v = N - 3$ ступенями вільності;

$\mathbf{z}_X^+$ – вектор з компонентами $(Z_{1_i} - \bar{Z}_1, Z_{2_i} - \bar{Z}_2)$ для і-го спостереження;

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

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

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

де $S_{Z_q Z_r} = \sum_{i=1}^{N} \left[(Z_{q_i} - \bar{Z}_q)(Z_{r_i} - \bar{Z}_r)\right], q, r = 1,2.$

Якщо є точки даних які виходять за інтервал прогнозування то потрібно вилучити їх із вибірки. Повторяємо крок 10, поки не буде вилучено усі викиди.

Нижче представлено видалення викидів.

Ітерація 1:

– 89-й застосунок: 6,7852 > 4,5508 (TS > $F_{\text{крит}}$);

Ітерація 2:

– 21-й застосунок: 8,5769 > 8,4256 (Y більше за верхню межу);

– 30-й застосунок: 4,3011 < 4,8869 (Y менше за нижню межу);

– 36-й застосунок: 4,8448 > 4.7461 (Y більше за верхню межу);

– 62-й застосунок: 3,2616 < 3,2627 (Y менше за нижню межу).

Ітерація 3:

– 1-й застосунок: 5,4744 < 5,6002 (Y менше за нижню межу);

– 100-й застосунок: 9,1455 > 8,9503 (Y більше за верхню межу);

Ітерація 4:

– 8-й застосунок: 5,6515 > 5,6407 (Y більше за верхню межу);

– 20-й застосунок: 5,9677 > 5,8936 (Y більше за верхню межу);

– 80-й застосунок: 5,8750 > 5,8285 (Y більше за верхню межу).

Ітерація 5:

– 99-й застосунок: 15,2977 > 15,2093 (Y більше за верхню межу).

Після видалення викидів за 5 ітерацій залишилась вибірка із 89 застосунків. Більше викидів не виявлено, тому можна зробити висновок, що попередня обробка даних завершена.

2.2 Побудова моделі для оцінювання якості застосунків, що створюються на платформі Flutter

Після попередньої обробки даних та видалення викидів було сформовано вибірку з 89-и застосунків. Дані були розділені на навчальну та тестову вибірки для побудови й оцінки моделі: 60% (53-х застосунків) склали навчальну вибірку для побудови моделі, а решта 40% (36-и застосунків) – тестову вибірку для перевірки якості моделі. Під час цього розподілу було забезпечено, щоб навчальна вибірка включала мінімальні та максимальні значення метрик, а тестова вибірка містила значення, близькі до граничних показників. У таблиці 2.2 представлено вибірку для побудови моделі, а у таблиці 2.3 вибірку для тестування моделі.

Таблиця 2.2 – Вибірка для побудови моделі

#	URL	RFC	CBO	WMC
1	2	3	4	5
1	https://github.com/iampawan/FlutterWhatsAppClone	1,5455	0,2727	1,9091
2	https://github.com/JHubi1/ollama-app	20,4500	0,5000	26,6500
3	https://github.com/IsaiasCuvula/quote_generator	3,0196	0,0523	3,7386
4	https://github.com/antonkarliner/timer-coffee	9,7673	1,4752	9,8960
5	https://github.com/nafishahmeddev/fintracker	3,7111	0,2444	5,9556
6	https://github.com/kangpeiqin/bilivideo_down	5,7541	0,1967	5,6393
7	https://github.com/R8bert/busappschedule	3,0000	0,3571	4,0714
8	https://github.com/poppingmoon/aria	9,7293	1,4622	15,3831
9	https://github.com/iManYarahmadi/TravelApp	1,9091	0,1364	2,4091

Продовження таблиці 2.2

1	2	3	4	5
10	https://github.com/dudecoderr/Electronics_E-Commerce_App_in_flutter	2,6667	0,3333	4,3333
11	https://github.com/AquaWallet/aqua-wallet	3,9360	0,1919	6,1721
12	https://github.com/aymanattieh77/fashion-app	3,0945	0,3727	3,6063
13	https://github.com/canxin121/app_rhyme	12,0263	0,7684	12,4632
14	https://github.com/dagmawibabi/ScholArxiv	7,0857	0,4000	7,9143
15	https://github.com/flutter/pinball	1,6846	0,3128	2,0051
16	https://github.com/wwwwww3q/social_app	6,6859	0,4660	7,4241
17	https://github.com/kronos-et-al/MensaApp	4,7753	0,9955	6,2809
18	https://github.com/Sameera-Perera/Flutter-TDD-Clean-Architecture-E-Commerce-App	2,3881	1,0699	2,4755
19	https://github.com/tusharhow/movie-app	2,6250	0,2500	2,5000
20	https://github.com/suleymangunes/TradingView-app	2,2532	0,1013	2,8101
21	https://github.com/lyming99/wenznote	14,0101	0,5263	17,8421
22	https://github.com/uiYzzi/Yolx	11,5517	0,2759	13,4828
23	https://github.com/Roaa94/wallet_app_workshop	2,7500	0,2250	3,5500
24	https://github.com/SamarthMovaliya/ecommerce_app_flutter_firebase	3,4688	0,4688	4,7500
25	https://github.com/javedahm3d/instagram-clone-using-flutter-and-firebase	3,3158	0,3421	4,0263
26	https://github.com/judemont/reciper	5,6129	0,4516	5,7742
27	https://github.com/dstark5/Openlib	4,1957	0,2391	6,3696
28	https://github.com/K3vinb5/Unyo	9,1319	0,3194	10,6458
29	https://github.com/surtecha/karman	8,5824	0,2527	9,6593
30	https://github.com/Predidit/BiliNeo	14,5865	0,4856	15,8125
31	https://github.com/alireza4585/Flutter-Shopping-App	3,2759	0,4483	3,2414
32	https://github.com/sunarya-thito/shadcn_flutter	6,3994	1,0137	10,3155
33	https://github.com/dreautall/waterfly-iii	7,4476	0,3991	11,6328
34	https://github.com/AmirBayat0/Flutter-Travel-App	2,2381	0,3333	2,5714
35	https://github.com/Predidit/Kazumi	7,4366	0,2676	9,0845
36	https://github.com/Flutterando/yuno	4,6019	0,4757	5,6602
37	https://github.com/Ferry-200/coriander_player	6,3709	0,3286	7,7136
38	https://github.com/JICA98/DailyAL	13,6362	0,3803	18,3357
39	https://github.com/mu-fazil-vk/FluxTube	4,8231	0,5238	7,4354
40	https://github.com/maelchiotti/LocalMaterialNotes	5,0450	0,2252	7,5045
41	https://github.com/Ruslanbek0809/Yummify	5,2857	0,7755	5,4490
42	https://github.com/fanglu0411/sgs	9,9240	0,5447	13,6550
43	https://github.com/adityar224/FlutterMarkdownEditor	8,2727	0,2727	9,5455
44	https://github.com/Sivan22/otzaria	7,9542	0,6260	9,0840
45	https://github.com/brandonp2412/FitBook	12,0483	0,9614	13,5700
46	https://github.com/mohamadayash22/flutter-movie-app	2,4494	0,3034	3,0056

47	https://github.com/Shashank02051997/AgentX-Flutter	3,3158	0,4211	3,3158

Продовження таблиці 2.2

1	2	3	4	5
48	https://github.com/nightFuryman/flutter-feed-app	5,4571	0,4571	7,5714
49	https://github.com/zetaloop/keyviz	6,3419	0,4444	9,6496
50	https://github.com/EmirBashiri/Flutter-E-commerce-X	1,7267	0,5467	2,4333
51	https://github.com/shebnik/PersonalFinanceApp	7,3125	0,2250	7,7000
52	https://github.com/dicky7/ngandika_app	3,6408	0,6204	3,9388
53	https://github.com/simplezhli/flutter_deer	5,2277	0,3508	6,7754

Таблиця 2.3 – Вибірка для тестування моделі

#	URL	RFC	CBO	WMC
1	2	3	4	5
1	https://github.com/rohitsangwan01/uni_control_hub	5,2714	0,0643	7,0071
2	https://github.com/mirarr-app/mirarr	10,1333	0,3500	12,5167
3	https://github.com/Saffar-Org/saffar_app	3,0407	0,5366	4,5528
4	https://github.com/mbcorona/flutter_cars_app	3,1667	0,2500	2,8333
5	https://github.com/thelooter/labplus_for_gitlab	6,2639	0,1715	7,2559
6	https://github.com/VaibhavCodeClub/learn	3,9352	0,3056	3,9815
7	https://github.com/kaina404/FlutterDouBan	5,6973	0,5747	7,2107
8	https://github.com/Re4ch-Jay/Meditation_App	2,8750	0,1250	2,5000
9	https://github.com/Hamad-Anwar/Flutter-Music-Player-App-With-BLoc	2,7571	0,4429	3,1429
10	https://github.com/kra-mo/sly	6,6923	0,5769	10,5000
11	https://github.com/cem256/manga_clean_arch	2,3304	0,2087	3,1478
12	https://github.com/mahdinazmi/Ecommerce-App-With-Flutter	2,7353	0,2588	2,6588
13	https://github.com/onlyGuo/chatgpt_desktop	5,2000	0,1200	5,4600
14	https://github.com/Predidit/oneAnime	8,2170	0,2642	10,3585
15	https://github.com/robert-muriithi/Game-Flix	3,0314	1,1220	3,0348
16	https://github.com/mumu-lhl/Ciyue	4,1296	0,5000	4,7963
17	https://github.com/mkobuolys/ai-photo-scavenger-hunt	2,1304	0,7391	2,8261
18	https://github.com/chicring/Themby	3,9424	0,1605	5,9218
19	https://github.com/AbdQader/flutter_grocery_app	3,3220	0,2881	4,6780
20	https://github.com/hm21/pro_image_editor	6,9725	0,7879	11,8595
21	https://github.com/Gambley1/flutter-instagram-offline-first-clone	3,2875	0,5000	5,0600
22	https://github.com/yaoxieyoulei/my_tv	6,0000	0,3146	5,9213
23	https://github.com/mahdinazmi/Spotify-Clone-With-Flutter	2,2532	0,2785	2,4810
24	https://github.com/VGVentures/io_crossword	2,1347	0,2489	2,9452
25	https://github.com/NobodyForNothing/blood-pressure-monitor-fl	6,8413	0,3386	10,8466
26	https://github.com/fregie/pho	16,4706	0,5882	18,7206

27	https://github.com/shebnik/poprey_app	5,5354	0,1616	7,9192

Продовження таблиці 2.3

1	2	3	4	5
28	https://github.com/brandonp2412/Flexify	7,0761	0,8696	8,0543
29	https://github.com/GalenBlabla/Hiddify-with-V2board	4,6328	0,1563	7,2184
30	https://github.com/KNKPA/KNKPAnime	7,5676	0,2703	7,1081
31	https://github.com/easybangumiorg/EasyMyGo	4,7091	0,4424	5,3030
32	https://github.com/riccardocescon/clean_chess	3,8899	0,5505	4,0298
33	https://github.com/zaaii/my_comic	2,0426	0,5000	2,3830
34	https://github.com/AbdQader/flutter_ecommerce_app	3,1957	0,1957	4,4783
35	https://github.com/jacky4631/sfb	10,0074	0,7098	11,7024
36	https://github.com/RivaanRanawat/flutter_twitter_clone	4,4098	0,1967	6,2131

Щоб побудувати нелінійну модель, спочатку слід застосувати рівняння лінійної регресії для нормалізованих даних:

$$\hat{z}_y = \beta_0 + \beta_1 Z_{x_1} + \beta_2 Z_{x_2}, \quad (2.9)$$

Z_{x_1} – нормалізована змінна (CBO);

Z_{x_2} – нормалізована змінна (WMC);

$\beta_0, \beta_1, \beta_2$ – коефіцієнти регресії.

Оцінка параметрів моделі β_0, β_1, β_2 буде здійснена звичайним методом найменших квадратів (МНК) [34]:

$$\hat{\beta} = (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}\mathrm{y}, \quad (2.10)$$

де $\hat{\beta}$ – звичайна оцінка найменших квадратів;

X – змінна матричного регресора X;

T – транспонування матриці;

y – вектор значення змінної відгуку.

За допомогою зворотного перетворення отримаємо модель нелінійної регресії (1.1). Після обчислень були отримані наступні значення коефіцієнтів: $\beta_0 = -0{,}0372$, β_1 = 0,0314, β_2 = 0,9467, ε – гаусівська випадкова величина з

нульовим математичним сподіванням та середньоквадратичним відхиленням, оцінка якого дорівнює 0,0642.

Таким чином, остаточна модель у вигляді рівняння нелінійної регресії має вигляд:

$$\hat{Y} = 10^{-0,0372} X_1^{\,0,0314} X_2^{\,0,9467}. \quad (2.11)$$

Модель працює у межах для метрик RFC [1,5455 – 20,45], CBO [0,0523 – 1,4752] та WMC [1,9091 – 26,65]. Крім того, для введених нормалізованих даних розраховується квадрат відстані Махаланобіса, який не повинен перевищувати критичне значення хі-квадрат [35] $\chi^2_{0.005,3} = 12,8382$.

Розраховуємо параметри якості для побудованої нелінійної регресійної моделі. Аналіз якості та перевірка адекватності побудованої моделі буде виконана за допомогою критеріїв якості R^2, *MMRE* та *PRED*. Результати оцінки для навчальної та тестової вибірок:

– Навчальна вибірка: R^2 = 0,9416, *MMRE* = 0,1214, *PRED(0.25)* = 0,9245.

– Тестова вибірка: R^2 = 0,8994, *MMRE* = 0,1428, *PRED(0.25)* = 0,8611.

Побудована нелінійна регресійна модель показує високу якість як на навчальній, так і на тестовій вибірках. Високі значення R^2, низькі значення *MMRE* і високий рівень *PRED(0.25)* підтверджують, що модель має високу якість і може використовуватись для оцінювання якості Flutter-застосунків.

Для оцінювання якості Flutter-застосунків на основі отриманої регресійної моделі запропоновано використання довірчих та прогнозних інтервалів [2]. Цей підхід дозволяє не лише отримати точкову оцінку метрики RFC, але й визначити діапазон можливих значень з заданою ймовірністю, що підвищує надійність оцінювання.

Прогнозний інтервал для нелінійної регресії з урахуванням нормалізації даних за допомогою десяткового логарифму розраховується за формулою (2.8).

Довірчий інтервал для нелінійної регресії розраховується аналогічно прогнозному, але без доданка «1» у фігурних дужках:

$$10^{\left(\hat{Z}_Y \pm t_{\frac{\alpha}{2},\nu} S_{Z_Y}\left\{\frac{1}{N}+\left(\mathbf{z}_X^+\right)^T \mathbf{S}_Z^{-1}\left(\mathbf{z}_X^+\right)\right\}^{\frac{1}{2}}\right)} \quad (2.12)$$

Для розрахунку інтервалів використовуючи нормалізовані дані для побудови моделі було отримано квантиль розподілу Стьюдента: 2,0086, залишкове стандартне відхилення (2.9): 0,0642, та обернена коваріаційна матриця: $\begin{pmatrix} 0{,}3066 & -0{,}1028 \\ -0{,}1028 & 0{,}2835 \end{pmatrix}$. У таблиці 2.4 представлено значення Y (RFC), передбачені значення $\hat{Y}$ та розраховані прогнозні та довірчі інтервали для кожного застосунку.

Таблиця 2.4 – Передбачені значення та інтервали

#	Y	Ŷ	Довірчий нижня	Довірчий верхня	Прогнозний нижня	Прогнозний верхня
1	2	3	4	5	6	7
1	1,5455	1,6253	1,4902	1,7727	1,1930	2,2143
2	20,4500	20,0938	18,1298	22,2706	14,6773	27,5092
3	3,0196	2,9157	2,5333	3,3557	2,0995	4,0491
4	9,7673	8,1379	7,3744	8,9804	5,9525	11,1255
5	3,7111	4,7556	4,5166	5,0073	3,5187	6,4274
6	5,7541	4,4856	4,2178	4,7703	3,3127	6,0737
7	3,0000	3,3574	3,1950	3,5281	2,4850	4,5362
8	9,7293	12,3527	11,1548	13,6791	9,0254	16,9065
9	1,9091	1,9821	1,8116	2,1687	1,4536	2,7028
10	2,6667	3,5538	3,3899	3,7258	2,6314	4,7997
11	3,9360	4,8819	4,5786	5,2054	3,6035	6,6140
12	3,0945	2,9972	2,8362	3,1672	2,2162	4,0534
13	12,0263	9,9184	9,2626	10,6207	7,3142	13,4499
14	7,0857	6,3217	6,0525	6,6029	4,6834	8,5331
15	1,6846	1,7100	1,5709	1,8614	1,2558	2,3283
16	6,6859	5,9790	5,7261	6,2430	4,4297	8,0701
17	4,7753	5,2267	4,8301	5,6559	3,8446	7,1056
18	2,3881	2,1697	1,9255	2,4450	1,5757	2,9878
19	2,6250	2,0923	1,9461	2,2495	1,5415	2,8399

Продовження таблиці 2.4

1	2	3	4	5	6	7
20	2,2532	2,2718	2,0540	2,5127	1,6606	3,1082
21	14,0101	13,7657	12,7241	14,8924	10,1263	18,7130
22	11,5517	10,3467	9,5834	11,1709	7,6152	14,0581
23	2,7500	2,9064	2,7366	3,0868	2,1470	3,9344
24	3,4688	3,9183	3,7309	4,1151	2,9004	5,2934
25	3,3158	3,3177	3,1565	3,4871	2,4555	4,4827
26	5,6129	4,7083	4,5094	4,9160	3,4883	6,3550
27	4,1957	5,0646	4,8009	5,3427	3,7460	6,8472
28	9,1319	8,3114	7,8336	8,8183	6,1410	11,2488
29	8,5824	7,5248	7,0593	8,0211	5,5546	10,1940
30	14,5865	12,2475	11,3930	13,1661	9,0236	16,6232
31	3,2759	2,7250	2,5547	2,9067	2,0112	3,6921
32	6,3994	8,3649	7,7448	9,0347	6,1560	11,3665
33	7,4476	9,1025	8,5874	9,6485	6,7268	12,3173
34	2,2381	2,1684	2,0193	2,3284	1,5980	2,9423
35	7,4366	7,1130	6,7068	7,5437	5,2560	9,6261
36	4,6019	4,6278	4,4251	4,8399	3,4278	6,2479
37	6,3709	6,1319	5,8566	6,4201	4,5412	8,2799
38	13,6362	13,9825	12,8498	15,2150	10,2700	19,0370
39	4,8231	6,0096	5,7408	6,2910	4,4507	8,1146
40	5,0450	5,9039	5,5590	6,2703	4,3614	7,9921
41	5,2857	4,5332	4,2369	4,8502	3,3436	6,1461
42	9,9240	10,6979	10,0266	11,4142	7,8953	14,4954
43	8,2727	7,4586	7,0239	7,9202	5,5100	10,0964
44	7,9542	7,3049	6,9229	7,7080	5,4029	9,8765
45	12,0483	10,8264	10,0003	11,7207	7,9627	14,7200
46	2,4494	2,5061	2,3543	2,6676	1,8504	3,3940
47	3,3158	2,7787	2,6125	2,9555	2,0521	3,7626
48	5,4571	6,0876	5,8303	6,3563	4,5102	8,2167
49	6,3419	7,6521	7,2841	8,0386	5,6639	10,3380
50	1,7267	2,0902	1,9154	2,2809	1,5340	2,8480
51	7,3125	6,0493	5,6911	6,4299	4,4680	8,1902
52	3,6408	3,3107	3,0940	3,5426	2,4418	4,4887
53	5,2277	5,4345	5,2111	5,6675	4,0270	7,3340

Таким чином ці інтервали дозволять оцінювати якість програмного забезпечення, створеного за допомогою платформи Flutter.

2.3 Оцінювання якості застосунків

На основі довірчих та прогнозних інтервалів встановлено наступні критерії оцінювання якості Flutter-застосунків:

1. Середній рівень якості:

– Фактичне значення RFC знаходиться в межах довірчого інтервалу, тобто не менше нижньої межі довірчого інтервалу та не перевищує його верхню межу;

– Свідчить про типову складність коду для даної комбінації метрик CBO та WMC.

2. Висока якість:

– Фактичне значення RFC не менше нижньої межі прогнозного інтервалу, але менше нижньої межі довірчого інтервалу;

– Вказує на меншу складність коду, ніж очікувалося.

3. Низька якість:

– Фактичне значення RFC перевищує верхню межу довірчого інтервалу;

– Свідчить про підвищену складність коду.

Такий підхід до оцінювання дозволяє:

– Враховувати статистичну невизначеність при оцінці якості;

– Виявляти застосунки з нетиповою складністю коду;

– Приймати обґрунтовані рішення щодо необхідності рефакторингу.

