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 розміру програмних застосунків, що використовують 2D графічні рушії, та розробка програми для її реалізації.

Виконав: студент групи 6151м
_____ Камінський С.С.
(підпис, ПІБ)

Керівник роботи:
доцент кафедри ПЗАС, к.т.н, доцент.
(посада, науковий ступень вчене звання)
_____ Макарова Л.М.
(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 Камінському Станіславу Станіславовичу _____
(Прізвище, ім'я, по батькові)

1. Тема роботи: Математична модель для оцінювання розміру програмних застосунків, що використовують 2D графічні рушії, та розробка програми для її реалізації._____

Керівник роботи доцент кафедри ПЗАС, к.т.н, доцент Макарова Лідія Миколаївна ____

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 дослідження, Завдання дослідження, Об'єкт дослідження, Предмет дослідження, Наукова новизна одержаних результатів, Практичне значення одержаних результатів, Особистий внесок здобувача, Апробація результатів досліджень, Публікації, Аналіз існуючих моделей оцінювання розміру ПЗ, Обґрунтування необхідності дослідження, Математична модель для оцінювання розміру програмних застосунків, що використовують 2D графічні рушії, Аналіз вимог до програмного забезпечення, Архітектура програмного забезпечення, Діаграма варіантів використання, Технічний проєкт 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 _____ Камінський С.С.
(підпис) (ПІБ)

Керівник роботи _____ Макарова Л.М.
(підпис) (ПІБ)

РЕФЕРАТ

Камінський Станіслав Станіславович

«Математична модель для оцінювання розміру програмних застосунків, що використовують 2D графічні рушії, та розробка програми для її реалізації»

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

Обсяг роботи: 118 стор., 3 табл., 16 рис., 62 використане джерело, 5 додатків.

Актуальність теми: Дослідження зосереджено на вирішенні актуальної задачі оцінювання розміру програмного забезпечення для застосунків, що базуються на 2D графічних рушіях. В умовах зростання популярності таких застосунків, ця робота сприяє розвитку галузі, надаючи адаптовані до специфічних характеристик моделі.

Мета та завдання дослідження: Підвищення достовірності оцінювання розміру застосунків з 2D графічними рушіями шляхом удосконалення математичної моделі та розробки ПЗ для її реалізації. Завдання включають вивчення існуючих підходів, визначення специфічних метрик, розробку моделі, створення ПЗ та його тестування для перевірки достовірності.

Об'єкт дослідження: Процес оцінювання розміру програмних застосунків з використанням 2D графічних рушіїв.

Предмет дослідження: Математична модель для оцінювання розміру застосунків з 2D графічними рушіями, яка враховує метрики: кількість класів, кількість методів на клас, глибину дерева наслідування.

Методи дослідження: У роботі застосовано методи регресійного аналізу, математичної статистики, теорії ймовірностей, об'єктно-орієнтованого програмування.

Наукова новизна: Удосконалено математичну модель для оцінювання розміру програмних застосунків, що використовують 2D графічні рушії за рахунок додавання нових параметрів моделі та застосування нормалізуючого перетворення у вигляді десяткового логарифма із врахуванням інтервалу значень метрик застосунків з 2D графічними рушіями, що забезпечує більш достовірне прогнозування метрики KLOC на основі ключових метрик NM (кількість методів), NC (кількість класів) і DIT (глибина дерева наслідування), застосунків з 2D графічними рушіями, на відміну від існуючих моделей для Java, C# та PHP застосунків.

Практичне значення одержаних результатів: Практичне значення полягає у розробці програми для визначення метрик застосунків, що використовують 2D графічні рушії та побудови нелінійної регресійної моделі для оцінювання їх розміру. Результати роботи можуть бути використані розробниками програмного забезпечення для більш точного планування проектів з розробки застосунків, що використовують 2D графічні рушії та оцінювання необхідних ресурсів.

Апробація результатів дослідження: Основні положення і результати досліджень пройшли апробацію на XV Міжнародної науково-технічної конференції «Інновації в суднобудуванні та океанотехніці» (26-27 вересня 2024р., м. Миколаїв) та V Всеукраїнській науково-практичній інтернет-конференції «Інформаційні технології: моделі, алгоритми, системи (ITMAS-2024)» (30-31 жовтня 2024, м. Миколаїв).

Публікації: За результатами досліджень опубліковано дві наукові праці, а саме матеріали конференції.

Ключові слова: оцінювання розміру програмного забезпечення, 2D графічні рушії, нелінійна регресійна модель, метрики програмного забезпечення, планування ресурсів розробки.

ABSTRACT

Kaminskyi Stanislav Stanislavovich

«A mathematical model for estimating the size of software applications using 2D graphics engines and development of a program for its implementation»

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

Volume of work: 118 pages, 3 tables, 16 figures, 62 references, 5 appendices.

Relevance of the topic of the work: The study focuses on solving the urgent problem of sizing software for applications based on 2D graphics engines. Given the growing popularity of such applications, this work contributes to the development of the industry by providing models adapted to specific characteristics.

Purpose and objectives of the study: Improving the reliability of sizing applications with 2D graphics engines by improving the mathematical model and developing software for its implementation. The tasks include studying existing approaches, defining specific metrics, developing the model, creating software, and testing it to verify its reliability.

The object of the study: The process of sizing software applications using 2D graphics engines.

Subject of the study: A mathematical model for estimating the size of applications with 2D graphics engines that takes into account the following metrics: number of classes, number of methods per class, depth of the inheritance tree.

Methods of the study: The methods of regression analysis, mathematical statistics, probability theory, and object-oriented programming were used in the study.

Scientific novelty of the results obtained: A mathematical model for estimating the size of software applications using 2D graphics engines has been improved by adding new model parameters and applying a normalizing transformation in the form

of a decimal logarithm, taking into account the range of values of metrics for applications with 2D graphics engines, which provides more reliable prediction of the KLOC metric based on the key metrics NM (number of methods), NC (number of classes) and DIT (depth of inheritance tree) of applications with 2D graphic engines, in contrast to existing models for Java, C# and PHP applications.

Practical significance of the obtained results: The practical significance of this work is to develop a program for determining the metrics of applications using 2D graphics engines and building a nonlinear regression model to estimate their size. The results of the work can be used by software developers to more accurately plan projects for the development of applications using 2D graphics engines and estimate the required resources.

Testing of the research results: The main provisions and results of the research were tested at the XV International Scientific and Technical Conference “Innovations in Shipbuilding and Ocean Engineering” (September 26-27, 2024, Mykolaiv) and the V All-Ukrainian Scientific and Practical Internet Conference “Information Technologies: Models, Algorithms, Systems (ITMAS-2024)” (October 30-31, 2024, Mykolaiv).

Publications: Based on the results of the research, two scientific papers were published, namely the conference proceedings.

Keywords: software size estimation, 2D graphics engines, nonlinear regression model, software metrics, development resource planning.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – Програмне забезпечення.

NC – Number of Classes;

NM – Number of Methods Per Class;

DIT – Depth of Inheritance Tree;

KLOC – Thousands of Lines of Code;

MMRE – Mean Magnitude of Relative Error;

Pred – Percentage of Prediction

UML – Unified Modeling Language

ВСТУП

Зі зростанням використання 2D графічних рушіїв у розробці ПЗ, оцінювання розміру таких програмних застосунків стає важливою частиною процесу управління розробкою. Особливості архітектури програм, що використовують 2D графічні рушії, створюють додаткові складнощі в оцінюванні їхнього розміру в порівнянні зі стандартними моделями [1].

Традиційні метрики, такі як, кількість тисяч рядків коду (KLOC), кількість класів (NC), кількість методів на клас (NM) та глибина дерева наслідування (DIT), є ключовими показниками для оцінювання розміру ПЗ [2]. Однак стандартні моделі, розроблені для інших платформ та мов програмування, не завжди можуть ефективно враховувати специфіку 2D графічних рушіїв, що може призводити до неточностей у прогнозуванні.

Особливо актуальним це питання стає при розробці проектів з використанням 2D графічних рушіїв, де традиційні моделі оцінювання, такі як COCOMO II [3], показують свою обмеженість. Специфіка таких проектів включає в себе складні системи рендерингу, управління ресурсами та оптимізацію пам'яті, системи обробки користувацького вводу та компоненти фізичного моделювання.

Існує ряд робіт, у яких розроблялися нелінійні регресійні моделі для оцінювання розміру програмного забезпечення. Наприклад, модель для застосунків на Java враховувала кількість класів і базувалася на десятковому логарифмічному перетворенні, що дозволило змоделювати залежність розміру коду від базової структури класів[4]. У іншій роботі для оцінювання застосунків на C# використовували кількість класів та методів на клас, що забезпечувало більш комплексний аналіз взаємозв’язків між структурними метриками[5]. Модель для фреймворку Symfony використовувала три ключові показники:

кількість класів, середню кількість методів на клас та глибину дерева наслідування, що дозволяло враховувати як внутрішні характеристики класів, так і їх ієрархічні зв’язки [6]. Нелінійна регресійна модель для оцінювання розміру програмного забезпечення інформаційних систем з кодом на PHP базувалася на кількох факторах, включаючи кількість класів, середню кількість методів та глибину наслідування, що дозволяло враховувати складніші аспекти проектування та структури коду[7]. На жаль, ці моделі без удосконалення не підходять до нашої специфіки через додаткові фактори складності.