За результатами застосування запропонованої в [2] методики до вибірки із 89 застосунків без викидів отримано наступний розподіл застосунків:

– Високий рівень якості: 74,2% застосунків;

– Середній рівень якості: 21,3% застосунків;

– Низький рівень якості: 2,2% застосунків.

Отже, запропонована методика [2] на основі довірчих та прогнозних інтервалів дозволяє оцінювати якість Flutter-застосунків.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ МАТЕМАТИЧНОЇ МОДЕЛІ

3.1 Постановка задачі та вимоги до програмного забезпечення

Виходячи з потреби розробки інструменту для достовірного оцінювання застосунків на платформі Flutter, сформулюємо наступну постановку задачі: необхідно створити програмне забезпечення, яке на основі спеціалізованої математичної моделі дозволить оцінювати якість Flutter-застосунків. Модель має враховувати особливості Flutter, застосовуючи метрики RFC (Response for Class), CBO (Coupling Between Object Classes) та WMC (Weighted Methods per Class), щоб забезпечити достовірне прогнозування показників якості.

На основі проведеного аналізу предметної області та математичної моделі було сформульовано вимоги до програмного забезпечення для оцінювання якості Flutter-застосунків. Розроблюване програмне забезпечення отримало назву «FlutterStats».

Програмне забезпечення «FlutterStats» призначене для автоматизації процесу оцінювання якості застосунків, що створюються з використанням фреймворку Flutter. Експлуатаційне призначення системи полягає у полегшенні процесу планування та управління розробкою програмних продуктів шляхом автоматизації розрахунку прогнозних значень якості проектів. Функціональність програмного забезпечення для оцінювання якості Flutter-застосунків, має забезпечувати виконання двох основних завдань:

– Побудова нової моделі оцінювання якості — створення регресійної моделі на основі заданих даних (RFC, CBO, WMC), що дозволяє прогнозувати якість застосунків.

– Оцінювання якості на основі існуючої моделі — перевірка якості конкретного застосунку за допомогою вже побудованої моделі.

Функціональність програмного забезпечення розподілена на два окремі сценарії використання.

Сценарій 1: Побудова нової моделі.

Завантаження даних:

– Імпорт набору даних у форматі CSV, що містить метрики Flutter-проектів (RFC, CBO, WMC);

– Попередня обробка даних, включаючи нормалізацію та видалення викидів;

– Розподіл на навчальну (60%) та тестову (40%) вибірки.

Побудова моделі:

– Створення лінійної регресійної моделі методом найменших квадратів;

– Створення нелінійної регресійної моделі методом зворотного перетворення;

– Оцінювання якості моделі за статистичними показниками (коефіцієнт детермінації R^2, MMRE, PRED(0.25)).

– Збереження моделі в локальній базі даних.

Сценарій 2: Оцінювання якості за існуючою моделлю

Введення даних:

– Введення метрик (RFC, CBO, WMC) для конкретного застосунку.

– Перевірка, чи відповідають значення введених метрик припустимим інтервалам моделі та перевірка квадрату відстані Махаланобіса, який не повинен перевищувати критичне значення хі-квадрат.

Прогнозування:

– Побудова довірчих інтервалів та інтервалів прогнозування.

– Використання існуючої моделі для розрахунку прогнозного значення якості застосунку.

Виведення результатів:

– Надання користувачу прогнозних значень якості.

– Вимоги до організації вхідних та вихідних даних

Програмне забезпечення повинно забезпечувати наступну організацію вхідних та вихідних даних:

1. Вхідні дані

Для побудови нової моделі та використання вже побудованої моделі:

– Набір даних у форматі CSV, що містить метрики RFC, CBO, WMC.

Для оцінювання якості:

– Значення метрик RFC, CBO, WMC для конкретного застосунку.

2. Вихідні дані

Для побудови нової моделі:

– Параметри нелінійної регресійної моделі;

– Показники якості моделей;

– Довірчі інтервали та інтервали прогнозування.

– Зберігання даних проектів (вибірки для побудови та тестування моделі) у форматі CSV, що містить метрики RFC, CBO, WMC.

Для оцінювання якості:

– Прогнозне значення якості застосунку.

– Результати перевірки припустимості введених метрик.

Визначені вимоги забезпечують необхідну функціональність для ефективного оцінювання якості Flutter-застосунків та повинні бути використані як основа для проектування архітектури системи та її подальшої реалізації.

3.2 Архітектура програмного забезпечення

Для розробки програмного забезпечення було обрано архітектуру MVC (Model-View-Controller) [36]. Вибір цієї архітектури для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter, обґрунтований її здатністю розділяти задачі та покращувати підтримку, масштабованість і повторне використання коду. MVC розділяє систему на три компоненти: модель (Model), представлення (View) та контролер (Controller) [16]. Ось як ці компоненти використовуються в архітектурі:

Модель (Model):

– Представляє дані та бізнес-логіку системи.

– Відповідає за отримання та маніпулювання даними з бази даних, пов'язаними з проектами, метриками, та параметрами якості побудованної регресії.

– Забезпечує доступ до даних і їх модифікацію, підтримуючи їх цілісність і узгодженість.

Представлення (View):

– Відповідає за логіку представлення системи.

– Забезпечує відображення даних користувачеві.

– Використовує різні форми візуалізації, наприклад графіки, таблиці або інші елементи інтерфейсу користувача.

– Дозволяє користувачеві взаємодіяти з системою і отримувати необхідну інформацію у зручному вигляді.

Контролер (Controller):

– Діє як посередник між моделлю та представленням.

– Обробляє ввід від користувача.

– Взаємодіє з моделлю для оновлення стану системи.

– Оновлює представлення в залежності від результатів взаємодії з моделлю.

– Забезпечує логіку взаємодії між моделлю і представленням.

Взаємодія між компонентами може бути представлена діаграмами компонентів і макетами UML, що ілюструють структурну організацію системи та взаємодію між компонентами. Використання архітектури MVC забезпечує наступні переваги:

– Кожен компонент має чітко визначену відповідальність, що сприяє модульності та полегшує обслуговування.

– Модель і контролер можна використовувати повторно на різних платформах, зменшуючи зусилля з розробки.

– Нові функції можуть бути додані шляхом розширення моделі, представлення або контролера без впливу на інші компоненти.

– Поділ завдань полегшує модульне тестування окремих компонентів, забезпечуючи їх коректність незалежно від інших.

На рисунку 3.1 представлено загальну діаграму компонентів програмного забезпечення для оцінювання якості застосунків, що створюються за допомогою фреймворку Flutter.

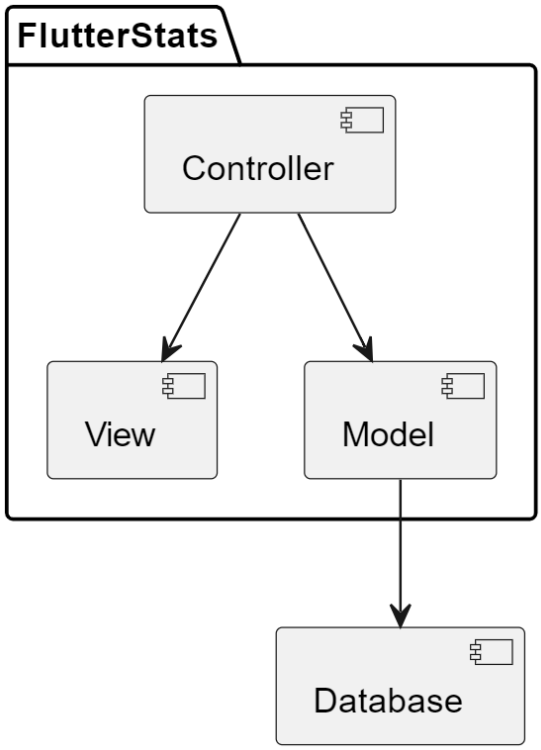


Рисунок 3.1 – Діаграма компонентів програмного забезпечення для оцінювання якості застосунків, що створюються за допомогою фреймворку Flutter

На цій схемі представлена MVC архітектура програмного забезпечення та її зв'язок із локальною базою даних. Контролер взаємодіє із представленням та моделлю. Модель зберігається у базі даних. На рисунку 3.2 представлено ланцюжок взаємодії компонентів програмного забезпечення для оцінювання якості застосунків, що створюються за допомогою фреймворку Flutter.

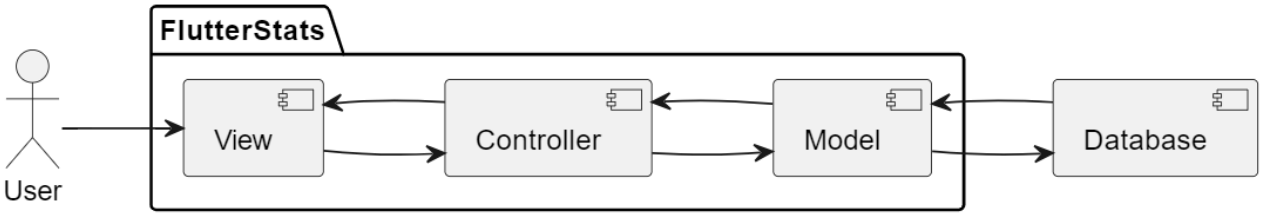


Рисунок 3.2 – Діаграма взаємодії компонентів програмного забезпечення для оцінювання якості застосунків, що створюються за допомогою фреймворку Flutter

Ця діаграма зображує архітектуру взаємодії компонентів, яка представляє наступне:

1. Користувач (User) взаємодіє з Представленням (View), яке є інтерфейсом користувача.

2. Представлення (View) спілкується з Контролером (Controller), передаючи введені користувачем дані та отримуючи оновлений стан для відображення.

3. Контролер (Controller) керує потоком даних між Представленням (View) та Моделлю (Model). Він отримує дані від Представлення, обробляє їх і передає у Модель, а також отримує оновлені дані від Моделі й передає їх у Представлення.

4. Модель (Model) містить логіку додатку та зберігає його стан. Вона взаємодіє з Базою Даних (Database) для зчитування та запису даних.

5. База Даних (Database) зберігає постійні дані.

3.3 Модель предметної області програмного забезпечення

Модель предметної області для задачі прогнозування якості програмних застосунків, реалізованих за допомогою фреймворку Flutter, відображає ключові елементи та їх взаємозв’язки, які необхідно враховувати під час розробки та оцінювання програмного забезпечення. Модель допомагає структурувати інформацію та визначити основні компоненти, що впливають на якість та складність проекту.

На рисунку 3.3 представлено модель предметної області аналізу якості застосунків реалізованих за допомогою фреймворку Flutter.

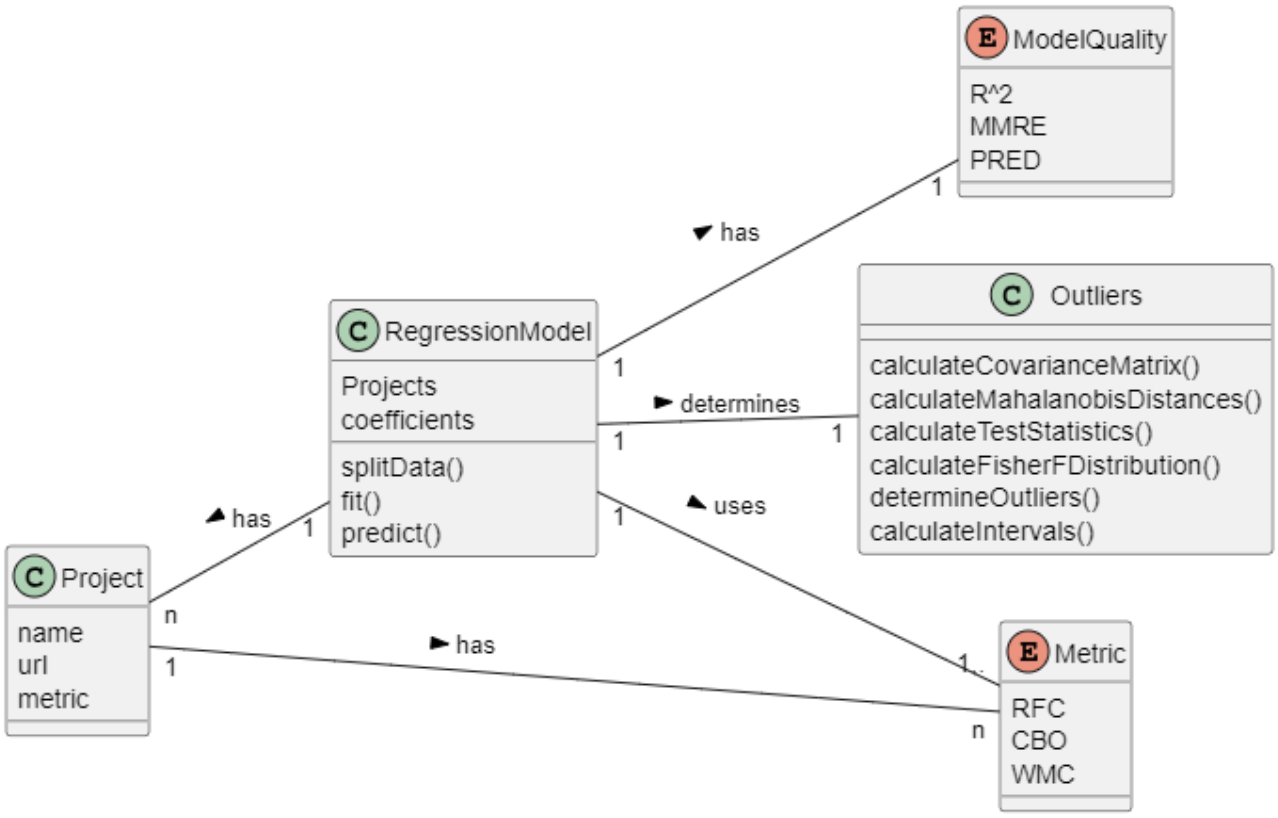


Рисунок 3.3 – Модель предметної області аналізу якості застосунків реалізованих за допомогою фреймворку Flutter

Дана модель предметної області описує систему для побудови регресійної моделі з метою прогнозування якості застосунків. Основні сутності моделі:

Project – представляє програмний проект, що аналізується. Має атрибути, такі як назва, посилання та метрики проекту.

Metric – перелік метрик програмного проекту такі як RFC, CBO, DIT, які виступають як незалежними так і залежними змінними при побудові регресії.

RegressionModel – регресійна модель, яка будується для прогнозування якості програмного забезпечення на основі метрик проекту. Має коефіцієнти регресії та метод для розділення вибірки та методи для побудови і використання моделі.

Outliers – представляє визначення викидів, включає в себе методи, такі як розрахунки коваріційної матриці, відстаней Махаланобіса, тестових статистик, F-розподілу Фішера та інтервалів прогнозування.

ModelQuality – критерій якості побудованої моделі (R^2, *MMRE*, *PRED*).

Зв'язки між сутностями відображають, що:

– Проект має (виміряні) метрики.

– Модель використовує ці метрики як вхідні дані.

– Модель прогнозує якість проекту за допомогою RFC.

3.4 Бізнес модель програмного забезпечення

Бізнес-модель, яка описує основні бізнес-процеси системи прогнозування якості застосунків, реалізованих за допомогою фреймворку Flutter. На рисунку 3.4 представлена бізнес-модель аналізу якості застосунків реалізованих за допомогою фреймворку Flutter.

Рисунок 3.4 - Бізнес-модель аналізу якості застосунків реалізованих за допомогою фреймворку Flutter

Дана діаграма UML відображає ключові бізнес-процеси, пов'язані з аналізом даних та створенням системи прогнозування якості застосунків, що базується на регресійній моделі.

В моделі виділено такі ключові бізнес-процеси:

Імпорт проектних даних – завантаження вихідних даних метрик Flutter проектів у форматі .csv для подальшого аналізу. Включає розділення даних на навчальну (60%) та тестову (40%) вибірки.

Перегляд ітерацій видалення викидів – пошук аномальних значень в даних, які можуть негативно вплинути на модель. Включає процеси видалення викидів за допомогою розрахунку коваріаційної матриці, квадратів відстані Махаланобіса, Тестової статистики, F-розподілу Фішера та інтервалів прогнозування.

Побудова регресійної моделі – створення моделі лінійної регресії та нелінійної на основі вибірки для побудови моделі (60%). Включає розрахунок коефіцієнтів регресії.

Визначення якості моделі – оцінка отриманої регресійної моделі за допомогою критеріїв: R^2, *MMRE, PRED(0.25)*.

Здійснення прогнозів – отримання прогнозних значень якості проектів на основі побудованої регресійної моделі за допомогою метрик RFC, CBO, WMC.

Отже, діаграма детально описує основні процеси аналізу даних, побудови прогнозної моделі для задачі прогнозування якості застосунків.

3.5 Побудова діаграми діяльності програмного забезпечення

Під час розробки програмного забезпечення «FlutterStats» для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter було створено декілька артефактів для формалізації поведінки та дизайну системи. Це включало побудову діаграми діяльності для представлення сценаріїв основних варіантів використання.

На рисунку 3.5 представлено діаграму діяльності «FlutterStats» для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter.

Рисунок 3.5 – Діаграма діяльності «FlutterStats» для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter

Процес починається з імпорту проектних даних, необхідних для аналізу. Наступним кроком є видалення викидів. Здійснюється розрахунок квадратів відстаней Махаланобіса (2.1) для кожного спостереження для нормалізованих даних. На основі квадратів відстаней Махаланобіса обчислюється тестова

статистика (2.6), а потім визначається критичне значення за допомогою F-розподілу Фішера (2.7). Якщо тестова статистика перевищує критичне значення, викиди видаляються, і процес повторюється до тих пір, поки не залишиться викидів. Далі будується рівняння нелінійної регресії (2.9) та перевіряється, якщо є точки даних які виходять за інтервал прогнозування (2.8), то потрібно вилучити їх із вибірки. Повторюється цей крок, поки не буде вилучено усі викиди.

Після видалення викидів дані розділяються на навчальну (60%) та тестову (40%) вибірки. Використовуючи метод найменших квадратів (2.10), розраховується коефіцієнті регресії та будується рівняння лінійної регресії. Далі, за допомогою зворотного перетворення отримаємо рівняння нелінійної регресії (2.11). Потім проводиться оцінка якості побудованої моделі.

Завершальним етапом є здійснення прогнозів якості на основі побудованої регресійної моделі. Розраховуються довірчі інтервали (2.12) та інтервали прогнозування. Розраховується якість застосунку за метриками RFC, CBO та WMC. На цьому процес завершується.

На рисунку 3.6 представлено діаграму діяльності «FlutterStats» для розділення вибірки на дві для побудови та тестування моделі.