Додаткові фактори складності у проектах з 2D графікою включають необхідність оптимізації продуктивності, забезпечення крос-платформної сумісності, інтеграцію з різними форматами графічних ресурсів та ефективне управління станом і анімаціями. Ці аспекти суттєво впливають на загальну складність розробки та потребують спеціального підходу до оцінювання.

Технічні виклики таких проектів охоплюють синхронізацію графічних та логічних компонентів, масштабованість графічної системи, обробку колізій та взаємодій об'єктів, а також оптимізацію використання GPU. Кожен з цих аспектів додає свій рівень складності до загальної картини проекту.

Організаційні аспекти розробки включають необхідність залучення спеціалістів з специфічними знаннями та навичками, підвищені вимоги до тестування та якості, більш складний процес відладки та комплексне управління версіями ресурсів. Ці фактори суттєво впливають на планування проекту та розподіл ресурсів.

Виникає потреба у розробці нових або адаптації існуючих моделей оцінювання, які б враховували специфіку графічних компонентів, складність візуальних ефектів, вимоги до продуктивності, особливості цільових платформ та інтеграцію з іншими системами. Це дозволить більш точно прогнозувати терміни розробки та планувати необхідні ресурси.

Достовірне оцінювання розміру таких проектів є критичним для успішного планування та реалізації, оскільки недооцінка може призвести до серйозних проблем у процесі розробки та перевитрат ресурсів [8]. Тому важливо розробити та впровадити нову модель, яка буде враховувати усі особливості проектів з 2D графічними рушіями.

Одними з ключових чинників, що впливають на оцінювання розміру ПЗ, є такі метрики, як кількість рядків коду, кількість класів, кількість методів на клас і глибина дерева наслідування. Класи є базовими одиницями програмного коду, що відповідають за конкретну функціональність. Кількість методів на клас визначає функціональне навантаження кожного класу, що впливає на складність його розробки та подальшої підтримки. В свою чергу, глибина дерева наслідування відображає ієрархічну складність архітектури, що також є важливим фактором при оцінюванні розміру проекту. Комплексне врахування цих ключових метрик дозволяє точніше оцінювати масштаб програмного проекту та спрогнозувати необхідні ресурси для його успішної реалізації.

Виходячи з цього, розробка ефективних моделей та інструментів для оцінювання розміру ПЗ, що використовує 2D графічні рушії, є актуальним і важливим завданням. Це сприятиме підвищенню достовірності прогнозування, покращенню управління проектами та забезпеченню ефективного використання ресурсів. Такий підхід відповідає як національним, так і міжнародним тенденціям розвитку IT-сфери, де пріоритетом є створення якісних програмних продуктів з оптимальним використанням часу і ресурсів, що є важливим для України у світовій IT-конкуренції.

Мета дослідження полягає в підвищенні достовірності оцінювання розміру програмних застосунків, що використовують 2D графічні рушії, за рахунок удосконалення математичної моделі для оцінювання розміру та розробці ПЗ для її реалізації, що забезпечить більш високу достовірність прогнозування розміру на ранніх етапах розробки.

Для досягнення цієї мети необхідно вирішити такі завдання:

– Дослідити наявні методи оцінювання розміру ПЗ;

– Визначити ключові метрики для оцінювання розміру застосунків, що створюються за допомогою 2D графічних рушіїв;

– Розробити математичну модель для оцінювання розміру програмних проектів;

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

– Провести тестування розробленого ПЗ з метою забезпечення достовірності прогнозування.

Об'єктом дослідження є процес оцінювання розміру програмних застосунків, розроблених з використанням 2D графічних рушіїв.

Предметом дослідження є математична модель для оцінювання розміру програмних застосунків, створених з використанням 2D графічних рушіїв, яка враховує такі метрики, як кількість класів, кількість методів на клас, глибина дерева наслідування для прогнозування показників розміру проекту.

Методи дослідження У дослідженні використано методи регресійного аналізу, математичної статистики, теорії ймовірностей, об'єктно орієнтованого програмування.

Наукова новизна. Удосконалено математичну модель для оцінювання розміру програмних застосунків, що використовують 2D графічні рушії за рахунок додавання нових параметрів моделі та застосування нормалізуючого перетворення у вигляді десяткового логарифма із врахуванням інтервалу значень метрик застосунків з 2D графічними рушіями, що забезпечує більш достовірне прогнозування метрики KLOC на основі ключових метрик NM (кількість методів), NC (кількість класів) і DIT (глибина дерева наслідування), застосунків з 2D графічними рушіями, на відміну від існуючих моделей для Java, C# та PHP застосунків.

Практичне значення одержаних результатів полягає у тому, що розроблено програмний засіб для визначення метрик застосунків, що використовують 2D графічні рушії та побудови нелінійної регресійної моделі для оцінювання їх розміру. Результати роботи можуть бути використані розробниками програмного забезпечення для більш точного планування проектів з розробки застосунків, що використовують 2D графічні рушії та оцінювання необхідних ресурсів.

Особистий внесок здобувача. Всі отримані результати є результатом самостійної роботи автора. У роботах [9, 10], які були опубліковані у співавторстві, автором виконано: обробка метрик програм, що використовують 2D графічні рушії; побудова нелінійної регресійної моделі та перевірка її якості.

Апробація результатів досліджень. Основні положення і результати досліджень пройшли апробацію на XV Міжнародної науково-технічної конференції «Інновації в суднобудуванні та океанотехніці» (26-27 вересня 2024р., м. Миколаїв) та V Всеукраїнській науково-практичній інтернет-конференції «Інформаційні технології: моделі, алгоритми, системи (ITMAS-2024)» (30-31 жовтня 2024, м. Миколаїв).

Публікації. За результатами досліджень опубліковано дві наукові праці, а саме матеріали конференції.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА МОДЕЛЕЙ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Огляд основних метрик які використовуються для оцінювання розміру програмного забезпечення

Метрики ПЗ є важливим інструментом у процесі розробки та оцінювання розміру програмних продуктів [11]. Вони надають кількісні дані щодо різних аспектів ПЗ, що дає змогу об'єктивно оцінювати його розмір. У сфері об'єктно-орієнтованого програмування особливо значущими є метрики, що оцінюють структурні характеристики коду, такі як зв'язність, зчеплення та складність класів. До найбільш поширених метрик об'єктно-орієнтованого ПЗ відносяться:

1. NC (Number of Classes) – кількість класів. NC вказує на загальну кількість класів у програмному забезпеченні. Чим більше класів, тим складніша архітектура програми, оскільки це може призвести до численних залежностей між класами [12]. Високе значення NC може мати кілька наслідків:

– Збільшення кількості класів підвищує складність управління кодом через необхідність ретельного контролю їх взаємодії;

– Збільшення кількості класів підвищує ймовірність неправильного реалізування залежностей між ними, що може спричинити помилки;

– Велика кількість класів вимагає чіткої стратегії управління їхніми залежностями для запобігання зайвій складності та помилкам при зміні коду.

2. NM (Number of Methods Per Class) – кількість методів на клас. NM вимірює середню кількість методів у кожному класі. Більше методів в класі може означати більшу складність, оскільки з кожним методом зростає кількість точок входу, що потребують тестування і підтримки [13]. Високе значення NM може свідчити про такі аспекти:

– Клас з великою кількістю методів стає важким для розуміння, оскільки кожен метод додає новий рівень функціональності, який потрібно враховувати;

– Багато методів ускладнюють тестування, оскільки для кожного з них потрібно створювати окремі тести, що значно збільшує час перевірки програми;

– Клас з великою кількістю методів може стати занадто великим і складним для підтримки, що свідчить про необхідність розділення його на менші компоненти для зниження складності та покращення читабельності коду.

3. DIT (Depth of Inheritance Tree) – глибина дерева наслідування. DIT показує, скільки рівнів має ієрархія наслідування класу. Високе значення DIT означає, що клас глибоко наслідуються від інших класів [14]. Це може призвести до кількох проблем:

– Клас з великою глибиною наслідування може бути складним для розуміння, оскільки потрібно враховувати всі батьківські класи та їхні особливості, що знижує прозорість коду;

– Зміни в базовому класі можуть мати непередбачувані наслідки для похідних класів, ускладнюючи модифікацію коду та підвищуючи ймовірність помилок;

– Глибока ієрархія може обмежувати можливості для розширення та модифікації коду, вимагаючи змін у всіх класах ієрархії.

4. KLOC (Thousands of Lines of Code) – тисячі рядків коду. KLOC вимірює розмір програми за кількістю рядків коду [15]. Ця метрика допомагає оцінювати обсяг програми та може мати кілька наслідків:

– Великий обсяг коду свідчить про складність програми, оскільки збільшується ймовірність наявності багатьох взаємодій, залежностей і потенційних точок відмов;

– Чим більше рядків коду, тим більше часу та зусиль потрібно для тестування програми, а великі програми вимагають комплексного підходу до тестування для забезпечення якості;

– Великий обсяг коду може вказувати на необхідність реорганізації чи спрощення програми для підвищення її підтримуваності та зрозумілості, а рефакторинг допомагає зменшити складність і полегшити розробку та тестування.

Ці метрики часто використовуються для оцінювання розміру ПЗ, розробленого за допомогою різних мов програмування. Однак варто пам'ятати, що їх інтерпретація може змінюватися в залежності від специфіки платформи, архітектурних особливостей і прийнятих практик програмування.