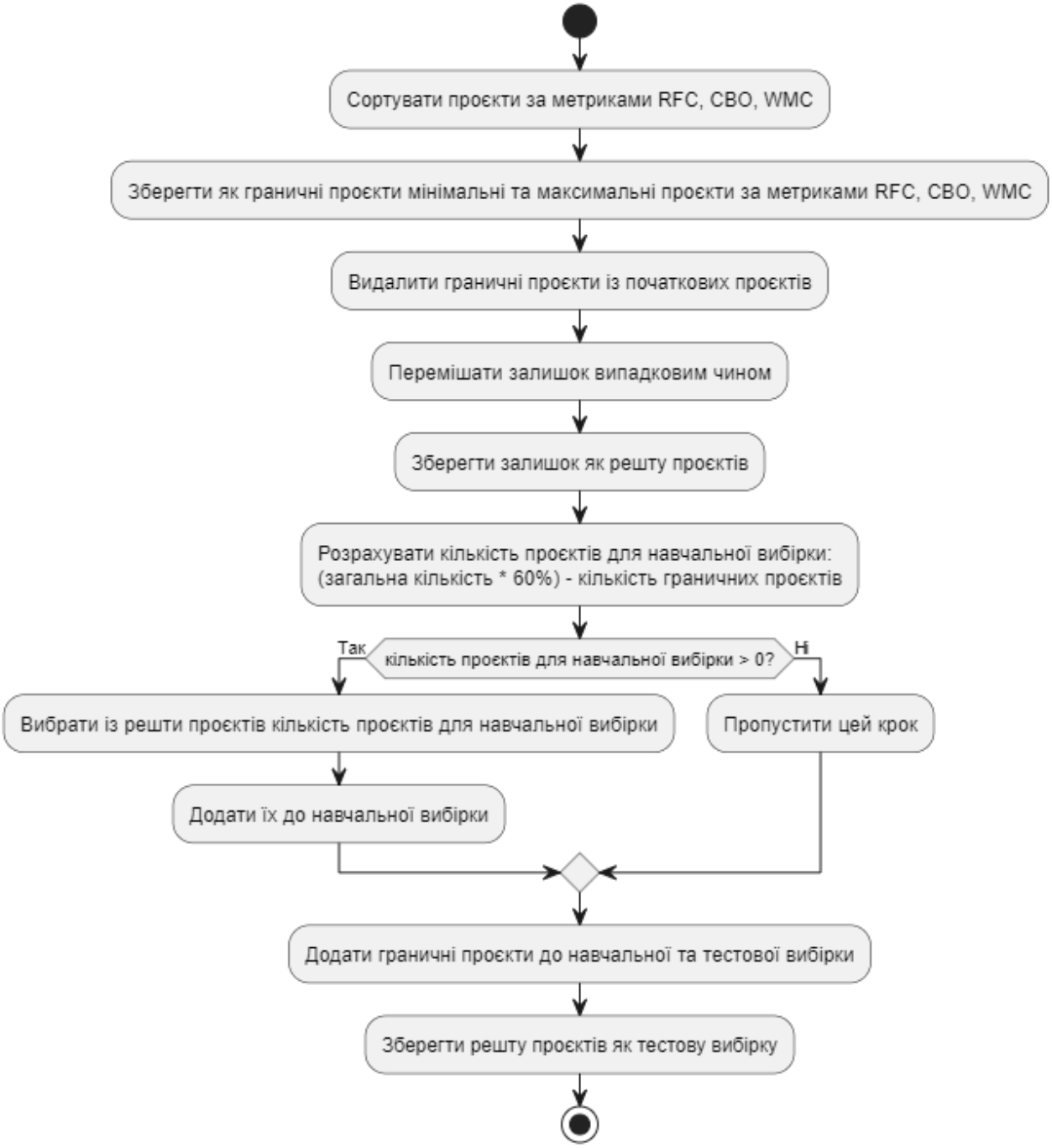


Рисунок 3.6 – Діаграма діяльності «FlutterStats» для розділення вибірки на дві для побудови та тестування моделі

Алгоритм розділяє список проектів на навчальну та тестову вибірки на основі заданого співвідношення (60%). Вона також гарантує, що в навчальній та для тестування вибірці завжди будуть представлені крайні значення (мінімальні

та максимальні) для кожного з визначених метрик. Спочатку створюється локальна копія початкового списку проектів, щоб уникнути змін у початкових даних. Список проектів сортується окремо за кожною з метрик: RFC, CBO, WMC. З кожного відсортованого списку вибираються проекти з найменшими та найбільшими значеннями метрик. Ці проекти вважаються важливими, оскільки представляють крайні випадки даних. Створюється множина із мінімальних та максимальних проектів для кожної метрики. Завдяки множині гарантується унікальність об'єктів. З основного списку проектів видаляються всі проекти, що входять до множини мінімальних/максимальних проектів. Решта проектів перемішуються випадковим чином для забезпечення випадкового розподілу. Визначається кількість проектів, необхідних для навчальної вибірки (60%), враховуючи співвідношення та кількість вже обраних мінімальних/максимальних проектів (граничні проекти). Навчальна вибірка складається з граничних проектів та частини випадково вибраних проектів із залишку. Решта проектів формують тестову вибірку.

3.6 Ескізний проект програмного забезпечення

На основі аналізу вимог до програмного забезпечення було розроблено ескізний проект програмного забезпечення. Прототипи вікон програми приведено на рисунках 3.6 – 3.8.

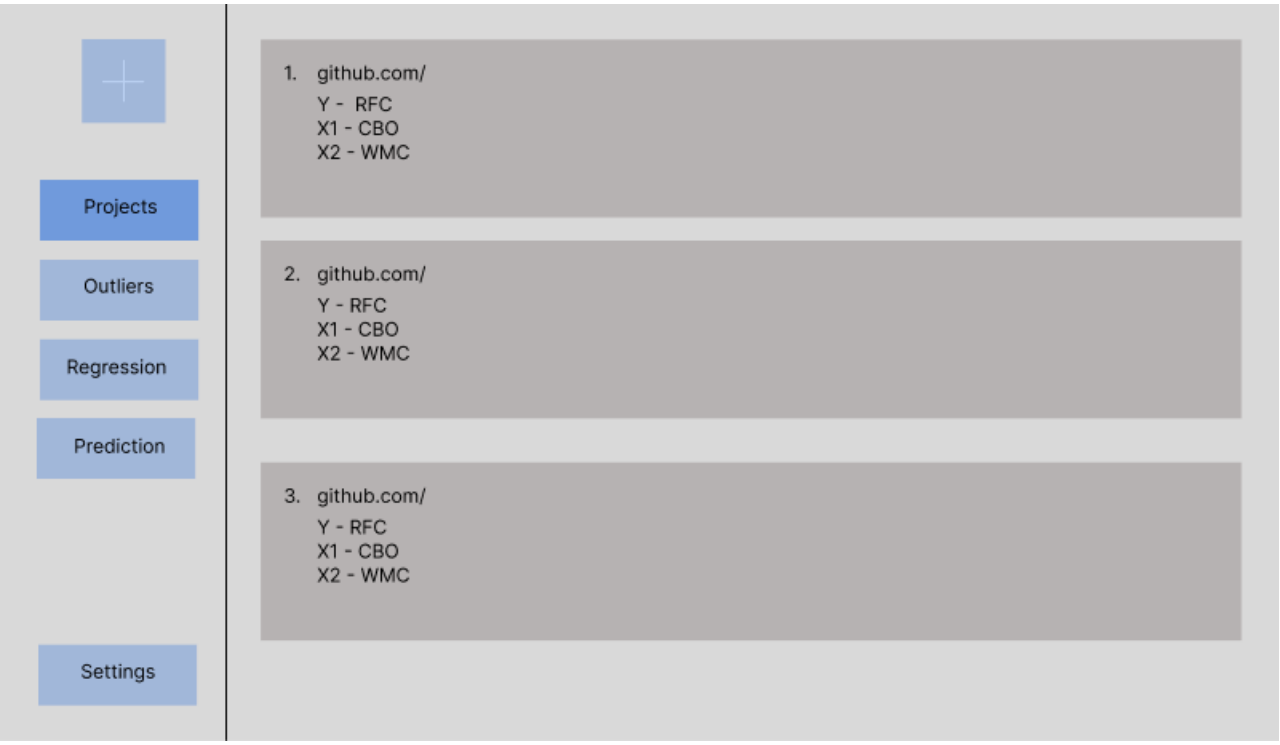


Рисунок 3.6 – Прототип головного екрана «FlutterStats» для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter

Головне вікно має бічну панель зліва з опцією «Додати» та пунктами «Проекти», «Викиди», «Регресія», «Передбачення» та «Налаштування». Основна область праворуч містить завантажені програмні проекти. Кожен проекти містить нумерацію, URL та метрики:

– Y – RFC (Response for Class).
– X1 – CBO (Coupling Between Object Classes).
– X2 – WMC (Weighted Methods per Class).

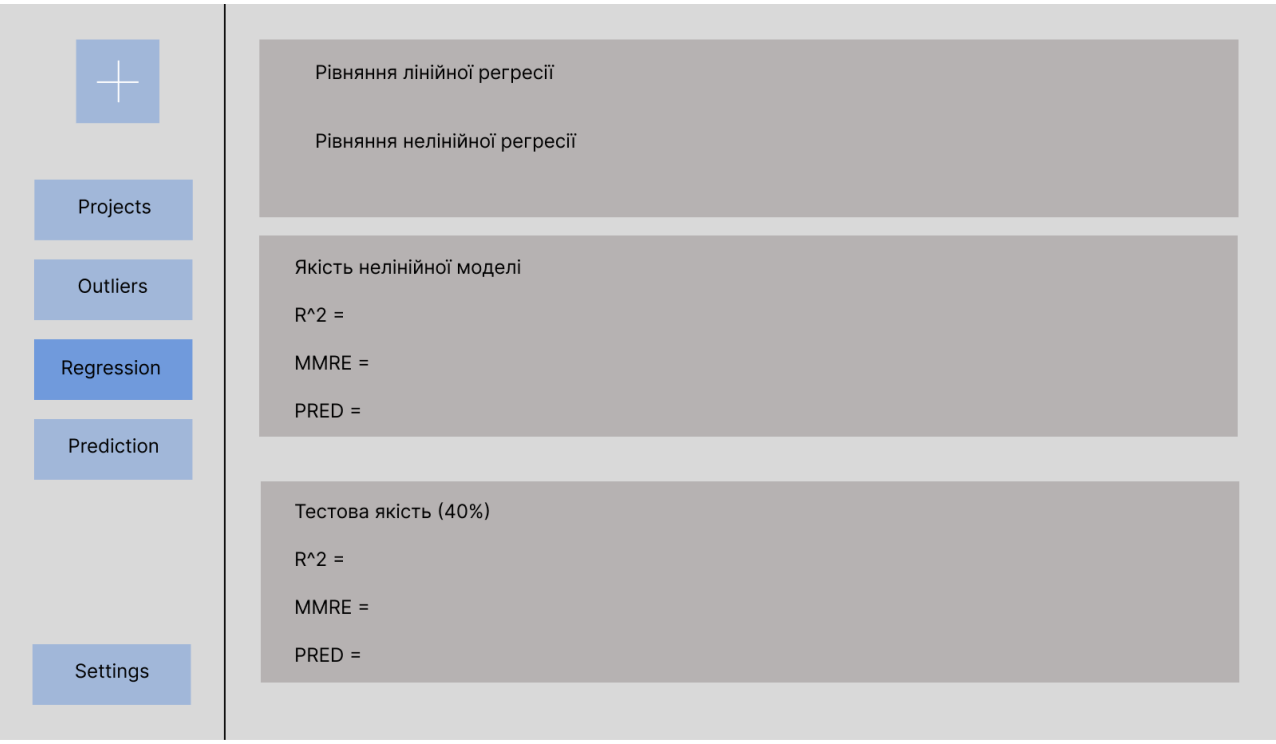


Рисунок 3.7 – Прототип екрана «Regression» «FlutterStats» для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter

Вкладка «Regression» відображає рівняння лінійної (2.9) та нелінійної регресійної моделі (2.11). Внизу наведено метрики якості нелінійної моделі та тестової: коефіцієнт детермінації R^2, середня відносна похибка *MMRE* та відсоток передбачень у допустимому діапазоні *PRED(0.25)*.

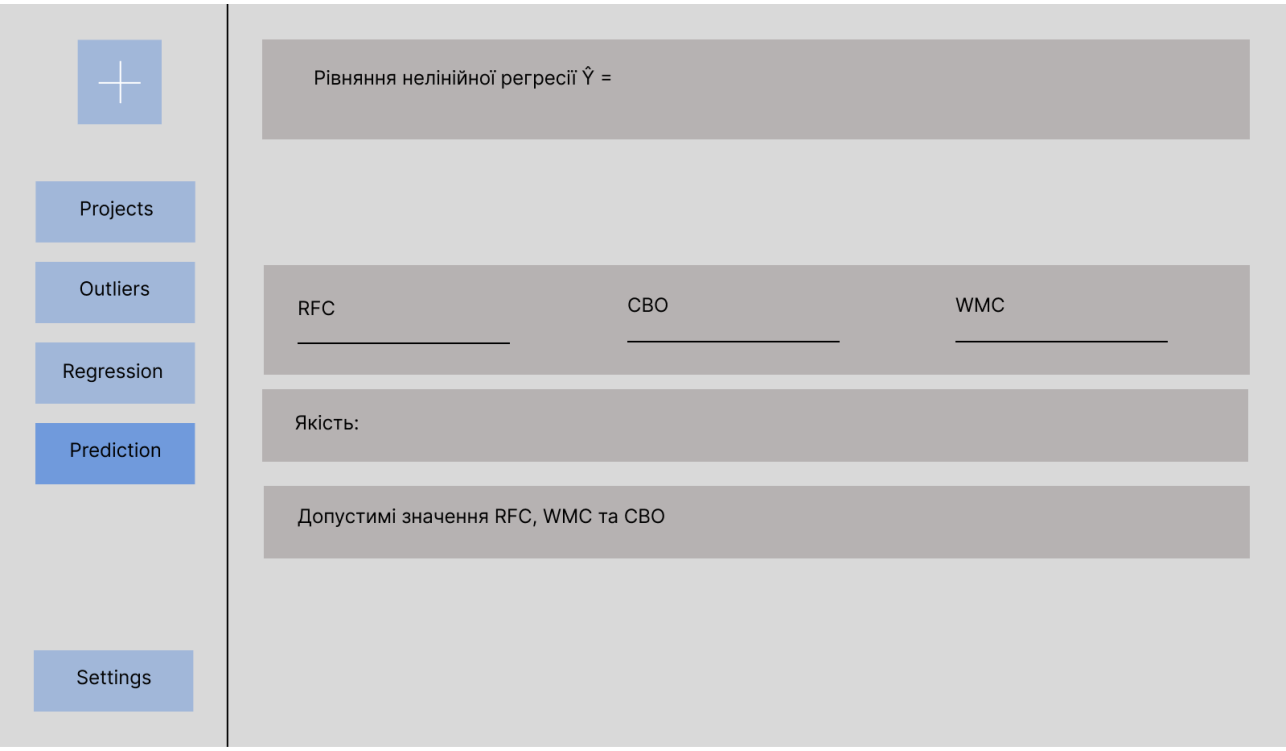


Рисунок 3.8 – Прототип екрана «Prediction» «FlutterStats» для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter

Вкладка «Prediction» відображає нелінійної регресійної моделі. Ця вкладка дозволяє виконувати регресійний аналіз, для оцінки якості застосунків за допомогою метрик CBO та WMC. Внизу наведено допустимі діапазони метрик WMC та CBO.

3.7 Технічний проект програмного забезпечення

Після ретельного аналізу початкового проекту було успішно створено комплексний технічний проект програмного забезпечення «FlutterStats», для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter. На рисунку 3.9 зображено статичну модель «FlutterStats» у вигляді діаграми класів.

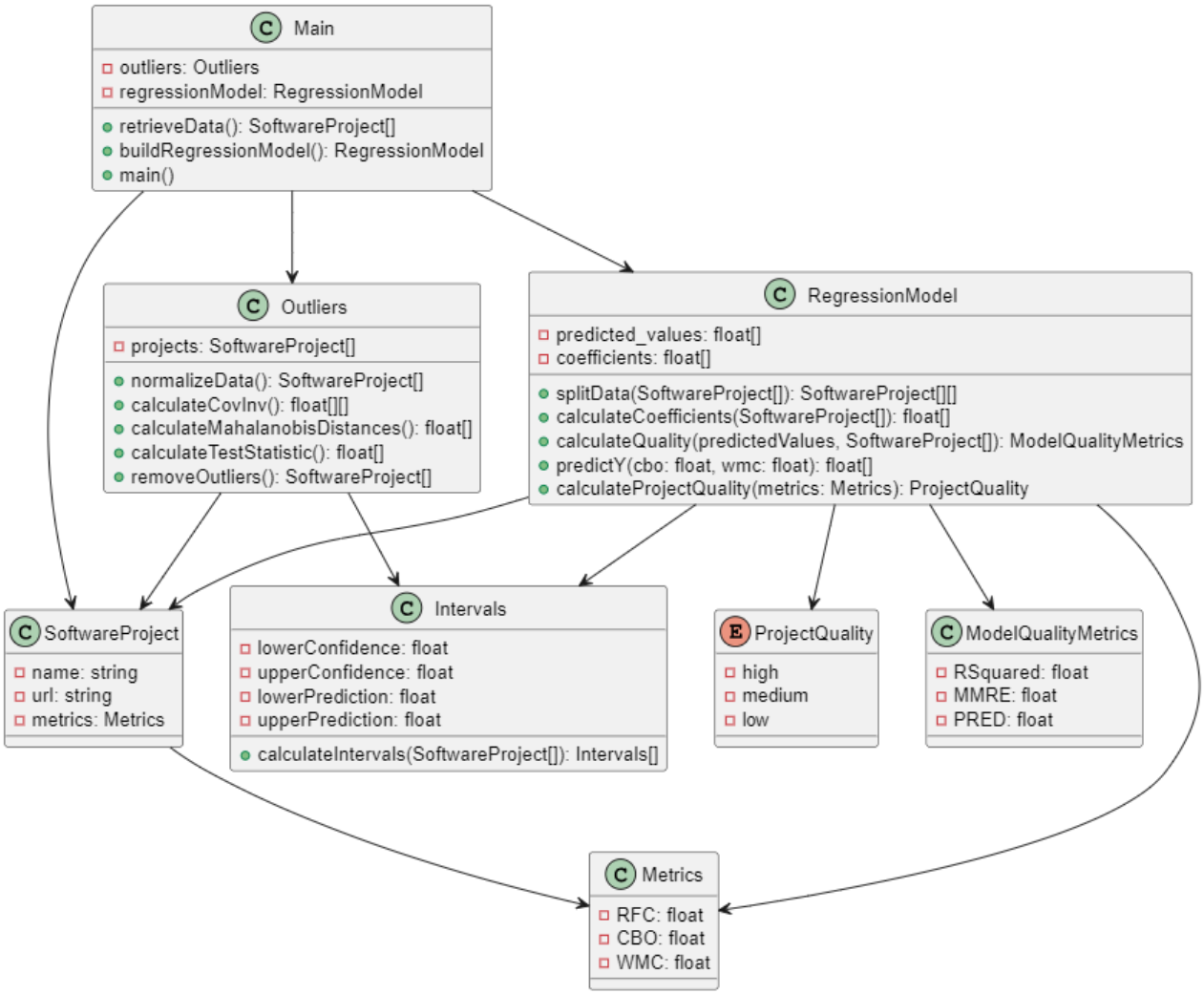


Рисунок 3.9 – Діаграма класів «FlutterStats» для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter

Ця діаграма класів UML представляє систему, яка аналізує дані проектів програмного забезпечення та будує регресійну модель для передбачення певної величини. Класи**:**

– Main: Цей клас є точкою входу в програму. Він відповідає за завантаження даних про проекти програмного забезпечення, видалення викидів, побудову регресійної моделі та виконання основної логіки програми.

– SoftwareProject: Цей клас представляє одиничний проект програмного забезпечення. Він містить інформацію про назву проекту, його URL-адресу та об'єкт класу Metrics, що зберігає метрики проекту.

– Metrics: Цей клас зберігає метрики проекту, такі як RFC, CBO, WMC.

– Outliers: Цей клас відповідає за обробку даних проектів. Він нормалізує дані, обчислювати коваріаційну матрицю та її обернену, розраховувує відстані Махаланобіса, тестову статистику, F-розподіл Фішера, інтервали прогнозування та видаляє викиди.

– RegressionModel: Цей клас представляє регресійну модель. Він розділяє проекти на вибірку для побудови та для тестування моделі, обчислює коефіцієнти регресії на основі вибірки, оцінює якість моделі та робить прогнози якості застосунків.

– ModelQualityMetrics: Цей клас зберігає показники якості моделі, такі як коефіцієнт детермінації (R^2), середню абсолютну похибку відновлення (MMRE) та прогнозоване значення (PRED).

– Intervals: Цей клас зберігає та розраховує довірчі інтервали та інтервали прогнозування.

– ProjectQuality: Цей клас представляє якість застосунку: високу, середню та низку.

Взаємозв'язки**:**

– Залежність: Клас Main залежить від класів Outliers та RegressionModel. Це означає, що Main використовує ці класи для виконання своєї роботи.

– Агрегація: Клас SoftwareProject містить об'єкт класу Metrics. Це означає, що SoftwareProject має зв'язок «має» з Metrics.

– Асоціація: Інші зв'язки на діаграмі є асоціаціями. Вони показують, як класи пов'язані між собою, але не обов'язково означають, що один клас містить інший.

3.8 Реалізація програмного забезпечення

При розробці програмного забезпечення «FlutterStats» для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter, будуть використані наступні технології та інструменти.

Visual Studio Code [37] буде використовуватися як середовище розробки. Це легкий та потужний редактор коду, розроблений компанією Microsoft. Він підтримує різні мови програмування та забезпечує широкі можливості для редагування, налагодження і розробки програмного коду.

Буде використовуватися Material Design як система дизайну, яка використовується для створення веб–сайтів, мобільних додатків та інших цифрових продуктів. Вона була розроблена Google. Material Design використовується для створення продуктів, які мають чіткий, сучасний та інтуїтивно зрозумілий дизайн.

Програмне забезпечення «FlutterStats» буде розроблено з використанням фреймворку Flutter. Flutter – це відкритий кросплатформений фреймворк для розробки мобільних, веб та настільних додатків з єдиною базою коду. Flutter використовує Material Design для створення красивих та інтуїтивно зрозумілих інтерфейсів. Буде встановлено Flutter SDK, а команди Flutter будуть використані для створення, побудови та запуску проекту Flutter.

Мова програмування Dart буде використана в поєднанні з Flutter для розробки «FlutterStats». Файли Dart (.dart) будуть написані для визначення компонентів інтерфейсу користувача, обробки взаємодії з користувачем.

Використання Flutter дозволяє значно прискорити розробку та спростити супровід програмних додатків для різних платформ – iOS, Android, Windows, Linux, MacOS та Web.

Офіційне сховище пакетів для Dart і Flutter додатків – це pub. Pub є менеджером пакетів для мови програмування Dart і використовується для виявлення, встановлення та управління пакетами Dart.