1.2 Особливості розробки застосунків, що використовують 2D графічні рушії

Архітектура застосунків, що використовують 2D графічні рушії, зазвичай має модульну структуру, де кожен компонент, наприклад, рендеринг, обробка вводу, фізика чи анімація, реалізується окремо. Часто застосовують патерни проектування, такі як MVC (Model-View-Controller), що дозволяє чітко розділити логіку програми від її візуальної частини [16]. Це дає змогу легше підтримувати і змінювати програму, оскільки зміни в одному компоненті (наприклад, у відображенні графіки) не впливають на інші частини програми. Застосунки з 2D графікою також часто спираються на потоки подій, що відповідають за взаємодію користувача з інтерфейсом, забезпечуючи інтерактивність і швидку реакцію на дії користувача.

Основною особливістю розробки застосунків, що використовують 2D графічні рушії, є високий рівень абстракції для створення графічних елементів. Мова програмування Java надає потужні інструменти для побудови візуальних компонентів, такі, як бібліотеки AWT (Abstract Window Toolkit) [17] та Swing [18] для графічних інтерфейсів користувача, а також Java 2D API [19] для малювання

та маніпуляцій з графікою. Це дозволяє розробникам створювати складні графічні застосунки без необхідності глибокого занурення у низькорівневі деталі графічних процесорів. Такий підхід забезпечує спрощену інтеграцію графічних елементів у програми, що знижує складність розробки, проте вимагає чіткої організації коду для ефективного управління графічними елементами.

Важливою особливістю є ефективна обробка вхідних подій, таких як натискання клавіш або рух миші, що є необхідною частиною інтерактивних 2D графічних застосунків. Мова Java пропонує потужні механізми для обробки подій, що дозволяють швидко реагувати на дії користувача. Це дозволяє створювати програми з інтерактивними елементами, такими як кнопки, меню або анімації, де кожна дія викликає відповідну реакцію програми. Взаємодія з користувачем здійснюється через обробники подій, що визначають, як програма повинна реагувати на кожну конкретну дію. Завдяки цьому застосунки з 2D графікою можуть бути дуже зручними та адаптивними для кінцевого користувача.

Специфіка роботи застосунків, що використовують 2D графічні рушії, полягає в необхідності обробки координат, трансформацій (таких як масштабування, переміщення, обертання) та маніпуляцій з графічними об'єктами. Зазвичай для створення ефектів і анімацій використовуються матриці для трансформацій, а також спеціальні алгоритми для рендерингу, що дозволяють здійснювати точні й ефективні зміни в зображенні. Особливу увагу потрібно приділяти оптимізації алгоритмів, що забезпечують максимальну ефективність роботи застосунків, особливо при великій кількості одночасно відображених елементів.

Тестування застосунків, що використовують 2D графічні рушії, є важливим етапом розробки, оскільки необхідно не тільки перевірити коректність логіки програми, але й забезпечити точність відображення графіки на екрані. Особливу увагу слід приділяти перевірці інтерфейсу користувача на різних розмірах екрану

і з різними параметрами відображення. Також необхідно перевіряти, як програма реагує на різні події користувача, а також її стабільність при взаємодії з різними компонентами. Для тестування таких застосунків використовують не лише традиційні методи юніт-тестування, але й інструменти для автоматизованого тестування графічних елементів і взаємодії з ними.

Мова Java залишається однією з найпопулярніших мов програмування завдяки своїй універсальності, стабільності та здатності працювати на різних платформах завдяки принципу "Write Once, Run Anywhere". Вона активно підтримує об'єктно-орієнтоване програмування та новітні можливості, такі як лямбда-вирази і модульна система, що значно покращує продуктивність і зручність використання. Завдяки потужним фреймворкам і бібліотекам, Java є основою для розробки корпоративних, розподілених і хмарних систем. Це робить її однією з найбільш надійних мов для великих, критичних та довготривалих проектів [20].

Продуктивність є важливим аспектом при розробці 2D графічних застосунків на Java. Оскільки Java не підтримує апаратне прискорення, розробникам доводиться активно оптимізувати програму для забезпечення максимальної швидкості рендерингу. Це включає використання таких технік, як "double buffering" (подвійне буферизування), яке зменшує мерехтіння екрана і покращує плавність анімацій. Крім того, важливо оптимізувати алгоритми рендерингу для зменшення навантаження на процесор, наприклад, зменшувати кількість малюнків, що відображаються одночасно, або використовувати простіші графічні елементи.

Всі ці особливості разом визначають ефективність розробки та використання 2D графічних рушіїв у Java. Архітектурні підходи, ефективна обробка подій, правильна оптимізація і тестування є основою для створення потужних, інтерактивних і продуктивних графічних застосунків. Врахування всіх цих аспектів дозволяє досягти високої якості продукту, що може працювати на

різних платформах, при цьому забезпечуючи хорошу продуктивність і зручний інтерфейс для користувача.

1.3 Аналіз існуючих моделей оцінювання розміру ПЗ та обґрунтування необхідності дослідження

Оцінювання розміру ПЗ є важливим етапом у процесі розробки, оскільки дозволяє визначити обсяг роботи, необхідні ресурси та час для реалізації проекту. Існуючі моделі оцінювання розміру ПЗ можна поділити на кілька категорій: лінійні регресійні моделі, нелінійні регресійні моделі, моделі на основі машинного навчання та експертні системи. Кожна з моделей має свої переваги та обмеження, що впливають на точність та ефективність оцінювання розміру програмного продукту.

Лінійні регресійні моделі, хоча і є простими для використання та інтерпретації, часто не можуть точно описати складні взаємозв'язки між метриками ПЗ. Дослідження показують, що зв'язки між показниками розміру програмного продукту, такими як кількість рядків коду та кількість класів і методів, мають нелінійний характер. Існують моделі для оцінювання розміру ПЗ, написаного на Java, PHP та C# які ґрунтуються на припущеннях структури і організації коду, що потребує адаптації методів оцінювання розміру ПЗ під конкретні умови розробки. Розглянемо більш детально деякі з цих моделей.

Модель для веб-додатків, реалізованих мовою Java, була спрямована на оцінювання розміру програмного забезпечення, реалізована через кількість класів в якості незалежної змінної X. В роботі представлено декілька моделей, одна з яких модель з використанням одновимірного нормалізуючого перетворення Джонсона, інша модель - на основі десяткового логарифмічного перетворення, які дозволяють наблизити розподіл метрик до гаусівського.

Нелінійна регресійна модель на основі десяткового логарифмічного перетворення має вигляд [4]:

$$Y = 10^{\varepsilon - 1{,}469} X^{1{,}2266}, \quad (1.1)$$

де: X – загальна кількість класів,

Y – число рядків коду веб-додатку в KLOC.

Дослідження включало збирання даних з 33 Java-додатків, видалення викидів та використання кількості класів як ключового параметра. Параметри моделі розраховано з використанням методу максимальної правдоподібності. Оцінка якості моделі проводилась за трьома показниками: коефіцієнтом детермінації R^2 = 0,6147, середньою величиною відносної похибки *MMRE* = 0,3477, та рівнем прогнозування *PRED(0,25)* = 0,5000.

Модель має перевагу у зменшенні ширини інтервалу передбачення порівняно з іншими регресійними моделями. Однак показники *MMRE* та *PRED(0,25)* не досягли прийнятних рівнів (не більше 0,25 та не менше 0,75 відповідно), що свідчить про необхідність застосування багатовимірного перетворення Джонсона для врахування взаємного впливу двох випадкових величин.

Ця модель підходить для оцінювання розміру Java-додатків на ранніх стадіях розробки та дозволяє отримувати точніші прогнози порівняно з іншими підходами, такими, як лінійна регресія.

Ще одна модель, розроблена для оцінювання розміру веб застосунків, створених мовою програмування C#, враховує два основні фактори: кількість класів X_1 та кількість методів на клас X_2, що є ключовими метриками для аналізу. Модель побудована на основі багатофакторної нелінійної регресії, використовуючи дані з 40 веб застосунків. Фінальна модель має вигляд [5]:

$$Y = 10^{\varepsilon+1,5805} X_1^{0,7562} X_2^{0,6147}, \quad (1.2)$$

де: X_1 – кількість класів проекту

X_2 – кількість методів на клас проекту

Y – розмір коду в тисячах рядків (KLOC).

Оцінка якості моделі показала високу точність: R^2 = 0,96, $MMRE$ = 0,13, $PRED(0,25)$= 0,97. У порівнянні з раніше відомими моделями для інших мов, дана модель демонструє значне підвищення достовірності оцінювання для C#-веб застосунків. Але ця модель також не враховує глибину дерева наслідування DIT, яка має суттєвий вплив на архітектуру 2D графічних рушіїв. Крім того, особливості цієї моделі краще адаптовані для задач веб розробки, що звужує її застосовність для аналізу застосунків із графічними рушіями.

Для порівняння також було розглянуто модель для веб застосунків, створених з використанням фреймворку Symfony. Ця модель враховує три фактори: кількість класів, середню кількість методів на клас і глибину дерева наслідування, що відповідає вимогам нашої задачі [6]:

$$Y = 10^{\varepsilon-1,6378} X_1^{0,9664} X_2^{0,9630} X_3^{0,4627}, \quad (1.3)$$

де Y – залежна змінна,

X_1 , X_2 , X_3 – незалежні змінні (фактори впливу),

ε – гаусівська випадкова величина (випадкова похибка).