Завдяки кросплатформеності, «FlutterStats», розроблений на Flutter, можна буде запускати як на мобільних пристроях, так і на настільних комп'ютерах чи в браузері з мінімальними змінами коду. Використання єдиного кодової бази на Dart значно пришвидшить розробку та спростить подальше масштабування і підтримку програмного застосунку. Отже, вибір кроссплатформеного фреймворку Flutter є стратегічно вірним рішенням для даного програмного проекту.

Для ініціалізації проекту Flutter та налаштування необхідних залежностей будуть виконані наступні кроки:

– Flutter SDK повинен бути завантажений та встановлений для розробки.

– Необхідно створити новий проект Flutter за допомогою команди «flutter create». Це створить базову структуру проекту та файли.

– До файлу «pubspec.yaml» проекту повинні бути додані необхідні залежності. Сюди увійдуть наступні пакети із pub.dev: file_picker, csv для завантаження даних та provider для керуванням станом застосунка.

– Потрібно написати код використовуючи мову програмування Dart. Це буде включати створення компонентів інтерфейсу користувача та реалізацію функціональності застосунка, такої як завантаження даних із csv файлу, видалення викидів та побудову моделі.

– Після завершення розробки «FlutterStats» наступним кроком буде його збірка на платформу Windows. У проекті Flutter будуть створені оптимізовані ресурси у каталозі збірки проекту за допомогою команди flutter build windows.

Загалом, використання цих технологій та інструментів дозволяє створити функціональне та зручне програмне забезпечення «FlutterStats», яке надає

користувачеві завантажувати набор даних, які містять дані про RFC, CBO, WMC та на основі цих даних будувати регресійну модель, яка дозволяє прогнозувати якість.

У результаті розробки було створено функціональне програмне забезпечення «FlutterStats». На рисунку 3.10 представлено головну сторінку застосунку «FlutterStats».

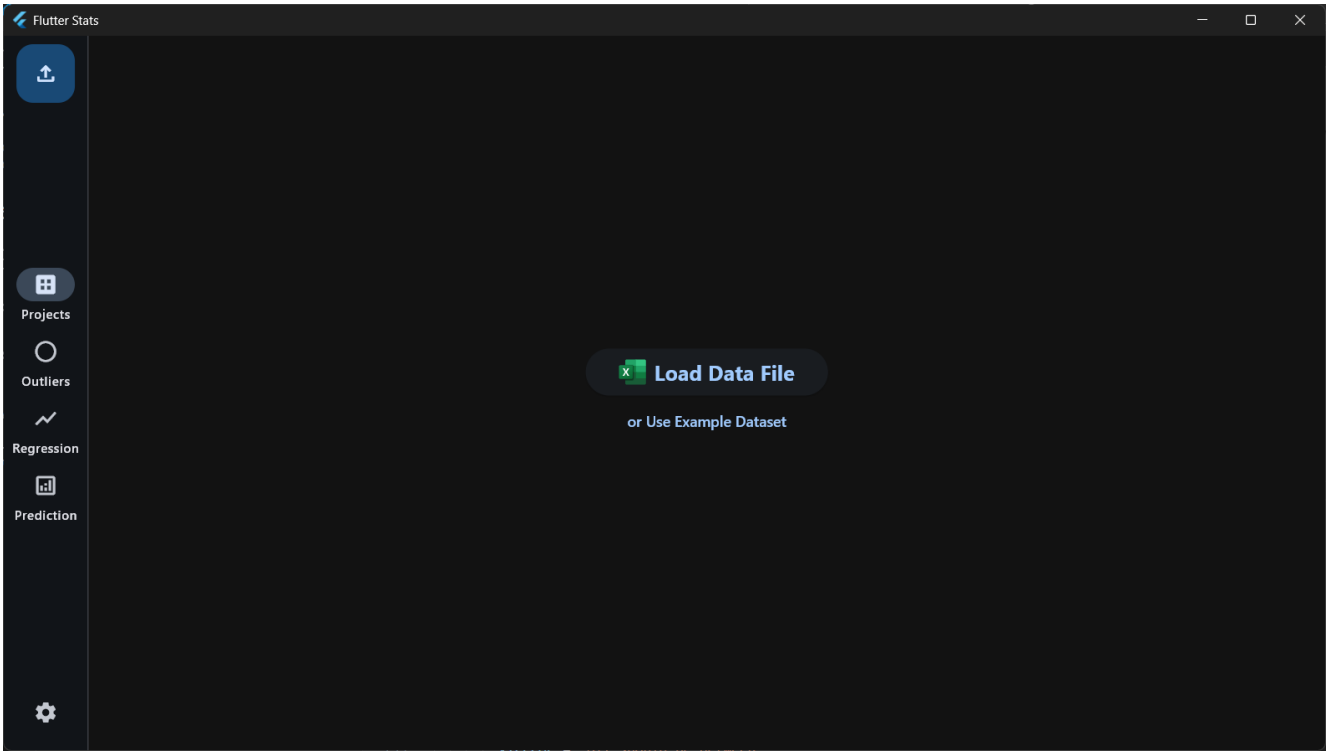


Рисунок 3.10 – Головна сторінка застосунку «FlutterStats»

На головній сторінці можна побачити кнопку із завантаженням даних із .csv файлу або використання прикладу набору даних із 100 проектів та бокові компоненти меню, такі як:

– Projects – перегляд завантажених проектів.

– Outliers – перегляд ітерацій виявлених викидів, які були вилучені з набору даних.

– Regression – перегляд побудованих лінійної та нелінійної регресійних моделей, її коефіцієнтів та показників якості.

– Prediction – прогнозування якості проектів за допомогою метрик CBO та WMC.

Після того як данні були завантаженні, їх обробляє застосунок, автоматично видаляє викиди та відображає у вкладці Projects. На рисунку 3.11 представлено вкладку Projects із завантаженими даними та перегляд розділення проектів на вибірки для побудови (60%) та тестування моделі (40%). Користувач може завантажити їх у форматі .csv.

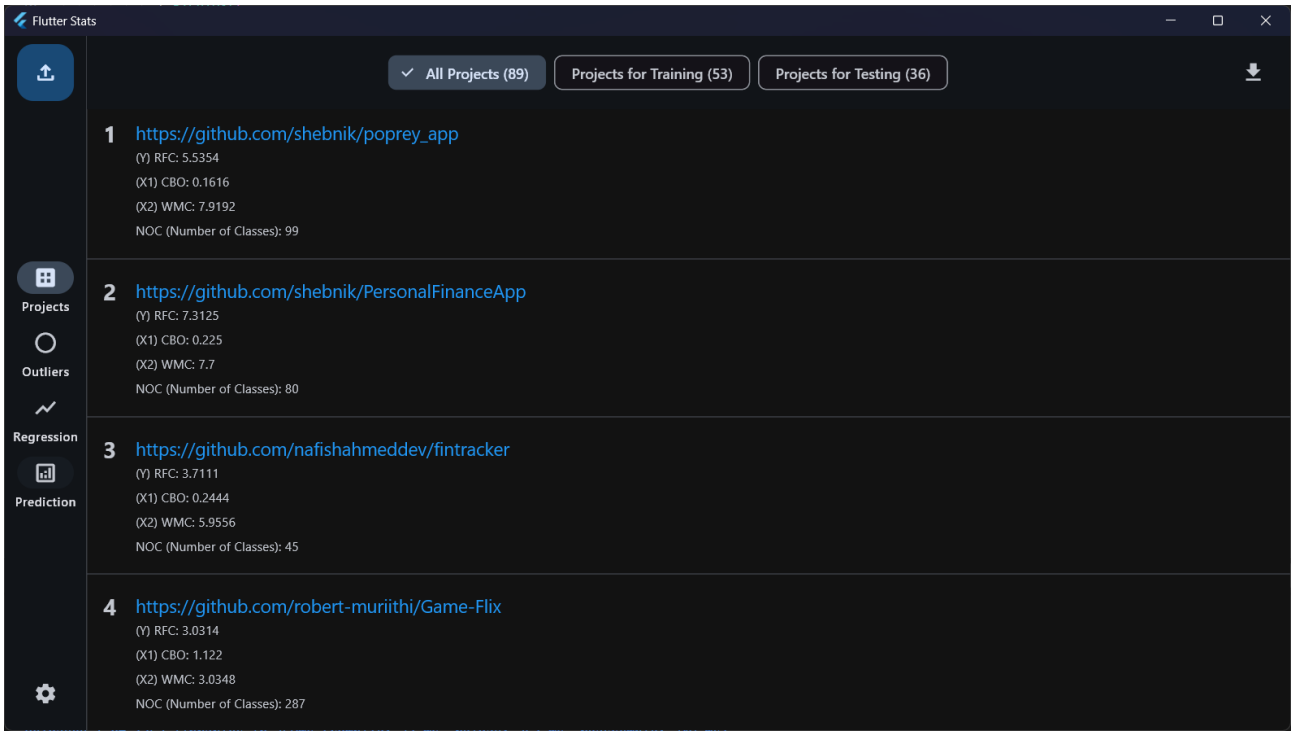


Рисунок 3.11 – Вкладка Projects застосунку «FlutterStats»

Користувач може переглядати набір завантажених даних без викидів та вибірки для побудови та тестування моделі. На рисунку 3.12 представлено вкладку із ітераціями видалення викидів.

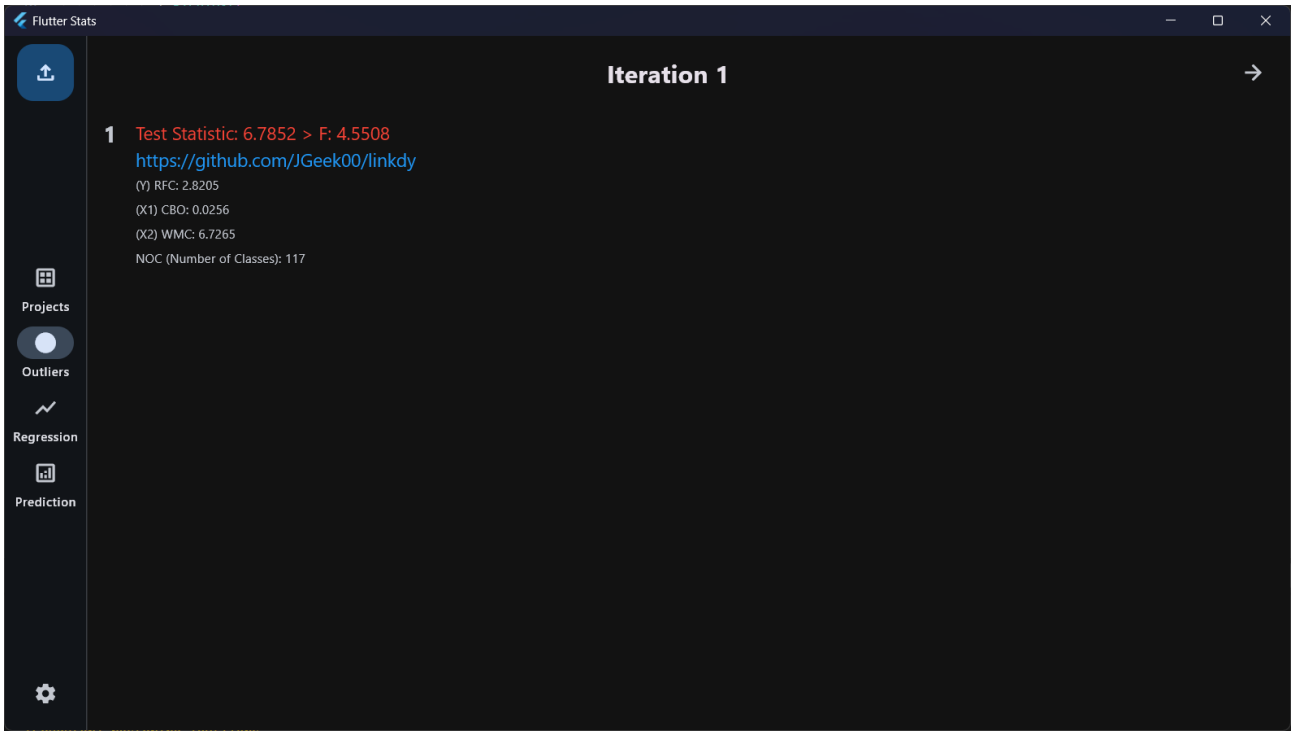


Рисунок 3.12 – Вкладка Outliers застосунку «FlutterStats»

Користувач може переглядати кожну ітерацію видалення викидів. На рисунку 3.13 представлена вкладка Regression (побудована лінійна та нелінійна регресійна модель із завантажених даних та її якість, без викидів).

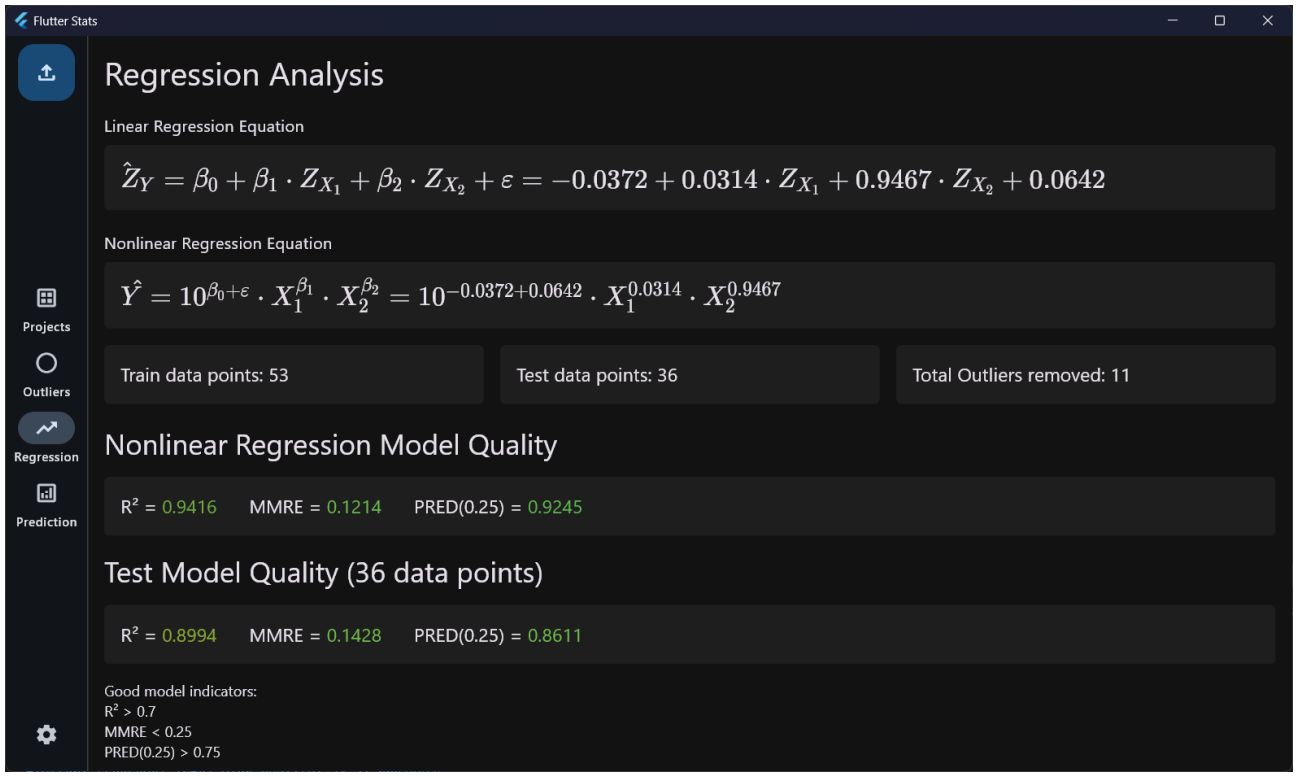


Рисунок 3.13 – Вкладка Regression застосунку «FlutterStats»

Можна побачити, що усього було вилучено 11 викидів та вибірка була розділена на дві для побудови моделі (53 застосунків) та для її тестування (36 застосунків). Коефіцієнти $\beta_0, \beta_1, \beta_2$ моделі склали відповідно -0,0372, 0,0314 та 0,9467. ε – гаусівська випадкова величина з нульовим математичним сподіванням та середньоквадратичним відхиленням, оцінка якого дорівнює 0,0642. Нелінійна модель має критерій якості R^2 = 0,9416, *MMRE* = 0,1214 та *PRED(0,25)* = 0,9245. Тестування моделі показало наступні критерії якості: R^2 = 0,8994, *MMRE* = 0,1428 та *PRED(0,25)* = 0,8611. На рисунку 3.14 представлена вкладка Prediction для прогнозування якості застосунків.

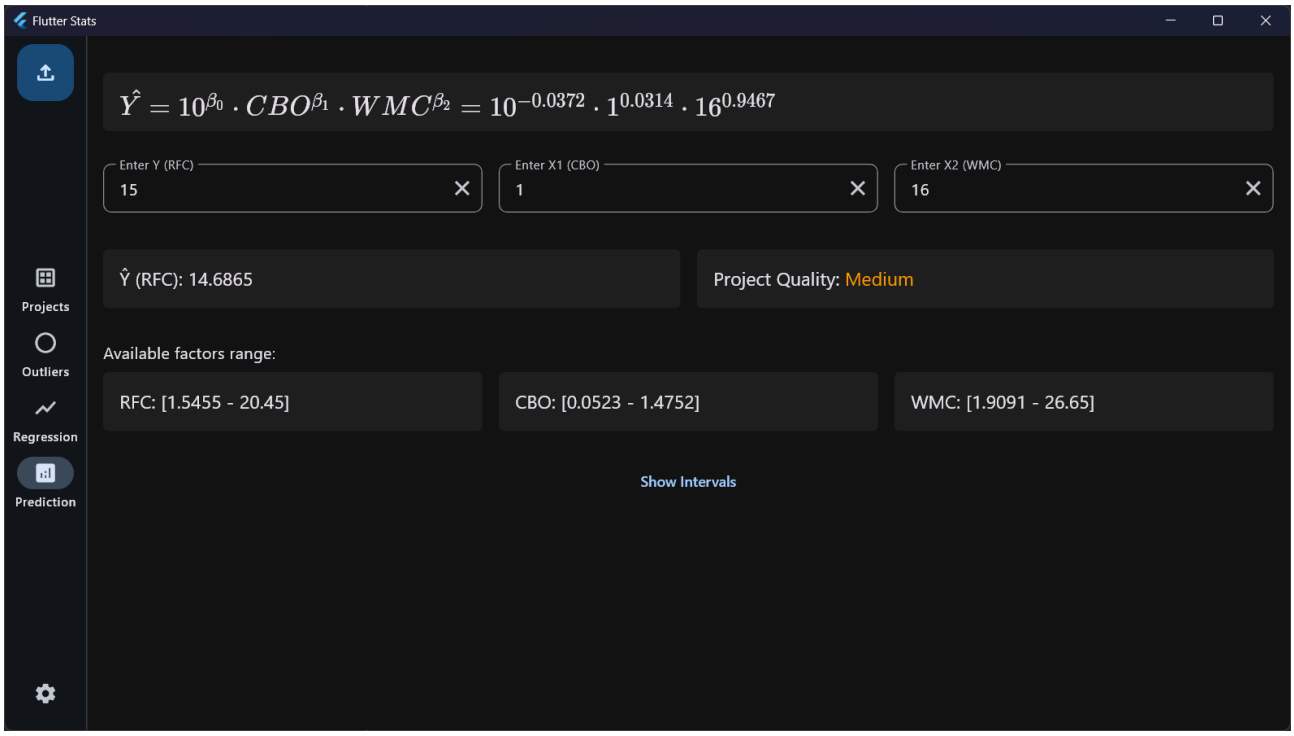
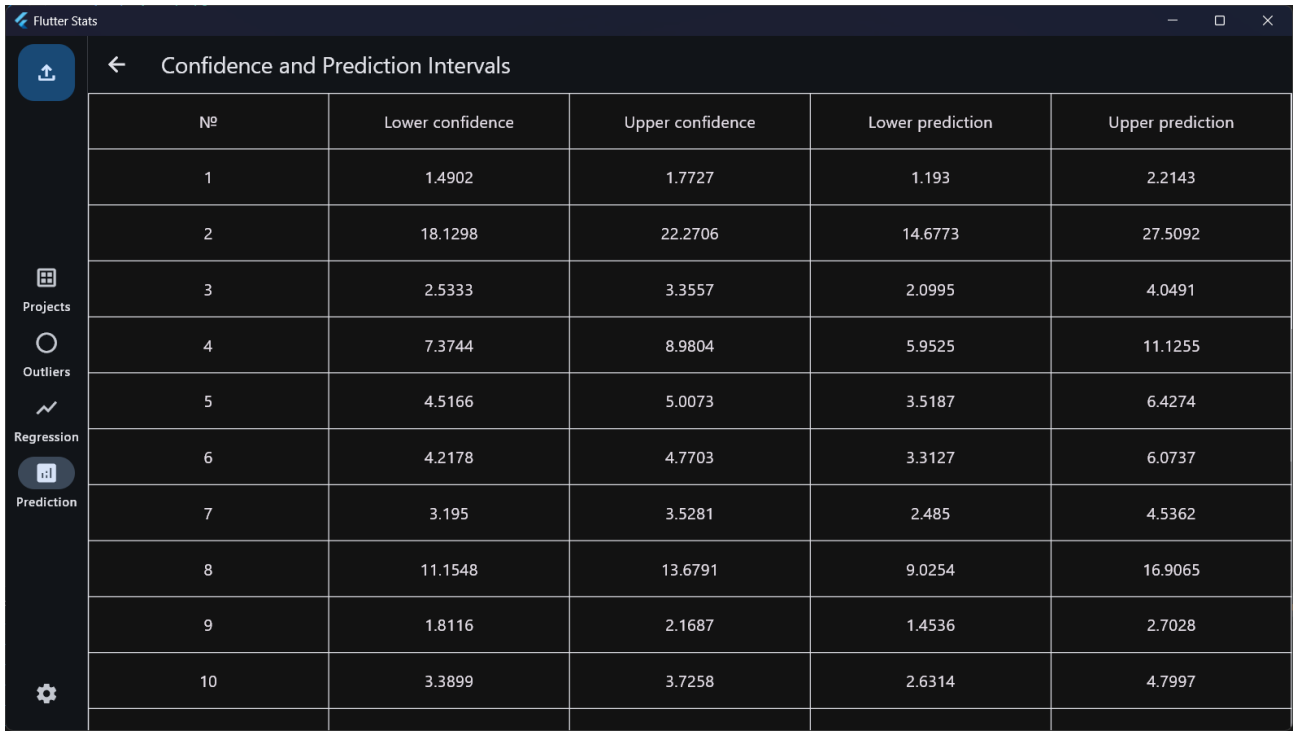


Рисунок 3.14 – Вкладка Prediction застосунку «FlutterStats»

Користувач може ввести RFC, CBO та WMC та згідно рівняння нелінійної регресії (2.11), застосунок обчислить передбачений RFC та якість. Можна побачити що для RFC: 15, CBO: 1 та WMC: 16, RFC передбачуване дорівнює 14,6865 та передбачена якість середня. Нижче представлені допустимі діапазони значень для RFC [1,5455 – 20,45], CBO [0,0523 – 1,4752] та WMC [1,9091 – 26,65]. Якщо якість неможливо визначити, застосунок повідомить про це. Нижче розташована кнопка для перегляду інтервалів. На рисунку 3.15 представлено розраховані інтервали застосунку «FlutterStats».



№	Lower confidence	Upper confidence	Lower prediction	Upper prediction
1	1.4902	1.7727	1.193	2.2143
2	18.1298	22.2706	14.6773	27.5092
3	2.5333	3.3557	2.0995	4.0491
4	7.3744	8.9804	5.9525	11.1255
5	4.5166	5.0073	3.5187	6.4274
6	4.2178	4.7703	3.3127	6.0737
7	3.195	3.5281	2.485	4.5362
8	11.1548	13.6791	9.0254	16.9065
9	1.8116	2.1687	1.4536	2.7028
10	3.3899	3.7258	2.6314	4.7997

Рисунок 3.15 – Вкладка Prediction застосунку «FlutterStats»

Користувач може переглядати інтервали завдяки яким прогнозувалась якість застосунку. На рисунку 3.16 представлено налаштування застосунку «FlutterStats».

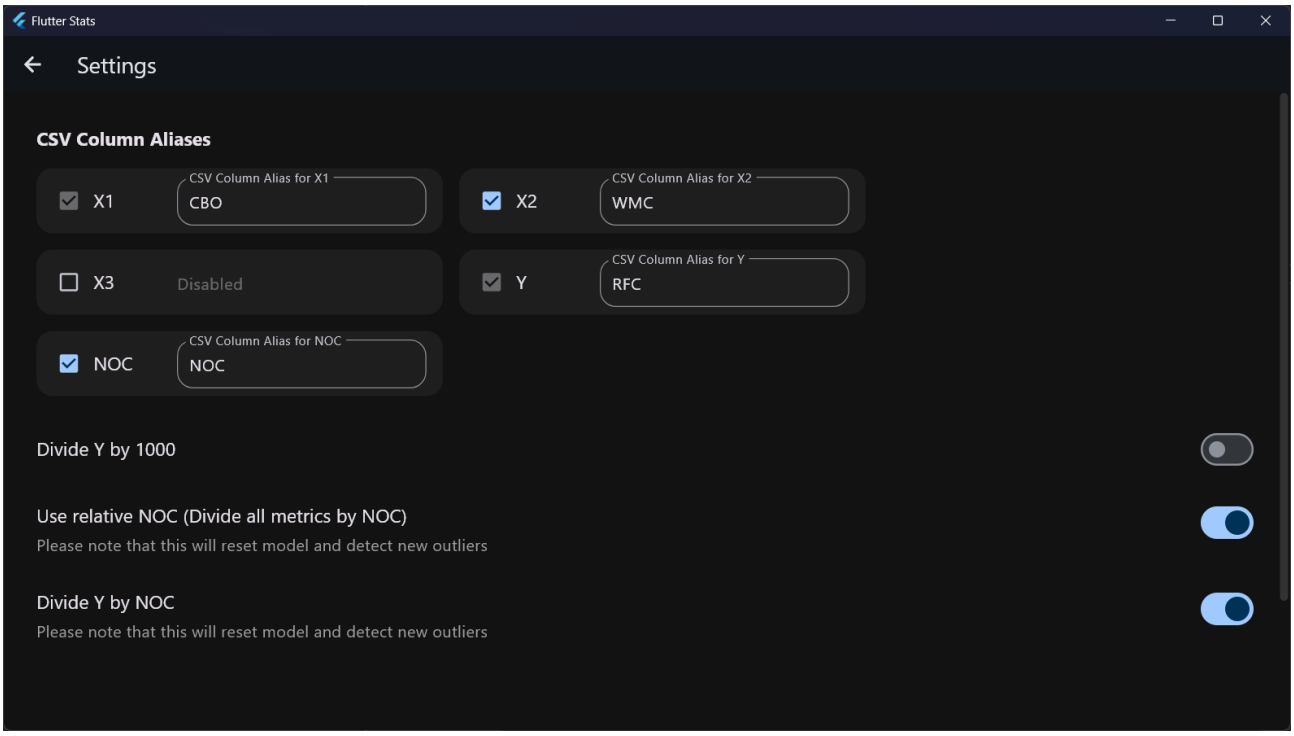


Рисунок 3.16 – Налаштування застосунку «FlutterStats»

Користувач може налаштувати рядки та метрики із .csv файлу для завантаження, поділ Y на 1000 або на NOC за бажанням та поділ усіх метрик на NOC для використання відносних значень в залежності від кількості класів.

3.9 Випробування програмного забезпечення

Під час проведення випробувань програмного забезпечення «FlutterStats» для аналізу якості застосунків реалізованих за допомогою фреймворку Flutter були проведені наступні тести для перевірки вимог до функціональних характеристик:

– Завантаження даних з CSV файлу;

– Виявлення та видалення викидів;

– Розділення проектів на вибірки для побудови (60%) та тестування моделі (40%);

– Побудова регресійної моделі;

– Здійснення прогнозів;

– Налаштування застосунку.

У таблиці 3.1 представлено випробування програмного забезпечення.

Таблиця 3.1 – Випробування програмного забезпечення

№	Дія	Очікуваний результат	Результат
1	Завантаження даних з CSV файлу	Дані успішно завантажені та відображаються у вкладці «Projects».	Успішно
2	Виявлення та видалення викидів	Викиди успішно виявляються та видаляються та відображаються у вкладці «Outliers».	Успішно
3	Розділення проектів на 2 вибірки	Дані успішно розділяються та відображаються у вкладці «Projects».	Успішно
4	Побудова регресійної моделі	Регресійна модель успішно побудована та відображається у вкладці «Regression».	Успішно
5	Здійснення прогнозів	Прогнози успішно здійснюються та відображаються у вкладці «Prediction».	Успішно
6	Налаштування застосунку	Усі налаштування застосовуються належним чином.	Успішно

Під час випробувань програмного забезпечення «FlutterStats» всі функціональні вимоги були успішно протестовані. Дані завантажувалися з CSV файлу без помилок, викиди виявлялися та видалялися, модель будувалася належним чином, прогнози здійснювалися успішно та усі налаштування застосовуються належним чином.

Таким чином, програмне забезпечення «FlutterStats» успішно пройшло всі тести на функціональність і може бути використане для оцінювання якості програмних застосунків, реалізованих з використанням фреймворку Flutter.

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВІД РОЗРОБКИ І ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ ЯКОСТІ ЗАСТОСУНКІВ, ЩО СТВОРЮЮТЬСЯ НА ПЛАТФОРМІ FLUTTER

4.1 Аналіз актуальності та ринкових перспектив

Програмне забезпечення, створене для оцінки якості застосунків на платформі Flutter, відповідає актуальним вимогам ринку IT-технологій. Зростаюча популярність кросплатформної розробки обумовлює потребу у спеціалізованих інструментах, які підвищують ефективність та якість розробки програмних продуктів.

Запропонована розробка орієнтована на програмістів, які використовують платформу Flutter, та підприємства, що займаються розробкою мобільних застосунків. Очікується, що програмний продукт матиме попит через його унікальні можливості автоматизованого аналізу та оцінювання коду.

4.2 Розрахунок витрат на розробку

Для розрахунку витрат враховані основні статті, які включають:

- Основну і додаткову заробітну плату.
- Витрати на допоміжні матеріали.
- Витрати на електроенергію та загальногосподарські потреби.

У таблиці 4.1 представлено витрати на допоміжні матеріали.

Таблиця 4.1 – Витрати на допоміжні матеріали

Пункти витрат	Сума, грн
Папір (5 пачок)	500
Заправлення картриджа до принтера	800
CD-диски (5 шт.)	200
Література	1000
Непередбачені витрати	500
Разом	3000

У таблиці 4.2 представлено витрати на розробку програмного забезпечення.

Таблиця 4.2 – Витрати на розробку програмного забезпечення

Пункти витрат	Сума, грн
Основна заробітна плата розробника	60 000
Додаткова заробітна плата (премії тощо)	5 000
Відрахування на соціальні заходи (22%)	14 300
Загальногосподарські витрати	6 000
Витрати на допоміжні матеріали	3 000
Витрати на електроенергію	2 500
Разом	90 800

4.3 Прогнозовані доходи та оцінка економічної ефективності

Програмне забезпечення буде реалізовано за ліцензійною моделлю з ціною однієї ліцензії 300 грн. Прогнозований обсяг продажів на перший рік становить 300 ліцензій. Очікуваний дохід: 300 × 300 = 90 000 грн.

Чиста приведена вартість (NPV) [38]:

$$NPV = \sum_{t=1}^{T} \frac{R_t}{(1+r)^t} - C, \tag{4.1}$$

де R_t – доходи за період t, r – ставка дисконтування (10%), C – витрати.

$$NPV = \frac{90\,000}{(1+0.1)^1} - 90\,800 = 81\,818 - 90\,800 = -8\,982 \text{ грн.}$$

Невеликий негативний показник пояснюється високими стартовими витратами.

Рентабельність інвестицій (ROI) [39]:

$$ROI = \frac{\text{Дохід} - \text{Витрати}}{\text{Витрати}} \times 100\%. \tag{4.2}$$

$$ROI = \frac{90\,000 - 90\,800}{90\,800} \times 100\% = -0.88\%.$$

Період окупності:

$$\text{Період окупності} = \frac{\text{Витрати}}{\text{Щомісячний дохід}}. \tag{4.3}$$

Щомісячний дохід (рівномірний розподіл): $90\,000/12 = 7\,500$ грн.

Період окупності $= 90\,800/7\,500 = 12.1$ місяців.

Економічний аналіз свідчить про перспективність проекту. Незначний негативний показник NPV може бути компенсований підвищенням продажів у наступні роки. Програмне забезпечення здатне окупити витрати через 12 місяців та забезпечити стабільний дохід.

5 ОХОРОНА ПРАЦІ

Охорона праці є важливим аспектом трудової діяльності, який спрямований на забезпечення безпеки та здоров'я працівників. Вона включає в себе різні аспекти, такі як правові, соціально-економічні, організаційно-технічні, санітарно-гігієнічні та лікувально-профілактичні заходи та засоби. Основні завдання охорони праці можна розділити на інженерно-технічні та соціальні аспекти.

Інженерно-технічне завдання охорони праці спрямоване на зменшення ризиків, пов'язаних з трудовою діяльністю. Це досягається шляхом застосування нових технологій, розробки безпечних робочих процесів, використання спеціальних устаткувань та засобів захисту. Такі заходи допомагають запобігати нещасним випадкам, професійним захворюванням та іншим ризикам, пов'язаним з роботою.

Соціальне завдання охорони праці включає компенсаційні та захисні заходи, спрямовані на відшкодування шкоди, запобігання несправедливості та захист прав працівників. Це включає компенсацію за шкоду, завдану під час трудового процесу, виплату страхових виплат в разі нещасних випадків, розробку правил та норм, що стосуються безпеки праці, а також захист працівників від дискримінації та несправедливого поводження.

Охорона праці регулюється законом України «Про охорону праці», який визначає права та обов'язки працівників і роботодавців у сфері охорони праці [40]. Цей закон містить норми та вимоги, які стосуються організації робочого процесу, професійної підготовки та інструктажу працівників, використання засобів індивідуального захисту, медичного обстеження та інших аспектів, що стосуються охорони праці.

Забезпечення безпеки та охорони праці є спільною відповідальністю роботодавців, працівників та держави, і вимагає постійного вдосконалення, навчання та контролю для забезпечення безпечних умов праці та попередження можливих ризиків.

5.1 Аналіз шкідливих та небезпечних факторів під час роботи з екранними пристроями

Аналіз шкідливих та небезпечних факторів під час роботи з екранними пристроями є важливою частиною охорони праці, оскільки тривалий час, проведений перед екраном, може викликати різні проблеми зі здоров'ям. Основні шкідливі та небезпечні фактори, пов'язані з роботою з екранними пристроями, включають:

– Вплив візуального навантаження: Постійна фокусування на екрані може призвести до втоми та напруги очей, сухості та подразнення очей, а також до розвитку офтальмологічних захворювань, таких як комп'ютерний синдром.

– Погіршення позиції тіла: Некоректна позиція тіла під час роботи з екраном може призвести до розладів опорно-рухового апарату, наприклад, до болей у шиї, плечах, спині, а також до м'язових напруг і захворювань суглобів.

– Вплив електромагнітного випромінювання: Екранні пристрої випромінюють електромагнітне випромінювання, яке може мати вплив на здоров'я працівників, включаючи можливість розвитку електромагнітної гіперчутливості.

– Погіршення психоемоційного стану: Тривала робота з екраном може спричиняти стрес, збільшену втомлюваність, розлади сну та інші психоемоційні проблеми.

– Регулювання робочого часу: Важливо встановити оптимальну тривалість роботи перед екраном і враховувати перерви для відпочинку та очей. Рекомендується регулярно проводити короткі перерви під час тривалої роботи з екраном.

– Забезпечення правильної позиції тіла: Рекомендується належно налаштувати робоче місце, включаючи висоту стільця та стола, кут нахилу екрана і клавіатури. Привертайте увагу до правильної позиції спини, шиї і рук під час роботи перед екраном.

– Організація робочого простору: Важливо забезпечити належне освітлення робочого місця, уникати відблисків на екрані та регулювати контрастність та яскравість екрану. Крім того, слід стежити за правильним розміщенням робочого обладнання та аксесуарів.

– Застосування засобів індивідуального захисту: Врахуйте можливість використання антиблискових екранів, захисних окулярів або інших засобів індивідуального захисту для зменшення впливу шкідливих факторів.

– Свідоме використання часу поза робочим місцем: Забезпечте собі достатньо часу для фізичної активності, відпочинку і віддалення від екранів поза робочим часом. Це допоможе знизити загальний вплив тривалої роботи з екраном на організм.

– Проведення фізичних вправ: Після тривалої роботи за комп'ютером важливо робити паузи і проводити фізичні вправи для розслаблення м'язів і покращення кровообігу.

– Регулярні огляди зі спеціалістом: Проходження регулярних медичних оглядів та консультацій з офтальмологом може допомогти виявити можливі проблеми зі здоров'ям, пов'язані з роботою за екранними пристроями.

– Корректура робочого часу: Важливо розподіляти робочий час з екранними пристроями раціонально, з урахуванням перерв і відпочинку, щоб уникнути перенавантаження та втоми.

Норми мікроклімату зазначені у санітарних нормах мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [41].

Для аналізу шкідливих та небезпечних факторів під час роботи з екранними пристроями, слід провести оцінку ризику на робочих місцях, що включає:

– Визначення тривалості роботи з екраном та частоти перерв.

– Забезпечення оптимального освітлення: Важливо мати належне освітлення на робочому місці, щоб уникнути зайвого напруження очей. Краще використовувати природне освітлення або штучне освітлення з регульованою яскравістю.

– Перевірка належності робочого місця до ергономічних стандартів, включаючи правильну позицію тіла, регулювання стільця та стола, використання ергономічних пристроїв.

– Оцінка впливу електромагнітного випромінювання та вжиття заходів для зниження можливих ризиків.

– Забезпечення навчання працівників з правил безпеки та користування екранними пристроями.

– Забезпечення належної вентиляції приміщення: Наявність достатньої свіжого повітря у робочій зоні є важливим для підтримки комфортних умов та запобігання накопиченню шкідливих речовин.

– Встановлення акустичного комфорту: Звуковий комфорт є важливим аспектом безпеки та благополуччя працівників. Необхідно використовувати засоби для зменшення шуму, такі як акустичні екрани або навушники з шумозаглушенням.

– Правильне розташування обладнання: Клавіатура, миша та інші пристрої повинні бути розташовані так, щоб забезпечувати природні та комфортні рухи рук і запобігати напруженням м'язів і суглобів.

– Належне організування кабелів: Кабелі повинні бути правильно організовані, щоб уникнути спотикання, заплутування та інших потенційних небезпек.

Загалом, важливо підтримувати баланс та свідомо ставитись до роботи з екранними пристроями, дотримуючись принципів ергономіки та забезпечуючи безпечні умови праці для збереження здоров'я та працездатності. Врахування цих вимог допоможе зменшити вплив небезпечних та шкідливих факторів на здоров'я працівників, покращити комфорт роботи та забезпечити безпечні умови праці.

5.2 Методи нейтралізації або зменшення впливу негативних факторів під час роботи з екранними пристроями

Нейтралізація шкідливих та небезпечних факторів під час роботи з екранними пристроями може бути досягнута за допомогою наступних заходів:

– Використання фільтрів на екрани: Фільтри на екрани можуть допомогти зменшити випромінювання і відблискі, що негативно впливають на очі. Вони зменшують поглинання шкідливих променів і допомагають знизити напруження очей.

– Регулювання яскравості та контрастності екрану: Налаштування яскравості та контрастності екрану на комфортний рівень може знизити напруження очей. Потрібно налаштовувати екран таким чином, щоб було зручно читати текст і розрізняти зображення.

– Використання правильного розташування екрану: Екран повинен бути розташований на відстані близько 50-70 см від очей, з огляду на комфортну зону зору. Правильне розташування екрану допомагає знизити напруження очей і шиї.

– Регулярні перерви та вправи для очей: Важливо робити перерви під час тривалої роботи з екраном. Під час перерв можна виконувати спеціальні вправи для очей, які допомагають зменшити напруження та втому очей.

– Користування антибліковими окулярами: Антиблікові окуляри з спеціальними покриттями можуть допомогти зменшити відблисків і поглинання шкідливого світла, що негативно впливає на очі.

– Організація робочого простору: Належна організація робочого місця може знизити вплив негативних факторів. Наприклад, забезпечення належного освітлення, правильного розташування меблів і пристроїв, використання зручного стільця та клавіатури.

– Здоровий спосіб життя: Загальний стан здоров'я також впливає на толерантність до роботи з екранами. Важливо вести здоровий спосіб життя, забезпечувати достатню фізичну активність, правильне харчування та регулярний сон.

Ці методи можуть допомогти зменшити негативний вплив роботи з екранними пристроями на організм та забезпечити більш комфортні умови праці.

6 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Сучасні інформаційні технології мають значний вплив на навколишнє середовище, який можна оцінити як подвійний. З одного боку, розвиток інформаційних технологій дозволяє знижувати обсяги споживання ресурсів та оптимізувати виробничі процеси, що позитивно позначається на екологічному становищі. Автоматизація, цифровізація та використання кросплатформних рішень дають змогу зменшити витрати на матеріальні ресурси, знизити кількість відходів і скоротити використання паперу. Наприклад, цифрові технології активно сприяють розвитку дистанційної роботи, що, в свою чергу, знижує потребу в транспортуванні та економить енергію.