Для обґрунтування необхідності подальшого дослідження з розробки математичної моделі для оцінювання розміру застосунків, що використовують 2D графічні рушії, було здійснено побудову моделей згідно (1.1) - (1.3) за наступними даними програмних застосунків, що використовують 2D графічні рушії.

Дані були зібрані дані зі 108 проектів з відкритим кодом, розміщених на GitHub [21]. Метрики NC (кількість класів), NM (кількість методів) та DIT (глибина дерева наслідування) на рівні проекту були отримані за допомогою спеціалізованого інструменту аналізу коду SourceMeter [22].

З кожного проекту були отримані значення метрик: NC (X_1), що відображає кількість класів у проекті, NM (X_2), який показує кількість методів у проекті, та DIT (X_3), який характеризує глибину дерева наслідування класів. Залежна змінна Y визначає розмір програмного застосунку у тисячах рядків коду (KLOC) і оцінюється на основі значень метрик X_1, X_2 та X_3. Фрагмент даних наведений у таблиці 1.1.

Таблиця 1.1 – Фрагмент даних для оцінювання розміру застосунків, що використовують 2D графічні рушії

№	URL	KLOC	NC	NM	DIT
1	2	3	4	5	6
1	https://github.com/azurite-engine/Azurite	15,57	165	2023	76
2	https://github.com/rafael-esper/JLud2D	57,512	318	3959	109
3	https://github.com/dillanspencer/Quad-Engine	2,319	24	284	2
4	https://github.com/tombefieux/2DPhysicsEngine	1,18	12	160	17
5	https://github.com/j-ackyao/KY-Engine	2,327	22	740	21
6	https://github.com/Flafla2/Remote2D-Engine	14,413	137	2032	137
7	https://github.com/fastjengine/FastJ	24,886	194	3304	185
8	https://github.com/benblaszczak/BobEngine	10,136	104	2626	186
9	https://github.com/mlabouardy/2dgamesdk-engine	0,174	6	30	2
10	https://github.com/b3dgs/lionengine	129,428	1323	16249	1544
11	https://github.com/gurkenlabs/litiengine	71,476	604	13553	486
12	https://github.com/vanZeben/2D-Game-Engine	1,376	24	200	14

Продовження таблиці 1.1

1	2	3	4	5	6
13	https://github.com/nicolasgramlich/AndEngine	54,229	636	16773	1098
14	https://github.com/jbox2d/jbox2d	52,36	349	6808	242
15	https://github.com/AlmasB/FXGL	47,328	586	7097	349
16	https://github.com/sriharshachilakapati/SilenceEngine	30,133	286	3243	153
17	https://github.com/codepath/android_simple_game_engine	0,54	11	157	12
18	https://github.com/swordmaster2k/rpgwizard	24,967	290	3272	151
19	https://github.com/kennycason/java_games	7,675	119	1843	204
20	https://github.com/SypherEngine/SypherEngine	4,755	56	616	33

Розрахунок параметрів якості для моделі (1.1) дав такі значення: *MMRE* = 0,4624, *PRED*(0,25) = 0,3796. Моделі, побудовані за (1.2) та (1.3) мають суттєво гірші показники.

Наведені результати показали, що розглянуті моделі (1.1) - (1.3) не можуть бути використані для оцінювання розміру застосунків, що базуються на 2D графічних рушіях.

Таким чином, результати проведеного аналізу вказують на відсутність моделі, яка б враховувала специфіку застосунків із використанням 2D графічних рушіїв. Жодна з розглянутих моделей не здатна забезпечити необхідну достовірність результатів. Це обґрунтовує необхідність побудови трьохфакторної нелінійної регресійної моделі, що враховуватиме особливості структури 2D графічних рушіїв, забезпечуючи достовірне оцінювання розміру програмних застосунків і ефективність управління розробкою ПЗ в цьому сегменті.

2 РОЗРОБКА МАТЕМАТИЧНОЇ МОДЕЛІ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ЗАСТОСУНКІВ, ЩО ВИКОРИСТОВУЮТЬ 2D ГРАФІЧНІ РУШІЇ

2.1 Математичний апарат для оцінювання розміру ПЗ

Перевірка нормальності розподілу даних

У процесі оцінювання розміру програмного забезпечення, зокрема при розробці моделей для прогнозування кількості рядків коду (KLOC), перевірка багатовимірної нормальності даних є важливим етапом. Багато статистичних методів, включно з нелінійною регресією, базуються на припущенні нормальності розподілу. Одним із основних підходів для перевірки багатовимірної нормальності є тест Мардіа (Mardia's Test), який оцінює асиметрію та ексцес багатовимірного розподілу через статистичні показники $\beta_{1,p}$ та $\beta_{2,p}$ [23].