З іншого боку, розвиток IT-індустрії супроводжується значним зростанням обсягів електронних відходів і підвищеним енергоспоживанням. Серверні ферми та центри обробки даних, що підтримують сучасні інформаційні технології, вимагають великих обсягів енергії, що збільшує навантаження на енергосистему та сприяє підвищенню викидів парникових газів. Виробництво електронного обладнання, зокрема смартфонів, комп’ютерів та серверів, також має негативний екологічний вплив, адже вимагає видобутку рідкісних металів і використання токсичних матеріалів.

6.1 Розробка екологічно чистого програмного забезпечення

Одним із важливих напрямків у зменшенні негативного впливу на навколишнє середовище є розробка енергоефективного програмного забезпечення. Важливим прикладом цього є платформа Flutter, яка дозволяє

створювати кросплатформні додатки. Використання такої архітектури дозволяє уникнути дублювання коду для різних операційних систем, що зменшує навантаження на процесори та знижує енергоспоживання під час розробки і експлуатації додатків.

Програмне забезпечення, розроблене з урахуванням екологічних аспектів, може значно знизити енергоспоживання. Наприклад, оптимізація алгоритмів і зменшення кількості обчислювальних операцій дозволяють зменшити споживання енергії як кінцевими пристроями (мобільними телефонами, комп'ютерами), так і серверами, що знижує їхній вплив на екологію. Крім того, завдяки використанню кросплатформних рішень значно зменшується кількість тестових і нативних додатків, що в свою чергу дозволяє економити ресурси на розробку, тестування та оновлення програмного забезпечення.

6.2 Заходи для зменшення екологічного впливу IT-програм

Для мінімізації негативного впливу інформаційних технологій на навколишнє середовище існує кілька важливих стратегій. Одним із основних підходів є оптимізація програмного коду. Створення ефективного коду, який використовує менше ресурсів пристроїв, допомагає знизити енергоспоживання як на етапі розробки, так і на етапі експлуатації програм. Важливим кроком є впровадження енергоефективних тестувальних методів, коли автоматизовані системи тестування мінімізують кількість непотрібних перевірок і знижують витрати енергії.

Також важливо активно використовувати технології віртуалізації для зменшення кількості фізичних серверів. Віртуальні сервери дозволяють значно скоротити кількість енергозатрат і знизити витрати на обслуговування. В рамках стратегії сталого розвитку важливо також звертати увагу на мінімізацію

використання паперу — всі документи, пов'язані з розробкою і тестуванням програм, повинні оформлюватися в електронному вигляді. Це допоможе зменшити обсяги відходів та заощадити ресурси.

Не менш важливою є й впровадження відновлювальних джерел енергії для підтримки роботи серверів. Переведення дата-центрів на «зелену» енергетику дозволяє зменшити використання традиційних джерел енергії та знизити викиди CO2. Такий підхід є важливою складовою частиною «зеленого IT», яке спрямоване на збереження екологічного балансу.

6.3 Життєвий цикл програмного забезпечення та його екологічний вплив

Життєвий цикл програмного забезпечення охоплює три основні етапи: розробку, використання та утилізацію. Кожен з цих етапів має певний вплив на навколишнє середовище. Під час розробки програмного забезпечення відбувається споживання енергії, оскільки для тестування і обробки даних використовуються комп'ютери та сервери. Тому важливо впроваджувати енергоефективні технології вже на цьому етапі, використовуючи екологічно чисті джерела енергії та енергоефективне обладнання.

На етапі використання програмне забезпечення повинно бути оптимізоване для мінімізації енергоспоживання кінцевими пристроями, такими як мобільні телефони, ноутбуки та інші електронні пристрої. Програмне забезпечення повинно бути розроблене таким чином, щоб знижувати споживану енергію в процесі його роботи, особливо при виконанні завдань, які не потребують постійної високої обчислювальної потужності.

Щодо етапу утилізації, то важливо враховувати екологічні аспекти при списанні старого електронного обладнання. Старі пристрої та сервери повинні бути належно утилізовані, щоб зменшити кількість електронних відходів.

Переробка таких матеріалів дозволяє значно зменшити шкоду для навколишнього середовища, оскільки частина компонентів може бути повторно використана або перероблена без негативного впливу на природу.

6.4 Впровадження екологічних стандартів у розробку програмного забезпечення

Одним із важливих аспектів сталого розвитку інформаційних технологій є впровадження міжнародних екологічних стандартів. Визнані стандарти, такі як ISO 14001 [42], який визначає вимоги до систем екологічного менеджменту, допомагають забезпечити екологічну відповідальність на всіх етапах життєвого циклу програмного забезпечення. Також важливим є використання EPEAT [43], що оцінює екологічний вплив продуктів на довкілля.

Принципи «зеленого IT» мають на меті оптимізацію використання енергетичних та матеріальних ресурсів, скорочення викидів вуглекислого газу, зменшення використання токсичних матеріалів і підтримку довготривалого використання електронних пристроїв. Розробка програмного забезпечення із дотриманням цих стандартів дозволяє знизити негативний екологічний вплив і забезпечити сталий розвиток галузі.

Розробка програмного забезпечення з урахуванням екологічних аспектів є важливим кроком до забезпечення сталого розвитку інформаційних технологій. Впровадження енергоефективних рішень, дотримання міжнародних стандартів і використання екологічно чистих технологій дозволяють зменшити негативний вплив на навколишнє середовище. Такі заходи сприяють збереженню природних ресурсів і зменшенню екологічного сліду IT-індустрії, що є важливим для забезпечення довготривалого і сталого розвитку галузі.

ВИСНОВКИ

У кваліфікаційній роботі було удосконалено математичну модель для оцінювання якості програмних продуктів на платформі Flutter, яка базується на метриках CBO та WMC для прогнозування метрики RFC. Аналіз існуючих підходів показав, що традиційні моделі, розроблені для Java та Kotlin, не підходять для Flutter через відмінності в архітектурі та діапазонах метрик. Це підкреслило необхідність у новій моделі, спеціалізованій для Flutter. Зібрано та проаналізовано дані зі 100 відкритих Flutter-застосунків. Після обробки даних та видалення викидів сформовано вибірку з 89 застосунків, які використано для побудови моделі.

Загалом, розроблена модель надають можливість ефективно оцінювати якість застосунків, розроблених на платформі Flutter. Це важливо для підвищення стабільності та надійності програмного забезпечення, що особливо актуально в умовах швидкого розвитку IT-індустрії.

Було успішно створено програмне забезпечення «FlutterStats» для оцінювання якості Flutter-проектів, яка виконує обробку даних, будує регресійну модель. Розроблена система «FlutterStats» дозволяє ефективно аналізувати якість застосунків, реалізованих за допомогою фреймворку Flutter. Вона включає в себе різні метрики, такі як RFC, CBO та WMC, що допомагають у оцінці проектів. Модель продемонструвала високу точність прогнозування на тестових даних, що підтверджує її ефективність для оцінювання якості Flutter-додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Laing V., Coleman C. Principal Components of Orthogonal Object-Oriented Metrics. White Paper Analyzing Results of NASA Object-Oriented Data (Task 323-08-14). Greenbelt, Maryland (US): NASA Goddard Space Flight Center, the Software Assurance Technology Center (SATC), 2001.

2. 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, 2022. P. 499–502. DOI: https://doi.org/10.1109/CSIT56902.2022.10000532.

3. ISO/IEC 9126 in Software Engineering [Електронний ресурс]. URL: https://www.geeksforgeeks.org/iso-iec-9126-in-software-engineering/ (дата звернення: 06.10.2024).

4. Якість програмного забезпечення [Електронний ресурс]. URL: https://qalight.ua/baza-znaniy/yakist-programnogo-zabezpechennya/ (дата звернення: 06.10.2024).

5. McCall’s Quality Model [Електронний ресурс]. URL: https://www.geeksforgeeks.org/mccalls-quality-model/ (дата звернення: 12.10.2024).

6. Boehm’s software Quality Model [Електронний ресурс]. URL: https://www.researchgate.net/figure/Boehms-software-Quality-Model_fig4_328460644 (дата звернення: 12.10.2024).

7. Halstead & McCabe metrics: The wisdom of the ancients [Електронний ресурс]. URL: https://shape-of-code.com/2023/09/03/halstead-mccabe-metrics-the-wisdom-of-the-ancients/ (дата звернення: 06.10.2024).

8. Chidamber & Kemerer object-oriented metrics suite [Електронний ресурс]. URL: https://www.aivosto.com/project/help/pm-oo-ck.html (дата звернення: 13.10.2024).

9. Довідник по Machine Learning – Support Vector Machines [Електронний ресурс]. URL: https://itwiki.dev/data-science/ml-reference/ml-glossary/support-vector-machines (дата звернення: 08.10.2024).

10. Response for a Class | DCM - Code Quality Tool for Flutter Developers [Електронний ресурс]. URL: https://dcm.dev/docs/metrics/class/response-for-class/ (дата звернення: 08.10.2024).

11. Coupling Between Object Classes | DCM - Code Quality Tool for Flutter Developers [Електронний ресурс]. URL: https://dcm.dev/docs/metrics/class/coupling-between-object-classes/ (дата звернення: 08.10.2024).

12. Weighted Methods Per Class | DCM - Code Quality Tool for Flutter Developers [Електронний ресурс]. URL: https://dcm.dev/docs/metrics/class/weighted-methods-per-class/ (дата звернення: 08.10.2024).

13. Flutter. Build apps for any screen [Електронний ресурс]. URL: https://flutter.dev/ (дата звернення: 02.10.2024).

14. Dart Programming language | Dart [Електронний ресурс]. URL: https://dart.dev/ (дата звернення: 06.10.2024).

15. Material Design [Електронний ресурс]. URL: https://m3.material.io/ (дата звернення: 02.10.2024).

16. Widgets | Flutter [Електронний ресурс]. URL: https://docs.flutter.dev/ui/widgets (дата звернення: 02.10.2024).

17. Reactive programming in Flutter [Електронний ресурс]. URL: https://medium.com/@alvaro.armijoss/reactive-programming-in-flutter-c577d8e056d2 (дата звернення: 05.10.2024).

18. Приходько С.Б., Шебалін Н.О. Нелінійна регресійна модель для оцінювання метрики RFC застосунків, що створюються на платформі Flutter // Інформаційні технології: моделі, алгоритми, системи: V-я Всеукраїнська науково-практична інтернет-конференція (ITMAS–2024). Миколаїв, Україна, 2024. С. 43–44. URL: https://itconf.nuos.edu.ua/2024/wp-content/uploads/sites/7/NUOS-ITMAS-2024-Proceedings.pdf.

19. Приходько С.Б., Позняков Р.О. Нелінійна регресійна модель для оцінювання метрики RFC застосунків з відкритим кодом на Java // Інформаційні технології: моделі, алгоритми, системи: III-я Всеукраїнська науково-практична інтернет-конференція (ITMAS–2022). Миколаїв, Україна, 2022. С. 8–9. URL: http://itconf.nuos.edu.ua/2022/wp-content/uploads/sites/5/NUOS-ITMAS-2022-Proceedings.pdf.

20. Приходько С.Б., Кольцов А.В., Грабовський Є.В. Нелінійна регресійна модель для оцінювання метрики RFC застосунків з відкритим кодом на Kotlin // Інформаційні технології: моделі, алгоритми, системи: IV-я Всеукраїнська науково-практична інтернет-конференція (ITMAS–2023). Миколаїв, Україна, 2023. С. 20–21. URL: https://itconf.nuos.edu.ua/2023/wp-content/uploads/sites/6/NUOS-ITMAS-2023-Proceedings.pdf.

21. Оцінка складності програм за допомогою метрики Холстеда [Електронний ресурс]. URL: https://conf.ztu.edu.ua/wp-content/uploads/2016/06/29.pdf (дата звернення: 07.10.2024).

22. Tan H.B.K., Zhao Y., Zhang H. Estimating LOC for information systems from their conceptual data models // Proceedings of the 28th International Conference on Software Engineering (ICSE'06). Shanghai, China, 2006. P. 321–330. DOI: https://dl.acm.org/doi/10.1145/1134285.1134331.

23. Software Quality Metrics [Електронний ресурс]. URL: https://elitex.systems/blog/software-quality/ (дата звернення: 07.10.2024).

24. Master System Reliability [Електронний ресурс]. URL: https://www.squadcast.com/blog/system-reliability-metrics-a-comparative-guide-to-mttr-mtbf-mttd-and-mttf (дата звернення: 09.10.2024).

25. State management [Електронний ресурс]. URL: https://docs.flutter.dev/data-and-backend/state-mgmt/intro (дата звернення: 07.10.2024).

26. Asynchronous programming: futures, async, await [Електронний ресурс]. URL: https://dart.dev/libraries/async/async-await (дата звернення: 07.10.2024).

27. Flutter architectural overview [Електронний ресурс]. URL: https://docs.flutter.dev/resources/architectural-overview (дата звернення: 07.10.2024).

28. DCM – Code Quality Tool for Flutter Developers [Електронний ресурс]. URL: https://dcm.dev/ (дата звернення: 08.10.2024).

29. Mardia K.V. Measures of multivariate skewness and kurtosis with applications // Biometrika. 1970. Vol. 57. P. 519–530. DOI: https://doi.org/10.1093/biomet/57.3.519.

30. Mahalanobis Distance [Електронний ресурс]. URL: https://www.machinelearningplus.com/statistics/mahalanobis-distance/ (дата звернення: 22.10.2024).

31. Normal distribution [Електронний ресурс]. URL: https://www.investopedia.com/terms/n/normaldistribution.asp (дата звернення: 15.10.2024).

32. Afifi A.A., Azen S.P. Statistical analysis: a computer-oriented approach. New York; London: Academic Press, 1972.

33. F Distribution [Електронний ресурс]. URL: https://www.sciencedirect.com/topics/mathematics/f-distribution (дата звернення: 15.10.2024).

34. Ordinary Least Squares regression (OLS) [Електронний ресурс]. URL: https://www.xlstat.com/en/solutions/features/ordinary-least-squares-regression-ols (дата звернення: 15.10.2024).

35. Chi-Square (X^2) Distributions [Електронний ресурс]. URL: https://www.statlect.com/probability-distributions/chi-square-distribution (дата звернення: 16.10.2024).

36. MVC [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (дата звернення: 02.11.2024).

37. Visual Studio Code - Code Editing. Redefined [Електронний ресурс]. URL: https://code.visualstudio.com/ (дата звернення: 02.11.2024).

38. Net Present Value (NPV) [Електронний ресурс]. URL: https://www.investopedia.com/terms/n/npv.asp (дата звернення: 02.11.2024).

39. What Is ROI? How to Calculate Return on Investment [Електронний ресурс]. URL: https://www.techtarget.com/searchcio/definition/ROI (дата звернення: 02.11.2024).

40. Про затвердження Вимог щодо безпеки та захисту здоров’я працівників: Закон України від 14.02.2018 № 207 [Електронний ресурс]. URL: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 09.11.2024).

41. Санітарні норми мікроклімату виробничих приміщень (ДСН 3.3.6.042-99) [Електронний ресурс]. URL: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення: 09.11.2024).

42. ISO 14001:2015 - Environmental management systems [Електронний ресурс]. URL: https://www.iso.org/standard/60857.html (дата звернення: 09.11.2024).

43. EPEAT Registry [Електронний ресурс]. URL: https://www.epeat.net/ (дата звернення: 09.11.2024).

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «FlutterStats»

Вступ

Повна назва програмного продукту: Програмне забезпечення «FlutterStats» для визначення якості застосунків, що створюються з використанням фреймворку Flutter.

Коротка назва: «FlutterStats».

Сфера застосування: компанії, що займаються розробкою програмного забезпечення з використанням фреймворку Flutter.

1. Підстави для розробки

Підставою для розробки «FlutterStats» є наказ на затвердження тем кваліфікаційних робіт магістрів зі спеціальності 121 «Інженерія програмного забезпечення» в Національному університеті кораблебудування ім. адмірала Макарова.

2. Призначення розробки

2.1 Експлуатаційне призначення

Експлуатаційним призначенням є полегшення процесу планування та управління розробкою програмних продуктів та автоматизація розрахунку прогнозних значень якості проектів, що створюються з використанням фреймворку Flutter.

2.2 Функціональне призначення

Функціональним призначенням «FlutterStats» є завантаження та перегляд метрик Flutter проектів, побудова регресійної моделі для прогнозування якості проектів.

3. Вимоги до програми або програмного виробу

3.1 Вимоги до функціональних характеристик

3.1.1 Вимоги до складу виконуваних функцій

– завантаження набору даних у форматі csv;

– перегляд завантажених метрик проектів (RFC, CBO, WMC);

– виявлення та вилучення викидів;

– розділення даних на навчальну (60%) та тестову (40%) вибірки;

– побудова лінійної регресійної моделі методом найменших квадратів;

– побудова нелінійної регресійної моделі методом зворотного перетворення;

– аналіз якості побудованої моделі за статистичними критеріями;

– побудова довірчих інтервалів та інтервалів прогнозування;

– використання моделі для оцінки якості проекту на основі заданих RFC, CBO та WMC;

– налаштування застосунку (рядки та метрики із .csv файлу для завантаження).

3.1.2 Вимоги до організації вхідних та вихідних даних

Дані про метрики та модель зберігаються у локальній базі даних. СУБД забезпечує розмежування прав доступу до даних – надає користувачу права на читання та запис.

Вхідні дані:

– файл у форматі CSV, що містять метрики Flutter-проектів: RFC, CBO, WMC;

– метрики RFC, CBO та WMC для визначення якості.

Вихідні дані:

– список завантажених проектів та їх поділ на дві вибірки для побудови та тестування моделі;

– викиди;

– параметри побудованої лінійної моделі;

– параметри нелінійної регресійної моделі;

– показники якості нелінійної регресійної моделі;

– показники якості тестової вибірки для побудованої моделі;

– довірчі інтервали;

– інтервали прогнозування;

– допустимі інтервали значень метрик RFC, CBO та WMC;

– прогнозні значення якості проектів.

3.2 Вимоги до надійності

3.2.1 Вимоги до забезпечення надійного (сталого) функціонування програми

Надійне (стійке) функціонування програми має бути забезпечене виконанням сукупності організаційно-технічних заходів, перелік яких подано нижче:

– захист від некоректних дій користувача (під час автентифікації, аналізу);

– встановлення Flutter Framework не нижче версії 2.18.6.

3.2.2 Час відновлення після відмови

Вимоги до часу відновлення після відмови не пред’являються.

3.2.3 Відмови через некоректні дії оператора

Відмова можлива при наступних умовах:

– внаслідок закриття програми під час аналізу, в цьому випадку користувачеві показується додаткове вікно, чи впевнений він що хоче закрити програму.

3.3 Умови експлуатації

3.3.1 Кліматичні умови експлуатації

Кліматичні умови експлуатації, при яких повинні забезпечуватися задані характеристики, повинні відповідати вимогам, що пред'являються до технічних засобів в частині умов їх експлуатації.

3.3.2 Вимоги до чисельності та кваліфікації користувачів

Програму використовую один користувач, який повинен мати базові навички роботи з комп’ютером.

3.4 Вимоги до складу і параметрів технічних засобів

Для роботи із програмним забезпеченням необхідний наступний склад технічних засобів з мінімальними параметрами:

– Процесор Intel Pentium 1Ghz;

– Оперативна пам'ять 512Мб;

– Дискового простору 1 Гб;

– Роздільна здатність екрану 800x600 пікселів;

– Інтернет-з’єднання. Програмне забезпечення вимагає доступу до Інтернету.

3.5 Вимоги до інформаційної та програмної сумісності

3.5.1 Вимоги до вихідних кодів і мов програмування

Системні програмні засоби, які використовуються програмою, повинні бути представлені ліцензійною версією операційної системи Windows (версії не нижче Windows 7).

В якості інтегрованого середовища розробки програми буде використовуватися середовище Microsoft Visual Studio Code. Використовувати Flutter Framework версії не нижче 2.18.6. Вихідні коди програми повинні бути реалізовані на мові Dart.

3.6 Спеціальні вимоги

Програма повинна забезпечувати взаємодію з користувачем за допомогою графічного інтерфейсу користувача, розробленого відповідно до рекомендацій компанії-виробника операційної системи.

4. Вимоги до програмної документації

Склад програмної документації повинен включати в себе:

– технічне завдання;

– програму та методику випробувань;

– керівництво користувача;

– опис програми;

– код програми.

5. Техніко-економічні показники

У даній кваліфікаційній роботі техніко-економічні показники не розраховуються.

6. Етапи роботи

Етапи роботи представлені у таблиці А.1

Таблиця А.1 – Етапи роботи

№ п/п	Назва етапів роботи створення програмного забезпечення	Кінцевий термін виконання
1	2	3
1	Аналіз вимог до ПЗ	10.09.2024
2	Розробка архітектури ПЗ	21.09.2024
3	Розробка проекту ПЗ	16.10.2024
4	Реалізація та тестування ПЗ	26.10.2024
5	Випробування ПЗ	1.11.2024

7. Порядок контролю та приймання

Контроль за аналізом та проектуванням кожної окремої частини програмного забезпечення на кожному етапі з урахуванням вимог, визначених у технічному завдані. Кожна стадія розробки повинна бути представлена в зазначені терміни та узгоджена із замовником.

Прийом проводяться відповідно до програми і методики випробувань і документують за допомогою протоколу проведення випробувань. У разі знаходження помилок підчас прийому програмного виробу складається акт про знайдені помилки, який підписуються представниками замовника і розробника і затверджуються керівником організації – замовника та організації – розробника. Розробник повинен на протязі не більше ніж 2 тижнів виправити зазначені помилки і оповістити замовника про повторне проведення перевірки.

ДОДАТОК Б – КОД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «FlutterStats»

Програмне забезпечення «FlutterStats». Повний код програми наведено у GitHub URL: https://github.com/shebnik/flutter_stats/.

lib/main.dart

```
import 'dart:async';

import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter_stats/app/app.dart';
import 'package:flutter_stats/services/logging/logger_config.dart';
import 'package:flutter_stats/services/logging/logger_service.dart';
import 'package:flutter_web_plugins/url_strategy.dart';
import 'package:path/path.dart' as path;
import 'package:shared_preferences/shared_preferences.dart';
import 'package:universal_io/io.dart';

Future<void> main() async {
  await runZonedGuarded(() async {
    WidgetsFlutterBinding.ensureInitialized();

    final logger = await LoggerService.initialize(
      config: LoggerConfig(
        maxFileSize: 5 * 1024 * 1024, // 5MB
        logDirectory: path.join('FlutterStats', 'logs'),
        minLogLevel: kDebugMode ? LogLevel.debug : LogLevel.info,
        enableFileOutput: !kIsWeb,
      ),
    );
    logger.i(
      'Application started',
      metadata: {
        'version': '1.0.0',
        'buildNumber': '42',
        'isWeb': kIsWeb,
        'platform': Platform.operatingSystem,
        'operatingSystemVersion': Platform.operatingSystemVersion,
        'localHostname': Platform.localHostname,
        'numberOfProcessors': Platform.numberOfProcessors,
      },
```

```
    );

    usePathUrlStrategy();
    registerErrorHandlers(logger);

    final sp = await SharedPreferencesWithCache.create(
      cacheOptions: const SharedPreferencesWithCacheOptions(),
    );
    runApp(App(sp: sp));
  }, (error, stackTrace) {
    LoggerService.instance.e(
      'Error',
      error: error,
      stackTrace: stackTrace,
    );
  });
}

void registerErrorHandlers(LoggerService logger) {
  FlutterError.onError = (details) {
    logger.e(
      'FlutterError',
      error: details.exception,
      stackTrace: details.stack,
    );
  };

  PlatformDispatcher.instance.onError = (Object error, StackTrace stack) {
    logger.e(
      'PlatformDispatcher Error',
      error: error,
      stackTrace: stack,
    );
    return true;
  };
}
```

lib/services/outliers.dart

```
import 'dart:math';

import 'package:flutter/material.dart';
import 'package:flutter_stats/constants.dart';
import 'package:flutter_stats/models/project/project.dart';
import 'package:flutter_stats/models/regression_factors/regression_factors.dart';
import 'package:flutter_stats/services/algebra.dart';
import 'package:flutter_stats/services/fisher.dart';
import 'package:flutter_stats/services/logging/logger_service.dart';
```

```
import 'package:flutter_stats/services/normalization.dart';
import 'package:flutter_stats/services/regression_model.dart';
import 'package:flutter_stats/services/utils.dart';

enum OutliersType { epsilon, predictionInterval }

class OutliersIterations {
  OutliersIterations({
    required this.step,
    required this.projects,
    required this.type,
    required this.whyOutliers,
  });

  final int step;
  final List<Project> projects;
  final OutliersType type;
  final List<String> whyOutliers;
  int get count => projects.length;
}

class OutliersProvider extends ChangeNotifier {
  final Algebra _algebra = Algebra();
  final Normalization _normalization = Normalization();
  final _log = LoggerService.instance;

  List<OutliersIterations> _iterations = [];
  List<OutliersIterations> get iterations => _iterations;

  Future<List<Project>> removeAllOutliers(List<Project> projects) async {
    _log.i('Removing all outliers');
    _iterations = [];
    var outliers = <Project>[];
    var whyOutliers = <String>[];
    var iteration = 1;
    do {
      (whyOutliers, outliers) = await determineEpsilonOutliers(projects);
      if (outliers.isNotEmpty) {
        _iterations.add(
          OutliersIterations(
            step: iteration,
            projects: outliers,
            whyOutliers: whyOutliers,
            type: OutliersType.epsilon,
          ),
        );
        projects.removeWhere((p) => outliers.contains(p));
        iteration++;
```

```
    }
  } while (outliers.isNotEmpty);

  do {
    (whyOutliers, outliers) = await determinePredictionIntervalOutliers(
      projects,
    );
    if (outliers.isNotEmpty) {
      _iterations.add(
        OutliersIterations(
          step: iteration,
          projects: outliers,
          whyOutliers: whyOutliers,
          type: OutliersType.predictionInterval,
        ),
      );
      projects.removeWhere((p) => outliers.contains(p));
      iteration++;
    }
  } while (outliers.isNotEmpty);

  _iterations = List.from(_iterations);
  notifyListeners();

  return projects;
}

List<double> calculateMahalanobisDistances(List<RegressionFactors> factors) {
  final n = factors.length;
  final p = factors.first.x.length + 1;
  final values = factors.toArray();
  final covInv =
      _algebra.invertMatrix(_algebra.calculateCovarianceMatrix(values));
  final means = values.map(_algebra.average).toList();

  return List.generate(n, (i) {
    final diff = List<double>.generate(p, (j) => values[j][i] - means[j]);
    final product = List<double>.generate(
      p,
      (j) => List.generate(p, (k) => covInv[j][k] * diff[k])
          .reduce((a, b) => a + b),
    );
    return List.generate(p, (j) => product[j] * diff[j])
        .reduce((a, b) => a + b);
  });
}

List<double> calculateTestStatistics(List<RegressionFactors> factors) {
```

```
  final distances = calculateMahalanobisDistances(factors);
  final n = factors.length;
  final p = factors.first.x.length + 1;
  final factor = (n - p) * n / ((pow(n, 2) - 1) * p);
  return distances.map((d) => factor * d).toList();
}

Future<double?> calculateFisherFDistribution({
  required int n,
  required int p,
}) =>
    Fisher.inv(alpha: alpha, df1: p, df2: n - p);

Future<(List<String>, List<Project>)> determineEpsilonOutliers(
  List<Project> projects,
) async {
  _log.d('Epsilon Outlier Detection Started');
  final factors = _normalization
      .normalizeProjects(projects)
      .map(RegressionFactors.fromProject)
      .toList();
  final n = factors.length;
  final p = factors.first.x.length + 1;
  final testStatistics = calculateTestStatistics(factors);
  final f = await calculateFisherFDistribution(n: n, p: p);
  if (f == null) {
    debugPrint('Failed to calculate Fisher F distribution');
    return (<String>[], <Project>[]);
  }
  _log.i('Fisher F distribution: $f');

  final outliers = <Project>[];
  final whyOutliers = <String>[];
  for (var i = 0; i < n; i++) {
    if (testStatistics[i] > f) {
      final outlier = projects[i];
      final why = 'Test Statistic: ${Utils.formatNumber(testStatistics[i])} '
          '> F: ${Utils.formatNumber(f)}';
      _log.d('Epsilon Outlier detected: ${outlier.url} $why');
      outliers.add(outlier);
      whyOutliers.add(why);
    }
  }
  return (whyOutliers, outliers);
}

Future<(List<String>, List<Project>)> determinePredictionIntervalOutliers(
  List<Project> projects,
```

```
) async {
  _log.i('Prediction Interval Outlier Detection Started');
  final regressionModel = RegressionModel(projects);
  final factors = _normalization
      .normalizeProjects(projects)
      .map(RegressionFactors.fromProject)
      .toList();
  final n = factors.length;
  final coefficients =
      regressionModel.calculateRegressionCoefficients(factors: factors);
  final zyHat = regressionModel.calculatePredictedValues(
    factors: factors,
    coefficients: coefficients,
  );
  final intervals = await regressionModel.calculateIntervals(
    z: factors.map((f) => f.x).toList(),
    zy: factors.map((f) => f.y).toList(),
    zyHat: zyHat,
  );

  final outliers = <Project>[];
  final whyOutliers = <String>[];
  for (var i = 0; i < n; i++) {
    final actualValue = pow(10, factors[i].y);
    if (actualValue < intervals.predictionLower[i]) {
      final outlier = projects[i];
      whyOutliers.add('Actual: ${Utils.formatNumber(actualValue.toDouble())} '
          '< Lower: ${Utils.formatNumber(intervals.predictionLower[i])}');
      _log.d(
        'Prediction Outlier detected: ${outlier.url} '
        '$actualValue < ${intervals.predictionLower[i]} '
        '(actual < lower)',
      );
      outliers.add(outlier);
    }
    if (actualValue > intervals.predictionUpper[i]) {
      final outlier = projects[i];
      whyOutliers.add('Actual: ${Utils.formatNumber(actualValue.toDouble())} '
          '> Upper: ${Utils.formatNumber(intervals.predictionUpper[i])}');
      _log.d(
        'Prediction Outlier detected: ${outlier.url} '
        '$actualValue > ${intervals.predictionUpper[i]} '
        '(actual > upper)',
      );
      outliers.add(outlier);
    }
  }
  return (whyOutliers, outliers);
```

```
  }
}
```

lib/services/regression_model.dart

```
import 'dart:async';
import 'dart:math';

import 'package:flutter_stats/constants.dart';
import 'package:flutter_stats/models/coefficients/coefficients.dart';
import 'package:flutter_stats/models/intervals/intervals.dart';
import 'package:flutter_stats/models/model_quality/model_quality.dart';
import 'package:flutter_stats/models/project/project.dart';
import 'package:flutter_stats/models/regression_factors/regression_factors.dart';
import 'package:flutter_stats/services/algebra.dart';
import 'package:flutter_stats/services/logging/logger_service.dart';
import 'package:flutter_stats/services/normalization.dart';
import 'package:flutter_stats/services/outliers.dart';
import 'package:flutter_stats/services/student.dart';
import 'package:ml_linalg/matrix.dart';
import 'package:ml_linalg/vector.dart';

class RegressionModel {
  RegressionModel(
    this._projects, {
    List<Project>? trainProjects,
    List<Project>? testProjects,
  }) {
    if (trainProjects != null && testProjects != null) {
      _trainProjects = trainProjects;
      _testProjects = testProjects;
    } else {
      _splitData();
    }
    _evaluateModel();
  }

  final _log = LoggerService.instance;
  final Algebra _algebra = Algebra();
  final Normalization _normalization = Normalization();

  late final List<Project> _projects;
  late final List<Project> _trainProjects;
  late final List<Project> _testProjects;

  List<Project> get projects => _projects;
  List<Project> get trainProjects => _trainProjects;
  List<Project> get testProjects => _testProjects;
```

```
List<RegressionFactors> _factors = [];
List<RegressionFactors> get factors => _factors;

Coefficients _coefficients = Coefficients.empty();
Coefficients get coefficients => _coefficients;

List<double> _predictedValues = [];

ModelQuality _modelQuality = ModelQuality.empty();
ModelQuality get modelQuality => _modelQuality;

ModelQuality _testModelQuality = ModelQuality.empty();
ModelQuality get testModelQuality => _testModelQuality;

Intervals _intervals = Intervals.empty();
Intervals get intervals => _intervals;

int get p => (_factors.firstOrNull?.x.length ?? 0) + 1;

MinMaxFactors get minMaxFactors {
  final values = _trainProjects.map((p) => p.metrics).toList();
  return MinMaxFactors(
    y: MinMax(
      values.map((f) => f.y).reduce(min),
      values.map((f) => f.y).reduce(max),
    ),
    x1: MinMax(
      values.map((f) => f.x1).reduce(min),
      values.map((f) => f.x1).reduce(max),
    ),
    x2: values.first.x2 != null
        ? MinMax(
            values.map((f) => f.x2!).reduce(min),
            values.map((f) => f.x2!).reduce(max),
          )
        : null,
    x3: values.first.x3 != null
        ? MinMax(
            values.map((f) => f.x3!).reduce(min),
            values.map((f) => f.x3!).reduce(max),
          )
        : null,
  );
}

void _splitData([double trainRatio = 0.6]) {
  final projects = List<Project>.from(_projects);
```

```
  // Sort projects by each metric
  // ignore: cascade_invocations
  projects.sort((a, b) => a.metrics.y.compareTo(b.metrics.y));
  final sortedByY = List<Project>.from(projects);

  projects.sort((a, b) => a.metrics.x1.compareTo(b.metrics.x1));
  final sortedByX1 = List<Project>.from(projects);

  var sortedByX2 = <Project>[];
  if (projects.first.metrics.x2 != null) {
    projects.sort((a, b) => a.metrics.x2!.compareTo(b.metrics.x2!));
    sortedByX2 = List<Project>.from(projects);
  }

  var sortedByX3 = <Project>[];
  if (projects.first.metrics.x3 != null) {
    projects.sort((a, b) => a.metrics.x3!.compareTo(b.metrics.x3!));
    sortedByX3 = List<Project>.from(projects);
  }

  // Get min and max projects for each metric
  final minMaxProjects = {
    sortedByY.first,
    sortedByY.last,
    sortedByX1.first,
    sortedByX1.last,
    if (sortedByX2.isNotEmpty) sortedByX2.first,
    if (sortedByX2.isNotEmpty) sortedByX2.last,
    if (sortedByX3.isNotEmpty) sortedByX3.first,
    if (sortedByX3.isNotEmpty) sortedByX3.last,
  };

  // Remove min and max projects from the main list and shuffle the rest
  final remainingProjects = projects
      .where((p) => !minMaxProjects.contains(p))
      .toList()
    ..shuffle(Random());

  // Calculate the number of additional projects needed for the training set
  final nTrain =
      (projects.length * trainRatio).toInt() - minMaxProjects.length;

  // Split the remaining projects
  _trainProjects = [...minMaxProjects, ...remainingProjects.take(nTrain)];
  _testProjects = remainingProjects.skip(nTrain).toList();
}
```

```
Coefficients calculateRegressionCoefficients({
  List<RegressionFactors>? factors,
}) {
  factors ??= _factors;
  final n = factors.length;
  final p = this.p;

  if (n == 0 || p == 0) {
    _log.e('Insufficient data to calculate regression coefficients.');
    return Coefficients.empty();
  }

  // Construct the design matrix X and the target vector y
  final xData = factors.map((f) => f.x).toList();
  final yData = factors.map((f) => f.y).toList();

  // Add a column of 1s for the intercept term
  final designMatrixData =
      xData.map((x) => Vector.fromList([1.0, ...x])).toList();
  final X = Matrix.fromRows(designMatrixData);
  final y = Vector.fromList(yData);

  // Calculate X^T * X
  final xtX = X.transpose() * X;

  // Calculate (X^T * X)^-1
  Matrix xtXInverse;
  try {
    xtXInverse = xtX.inverse();
  } catch (e) {
    _log.e('Matrix inversion failed: $e');
    return Coefficients.empty();
  }

  // Calculate X^T * y
  final xty = X.transpose() * y;

  // Calculate the coefficients: (X^T * X)^-1 * (X^T * y)
  final coefficients = xtXInverse * xty;

  return Coefficients(
    b: List<double>.from(coefficients.map((value) => value.first)),
    epsilon: 0,
  );
}

double _calculateEpsilon(List<double> y, List<double> predictedY) {
  final residuals = List.generate(y.length, (i) => y[i] - predictedY[i]);
```

```
  final n = residuals.length;
  return sqrt(
    residuals.map((r) => pow(r, 2)).reduce((a, b) => a + b) / (n - 3),
  );
}

List<double> calculatePredictedValues({
  List<RegressionFactors>? factors,
  Coefficients? coefficients,
}) {
  factors ??= _factors;
  coefficients ??= _coefficients;
  return List.generate(factors.length, (i) {
    final factor = factors![i];
    var prediction = coefficients!.b[0];
    for (var j = 1; j < coefficients.b.length; j++) {
      prediction += coefficients.b[j] * factor.x[j - 1];
    }
    return prediction;
  });
}

ModelQuality _calculateModelQuality({
  required List<double> y,
  required List<double> yHat,
  bool normalized = true,
}) {
  if (normalized) {
    y = _normalization.revertNormalization(y);
    yHat = _normalization.revertNormalization(yHat);
  }

  final n = y.length;

  final yDiffSquared = List.generate(n, (i) => pow(y[i] - yHat[i], 2));
  final yAvgDiffSquared =
      y.map((value) => pow(value - _algebra.average(y), 2)).toList();

  final yDividedDiff = List.generate(n, (i) => (y[i] - yHat[i]) / y[i]);

  final sy = yDiffSquared.reduce((a, b) => a + b);
  final rSquared = 1 - (sy / yAvgDiffSquared.reduce((a, b) => a + b));

  final mmre =
      yDividedDiff.map((diff) => diff.abs()).reduce((a, b) => a + b) / n;

  final pred = yDividedDiff.where((diff) => diff.abs() < 0.25).length / n;
```

```
  return ModelQuality(
    rSquared: rSquared,
    mmre: mmre,
    pred: pred,
  );
}

Future<void> _evaluateModel() async {
  final normalizedProjects = _normalization.normalizeProjects(_trainProjects);
  _factors = normalizedProjects.map(RegressionFactors.fromProject).toList();
  _log.i('Factors: ${_factors.length}');

  _coefficients = calculateRegressionCoefficients();
  _log.i('Coefficients: ${_coefficients.b}');

  _predictedValues = calculatePredictedValues();
  _log.i('Predicted values ${_predictedValues.length}: $_predictedValues');

  _coefficients = _coefficients.copyWith(
    epsilon: _calculateEpsilon(
      _factors.map((e) => e.y).toList(),
      calculatePredictedValues(),
    ),
  );

  _modelQuality = _calculateModelQuality(
    y: _factors.map((f) => f.y).toList(),
    yHat: _predictedValues,
  );
  _log.i('Model quality: $_modelQuality');
  _intervals = await calculateIntervals(
    zyHat: _predictedValues,
    z: _factors.map((f) => f.x).toList(),
    zy: factors.map((f) => f.y).toList(),
  );
  await _testModel();
}

Future<void> _testModel() async {
  final normalizedProjects = _normalization.normalizeProjects(_trainProjects);
  final normalizedTestProjects =
      _normalization.normalizeProjects(_testProjects);
  final factors =
      normalizedTestProjects.map(RegressionFactors.fromProject).toList();

  _testModelQuality = _calculateModelQuality(
    y: factors.map((f) => f.y).toList(),
    yHat: calculatePredictedValues(
```

```
      factors: factors,
      coefficients: _coefficients,
    ),
  );
  _log.i('Test model quality: $_testModelQuality');

  await _testProjectsQuality(
    [...normalizedProjects, ...normalizedTestProjects],
  );
}

Future<void> _testProjectsQuality(List<Project> projects) async {
  var highQualityCount = 0;
  var mediumQualityCount = 0;
  var lowQualityCount = 0;

  final factors = _normalization
      .normalizeProjects(projects)
      .map(RegressionFactors.fromProject)
      .toList();
  final coefficients = calculateRegressionCoefficients(factors: factors);
  final zyHat = calculatePredictedValues(
    factors: factors,
    coefficients: coefficients,
  );
  final intervals = await calculateIntervals(
    z: factors.map((f) => f.x).toList(),
    zy: factors.map((f) => f.y).toList(),
    zyHat: zyHat,
  );

  for (var i = 0; i < projects.length; i++) {
    final project = projects[i];
    final y = pow(10, project.metrics.y);

    final predLower = pow(10, intervals.predictionLower[i]);
    final predUpper = pow(10, intervals.predictionUpper[i]);
    final confLower = pow(10, intervals.confidenceLower[i]);
    final confUpper = pow(10, intervals.confidenceUpper[i]);

    if (y > predUpper) {
      // RFC exceeds prediction interval upper bound
      lowQualityCount++;
    } else if (y >= confLower && y <= confUpper) {
      // RFC falls within confidence interval
      mediumQualityCount++;
    } else if ((y > confUpper && y <= predUpper) ||
        (y >= predLower && y < confLower)) {
```

```
      // RFC falls between confidence and prediction intervals
      highQualityCount++;
    } else {
      _log.w('Project ${project.url} has unusual Y value: $y');
    }
  }

  final totalProjects = projects.length;
  final highQualityPercentage =
      (highQualityCount / totalProjects * 100).toStringAsFixed(1);
  final mediumQualityPercentage =
      (mediumQualityCount / totalProjects * 100).toStringAsFixed(1);
  final lowQualityPercentage =
      (lowQualityCount / totalProjects * 100).toStringAsFixed(1);

  _log
    ..i('Quality distribution of $totalProjects projects:')
    ..i('High quality: $highQualityPercentage% projects')
    ..i('Medium quality: $mediumQualityPercentage% projects')
    ..i('Low quality: $lowQualityPercentage% projects');
}

double predictY(List<double> x) {
  var prediction = pow(10, _coefficients.b[0] + _coefficients.epsilon);
  for (var i = 1; i < _coefficients.b.length; i++) {
    prediction *= pow(x[i - 1], _coefficients.b[i]);
  }
  return prediction.toDouble();
}

Future<Intervals> calculateIntervals({
  required List<List<double>> z,
  required List<double> zy,
  required List<double> zyHat,
}) async {
  final n = z.length;
  final p = z.first.length;
  final nu = n - (p + 1); // Degrees of freedom

  // Calculate t-statistic
  final tStat = await Student.inv2T(alpha: 0.05 / 2, df: nu) ?? 0;

  try {
    // Calculate residual standard deviation
    final residuals = List.generate(n, (i) => zy[i] - zyHat[i]);
    final szy =
        sqrt(residuals.map((r) => r * r).reduce((a, b) => a + b) / nu);
```

```
// Calculate means for each predictor
final zMeans = List.generate(
  p,
  (j) => z.map((row) => row[j]).reduce((a, b) => a + b) / n,
);

// Calculate centered Z matrix
final zCentered =
    List.generate(n, (i) => List.generate(p, (j) => z[i][j] - zMeans[j]));

// Calculate covariance matrix
final sZ = _algebra.matrixMultiply(
  _algebra.transposeMatrix(zCentered),
  zCentered,
);

// Invert covariance matrix
final sZInv = _algebra.invertMatrix(sZ);

// Calculate quadratic form for each observation
final quadraticForm = List.generate(n, (i) {
  final temp = _algebra.matrixVectorMultiply(sZInv, zCentered[i]);
  return List.generate(p, (j) => temp[j] * zCentered[i][j])
      .reduce((a, b) => a + b);
});

// Calculate intervals in log scale
final predLower = List.generate(
  n,
  (i) => zyHat[i] - tStat * szy * sqrt(1 + 1 / n + quadraticForm[i]),
);
final predUpper = List.generate(
  n,
  (i) => zyHat[i] + tStat * szy * sqrt(1 + 1 / n + quadraticForm[i]),
);
final confLower = List.generate(
  n,
  (i) => zyHat[i] - tStat * szy * sqrt(1 / n + quadraticForm[i]),
);
final confUpper = List.generate(
  n,
  (i) => zyHat[i] + tStat * szy * sqrt(1 / n + quadraticForm[i]),
);

// Transform back to original scale
return Intervals(
  y: zy.map((x) => pow(10, x).toDouble()).toList(),
  yHat: zyHat.map((x) => pow(10, x).toDouble()).toList(),
```

```
      predictionLower: predLower.map((x) => pow(10, x).toDouble()).toList(),
      predictionUpper: predUpper.map((x) => pow(10, x).toDouble()).toList(),
      confidenceLower: confLower.map((x) => pow(10, x).toDouble()).toList(),
      confidenceUpper: confUpper.map((x) => pow(10, x).toDouble()).toList(),
    );
  } catch (e) {
    LoggerService.instance.e('Failed to calculate intervals: $e');
    return Intervals.empty();
  }
}

Future<QualityTypes> calculateProjectQuality({
  required List<double> x,
  required double y,
  required double predictedY,
}) async {
  final factors = [
    _normalization.normalizeFactors(RegressionFactors(y: y, x: x)),
    ..._factors,
  ];
  final distances = OutliersProvider().calculateMahalanobisDistances(factors);
  var chiSquare = 0.0;
  switch (p) {
    case 1:
      chiSquare = 7.879438577;
    case 2:
      chiSquare = 10.59663473;
    case 3:
      chiSquare = 12.83815647;
  }
  for (var i = 0; i < distances.length; i++) {
    if (distances[i] > chiSquare) {
      return QualityTypes.unknown;
    }
  }

  final intervals = await calculateIntervals(
    zyHat: [_normalization.normalize(predictedY), ..._predictedValues],
    z: factors.map((f) => f.x).toList(),
    zy: factors.map((f) => f.y).toList(),
  );
  final predLower = intervals.predictionLower.first;
  final predUpper = intervals.predictionUpper.first;
  final confLower = intervals.confidenceLower.first;
  final confUpper = intervals.confidenceUpper.first;

  _log.i(
    'Calculating quality for project with X: $x, Y: $y, '
```

```
      'predicted Y: $predictedY, '
      'prediction interval: $predLower - $predUpper, '
      'confidence interval: $confLower - $confUpper',
    );

    if (y > predUpper) {
      return QualityTypes.low;
    } else if (y >= confLower && y <= confUpper) {
      return QualityTypes.medium;
    } else if ((y > confUpper && y <= predUpper) ||
        (y >= predLower && y < confLower)) {
      return QualityTypes.high;
    }

    return QualityTypes.unknown;
  }
}

class MinMax {
  MinMax(this.min, this.max);

  final double min;
  final double max;
}

class MinMaxFactors {
  MinMaxFactors({
    required this.y,
    required this.x1,
    this.x2,
    this.x3,
  });

  final MinMax y;
  final MinMax x1;
  final MinMax? x2;
  final MinMax? x3;
}
```

ДОДАТОК В – ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «FlutterStats»

Програмне забезпечення «FlutterStats» створене для автоматизації оцінювання якості застосунків, розроблених на платформі Flutter. Воно дозволяє очистити данні від викидів, будувати математичну модель для прогнозування якості програмного забезпечення.

Призначення та функціональні можливості:

1. Завантаження метрик.

Програма працює з файлами формату CSV, що містять значення метрик: RFC (Response for Class), CBO (Coupling Between Object Classes), WMC (Weighted Methods per Class). Користувач може завантажувати власні набори даних для аналізу.

2. Розподіл даних.

Завантажені дані автоматично розподіляються на:

– Навчальну вибірку для побудови математичної моделі.

– Тестову вибірку для перевірки точності прогнозів моделі.

3. Видалення викидів

Програма здійснює поетапне очищення даних від аномальних спостережень. Значення метрик нормалізуються за допомогою десяткового логарифмування. На основі нормалізованих даних розраховуються квадрат відстані Махаланобіса (2.1) та коваріаційна матриця (2.2). Для кожного спостереження обчислюється тестова статистика (2.6). Викиди визначаються шляхом порівняння значення тестової статистики із критичним значенням F-крит (2.7). Процес повторюється до повного видалення всіх викидів. Після

побудови нелінійної регресійної моделі із вибірки видаляються точки, які виходять за межі інтервалу прогнозування (2.8).

4. Побудова математичної моделі:

– Розрахунок параметрів моделі, коефіцієнтів b_0, b_1, b_2 (2.10).

– Використання нелінійної регресії для прогнозування метрики RFC на основі значень CBO та WMC.

– Розрахунок довірчих інтервалів (2.12) та інтервалів прогнозування.

5. Тестування моделі:

– Визначення показників якості побудованої моделі, таких як R^2, *MMRE*, *PRED(0,25)*.

– Оцінка якості побудованої моделі на основі тестової вибірки.

6. Оцінювання якості застосунків:

– Визначення якості проекту на основі побудованої математичної моделі.

– Класифікація якості за рівнями (висока, середня, низька) відповідно до меж інтервалів.

7. Збереження результатів:

– Можливість експорту вибірки без викидів та вибірок для побудови та тестування моделі у форматі CSV.

Мова розробки та технології:

«FlutterStats» повністю розроблено на платформі Flutter з використанням мови програмування Dart.

Flutter: забезпечує реалізацію інтерфейсу та обробку даних у рамках єдиної екосистеми, що дозволяє створювати кросплатформні рішення для мобільних пристроїв, десктопу та вебдодатків.

Dart: використовується для написання всіх алгоритмів, зокрема обробки метрик, розподілу даних на вибірки, побудови математичної моделі та тестування її точності.

Принципи роботи:

– Користувач завантажує файл із метриками у форматі CSV.

– Програма автоматично розподіляє дані на навчальну і тестову вибірки.

– На основі навчальної вибірки будується математична модель.

– Виконується тестування моделі на тестовій вибірці.

– Користувач може зберегти вибірки у файл CSV.

– Користувач може отримати якість застосунку (висока, середня, низька) використовуючи значення метрик RFC, CBO, WMC.

Технічні характеристики:

Системні вимоги:

– Операційна система: Windows.

Мінімальні вимоги:

– Процесор із частотою 1 ГГц або вище.

– Оперативна пам’ять: 512 МБ і більше.

– Дисковий простір: 100 МБ.

Функціональні особливості:

– Підтримка роботи з великими наборами даних.

– Швидке виконання обчислень завдяки алгоритмам у Dart.

– Зручний інтерфейс для завантаження й експорту даних.

Переваги використання

– Автономність: Уся обробка даних виконується в межах програми.

– Гнучкість: Можливість завантаження власних вибірок для аналізу.

– Достовірність: Забезпечення коректної побудови моделі та тестування її на нових даних.

– Зручність: Простий у використанні інтерфейс, що підходить навіть для користувачів із базовими навичками.

FlutterStats є ефективним інструментом для розробників, який спрощує процес оцінювання якості застосунків, розроблених на Flutter.

ДОДАТОК Г – ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «FlutterStats»

1. Завантаження та встановлення

– Завантажте виконуваний файл FlutterStats.

– Розпакуйте архів та запустіть файл FlutterStats.exe

2. Завантаження даних

– Натисніть кнопку «Load Data File» або Кнопку «+» на головній сторінці програми.

– Виберіть файл у форматі CSV з даними про метрики проектів.

– Дані будуть автоматично завантажені та оброблені.

На рисунку Г.1 представлено завантаження даних.

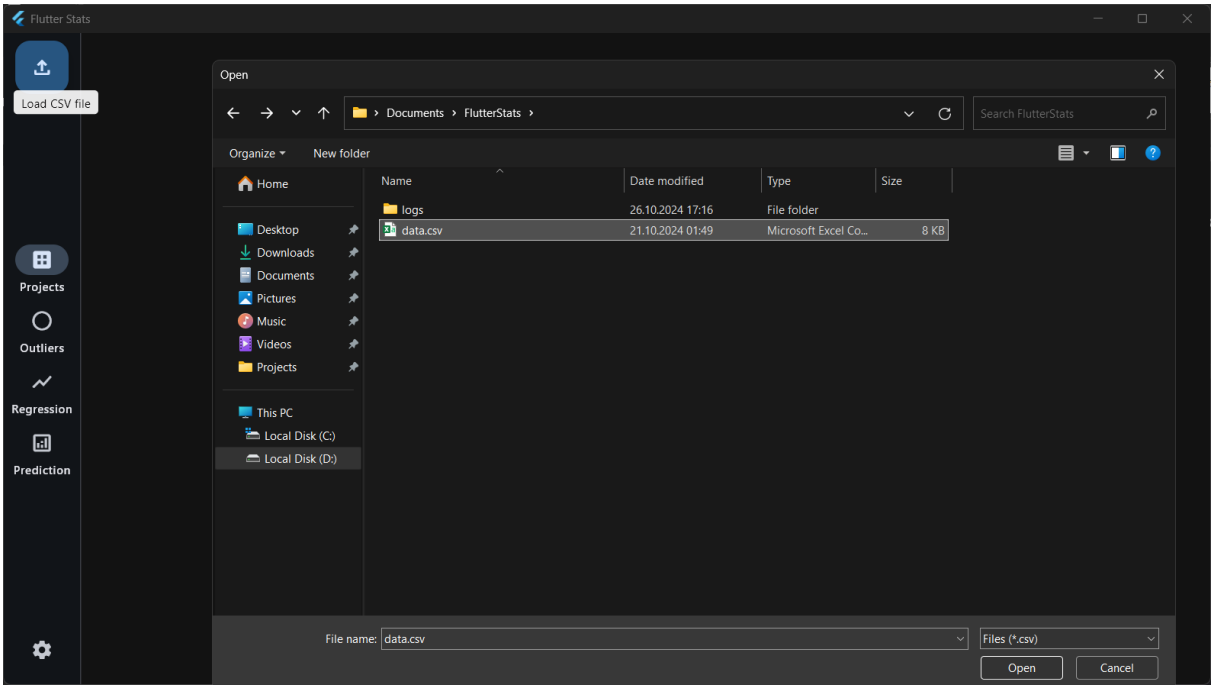


Рисунок Г.1 – Завантаження даних у «FlutterStats»

3. Перегляд та аналіз даних

– Використовуйте бічне меню для переходу між різними вкладками з даними.

– Вкладка «Projects» – перегляд завантажених проектів.

– Вкладка «Outliers» – перегляд ітерацій виявлених викидів, які були вилучені з набору даних.

– Вкладка «Regression» – перегляд побудованої регресійної моделі, її коефіцієнтів та показників якості нелінійної моделі.

– Вкладка «Prediction» – прогнозування якості за допомогою метрик RFC, CBO та WMC.

4. Використання побудованої моделі

– Перейдіть до вкладки «Prediction».

– Введіть бажану метрику RFC в поле «Enter Y (CBO)».

– Введіть бажану метрику CBO в поле «Enter X1 (CBO)».

– Введіть бажану метрику WMC в поле «Enter X2 (WMC)».

– Змінюйте значення та переглядайте якість.

На рисунку Г.2 представлено використання побудованої моделі.

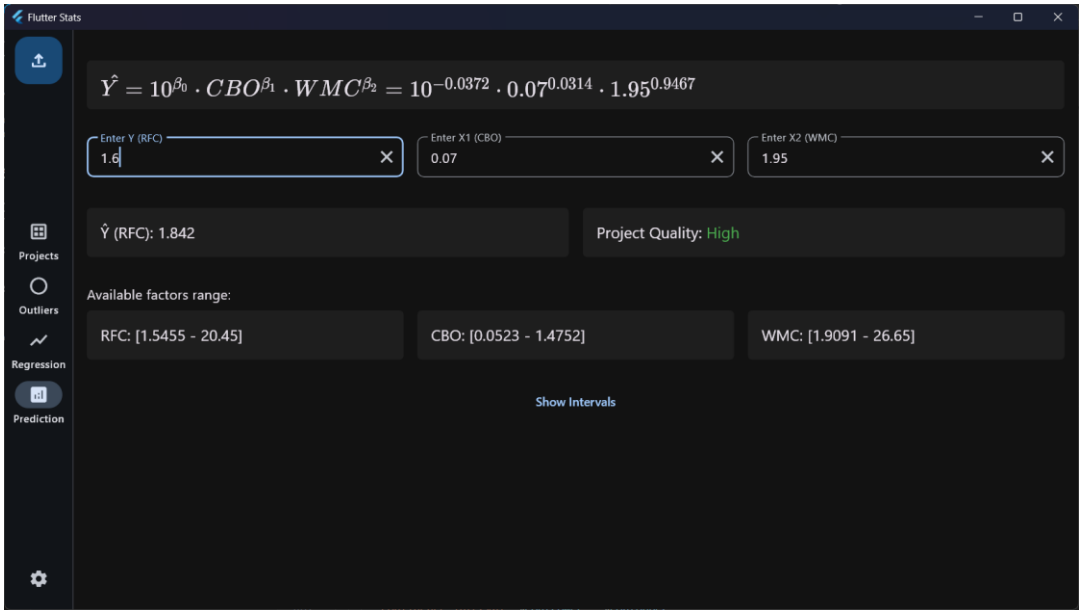


Рисунок Г.2 – Використання побудованої моделі у «FlutterStats»

ДОДАТОК Д – ПРОГРАМА І МЕТОДИКА ВИПРОБУВАНЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «FlutterStats»

1. Об’єкт випробування

Об'єктом випробування є програмне забезпечення «FlutterStats» для оцінювання якості застосунків, що створюються на платформі Flutter. Програмне забезпечення використовує математичну модель, яка базується на метриках CBO, WMC та RFC для прогнозування та аналізу якості програмного коду.

2. Мета випробування

Метою випробування програмного забезпечення «FlutterStats» є перевірка коректності функціонування, достовірності математичної моделі та відповідності заявленим вимогам до продуктивності й надійності. Конкретні цілі випробування включають:

– Оцінку функціональної відповідності програмного забезпечення заявленим вимогам.

– Перевірку коректності розрахунків.

– Тестування продуктивності та стійкості до навантажень.

– Виявлення та аналіз можливих дефектів і недоліків у роботі.

3. Вимоги до програмного забезпечення

При проведенні випробувань програмного забезпечення «FlutterStats» функціональні характеристики підлягають перевірці на відповідальність вимогам, викладеним у пункті «Вимоги до функціональних характеристик» технічного завдання, розміщеного у Додатку А.

4. Вимоги до програмної документації

Програмна документація повинна включати в себе наступні документи:

1) Текст програми.

2) Опис програми.

3) Програма та методика випробувань.

4) Інструкція користувача.

5. Засоби і порядок випробувань

5.1 Технічні засоби, що використовуються під час випробувань

– CPU: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz.

– RAM: 16 GB 2400MHz.

– SSD: Samsung SSD 970 EVO Plus 500GB.

– Ethernet: TP-LINK Gigabit Ethernet USB Adapter.

5.2 Програмні засоби, що використовуються під час випробувань

– Windows 11 Home x64 23H2 22631.4460.

– Visual Studio Code 1.95.3.0.

– Visual Studio Community 2022 17.11.5.

– Flutter 3.24.5.

– Dart 3.5.4.

5.3 Порядок проведення випробувань

Випробування програми «FlutterStats» складається з двох етапів: перевірки вимог до програмної документації та перевірки вимог до функціональних характеристик програмного забезпечення.

6 Методи випробувань

6.1 Методика проведення перевірки вимог до програмної документації

Під час перевірки вимог до програмної документації треба перевірити:

– наявність усіх необхідних документів і їх відповідність вимогам до змісту й формату;

– відповідність документів вимогам до змісту, зокрема перевірити наявність необхідних даних і опису програми;

– відповідність програми та методики випробувань вимогам до тестування;

– чи містить інструкція користувача зрозумілі кроки для використання програмного забезпечення;

– структуру й оформлення документів, включаючи заголовки, нумерацію сторінок, зміст, читабельність тексту, а також його відповідність граматичним і орфографічним правилам;

– чи відповідають документи стандартам документації та чи мають зрозумілу структуру.

6.2 Методика проведення перевірки вимог до функціональних характеристик програмного продукту

Під час тестування програмного забезпечення «FlutterStats» для оцінювання якості застосунків, що створюються на платформі Flutter, потрібно виконати наступні тести для перевірки вимог до функціональних характеристик:

– Завантаження метрик із файлу CSV: коректність завантаження даних (RFC, CBO, WMC).

– Розподіл даних на навчальну та тестову вибірки: правильність автоматичного розподілу.

– Видалення викидів з вибірки: ідентифікацію та видалення аномальних значень.

– Побудова математичної моделі: правильність розрахунку параметрів моделі (b_0, b_1, b_2) та прогнозування метрики RFC.

– Розрахунок довірчих і прогнозних інтервалів: правильність обчислення інтервалів.

– Тестування моделі на тестовій вибірці: якість моделі за показниками R^2, MMRE, PRED(0.25).

– Оцінювання якості застосунків: коректність класифікації якості проектів.

– Експорт результатів у форматі CSV: можливість експорту даних у правильній структурі.

– Обробка некоректних даних: стабільність роботи при завантаженні некоректних або порожніх файлів.

У таблиці Д.1 представлено перелік випробувань.

Таблиця Д.1 – Перелік випробувань основних функціональних можливостей

№	Дія	Очікуваний результат
1	Завантаження метрик із файлу CSV	Дані метрик (RFC, CBO, WMC) успішно завантажено без помилок
2	Автоматичний розподіл завантажених даних на навчальну та тестову вибірки	Дані розподілено відповідно до обраних параметрів
3	Видалення викидів з вибірки	Аномальні значення ідентифіковано та видалено, дані нормалізовано
4	Побудова математичної моделі на основі навчальної вибірки	Модель побудовано, параметри (b_0, b_1, b_2) розраховано
5	Прогнозування значення RFC на основі метрик CBO та WMC	Прогноз відповідає розрахунковим значенням
6	Розрахунок довірчих і прогнозних інтервалів	Інтервали обчислено коректно, межі визначено
7	Тестування моделі на тестовій вибірці	Показники R^2, MMRE, PRED(0.25) відповідають очікуваним значенням
8	Оцінювання якості Flutter-застосунку	Програма класифікує якість проекту (висока, середня, низька)
9	Експорт результатів у форматі CSV	Дані збережено у вказаному форматі, структура файлу відповідає вимогам
10	Обробка некоректних даних або порожнього файлу	Програма видає відповідне попередження або повідомлення про помилку

В результаті проведення випробувань «FlutterStats» для оцінювання якості застосунків, що створюються на платформі Flutter були перевірені його функціональні характеристики.