Коефіцієнт асиметрії, позначений як $\beta_{1,p}$, обчислює симетрію багатовимірного розподілу, враховуючи кореляцію між змінними. Формула цього коефіцієнта:

$$\beta_{1,p} = \frac{1}{N}\sum_{i-1}^{N}\sum_{j-1}^{N}\left\{(X_i - \bar{X})^T S^{-1}(X_j - \bar{X})\right\}^3 \quad , \tag{2.1}$$

де N – кількість спостережень,

X_i – вектор і-го спостереження,

$\bar{X}$ – середній вектор,

S^{-1} – обернена коваріаційна матриця.

Про негаусівський розподіл даних свідчить значення багатовимірного ексцесу β_2 , оцінка якого визначається як [24]:

$$\hat{\beta}_2 = \frac{1}{N}\sum_{i-1}^{N}\{(X_i - \bar{X})^T S_N^{-1}(X_i - \bar{X})\}^2. \quad (2.2)$$

Якщо значення β_1 і β_2 вказують на відхилення від нормальності, може знадобитися застосування методів, що не залежать від нормальності розподілу, або проведення трансформацій даних для наближення їх до нормального вигляду.

Для перевірки нормальності розподілу вхідних даних додатково можна використати квадрат відстані Махаланобіса, який дозволяє визначити відхилення спостережень від центру розподілу з урахуванням коваріаційної структури даних. Цей метод особливо ефективний для багатовимірних даних, оскільки враховує не тільки розкид значень, але й взаємозв'язки між змінними [25]:

$$d_i^2 = (X_i - \bar{X})^2 S_N^{-1}(X_i - \bar{X}), \quad (2.3)$$

де $\bar{X}$ – вектор вибіркових середніх,

S_N – коваріаційна матриця.

Коваріаційна матриця, яка відображає взаємозв'язки між змінними та їх варіацію, розраховується за формулою [26]:

$$S_N = \frac{1}{N}\sum_{i=1}^{N} (X_i - \bar{X})(X_i - \bar{X})^T. \quad (2.4)$$

Тестова статистика для кожного спостереження дозволяє визначити, наскільки сильно конкретне спостереження відхиляється від очікуваного розподілу:

Розрахунок тестової статистики для кожного спостереження [27]:

$$T_i = \frac{D_i^2}{c_{\alpha(p,n)}}, \quad (2.5)$$

де $c_{\alpha(p,n)}$ – критичне значення розподілу Фішера з p ступенями вільності та n спостереженнями.

Розрахунок критичного значення розподілу Фішера [28]:

$$F_{\text{крит}} = F(\alpha, p, n-2), \quad (2.6)$$

де α – рівень значущості дорівнює 0,005,

p – кількість ступенів вільності (кількість факторів у моделі)

n – розмір вибірки.

Нормалізація за допомогою десяткового логарифму

Для кожного спостереження X в наборі даних застосовується десятковий логарифм:

$$z = \log_{10}(\text{x}), \quad (2.7)$$

де z – нормалізоване значення (логарифмоване значення),

x – початкове значення,

$\log_{10}(x)$ – десятковий логарифм значення x.

Цей процес переводить значення на шкалу з меншими варіаціями та допомагає зменшити вплив великих значень або викидів.

Зворотне нормалізуюче перетворення

Щоб повернути нормалізоване значення назад до початкового масштабу, потрібно скористатися оберненою операцією – піднесенням до степеня 10:

$$x = 10^z, \tag{2.8}$$

де: x – відновлене значення після нормалізації,

z – нормалізоване значення (логарифмоване значення),

10^z – обернений процес для відновлення початкового значення.

Побудова регресійної моделі

Рівняння лінійної регресії для нормалізованих даних, яке описує лінійну залежність між змінними у нормалізованому просторі:

$$\hat{z}_y = \beta_0 + \beta_1 * Z_{x_1} + \beta_2 * Z_{x_2} + \beta_3 * Z_{x_3}, \tag{2.9}$$

де: Z_{x_1} – нормалізована змінна (NC);

Z_{x_2} – нормалізована змінна (NM);

Z_{x_3} – нормалізована змінна (DIT);

$\beta_0, \beta_1, \beta_2, \beta_3$ – коефіцієнти регресії.

Нелінійна регресійна модель має вигляд:

$$\hat{z}_y = \beta_0 + \beta_1 * Z_{x_1} + \beta_2 * Z_{x_2} + \beta_3 * Z_{x_3} + \varepsilon. \tag{2.10}$$

Оцінка параметрів моделі методом найменших квадратів, що мінімізує суму квадратів відхилень:

$$\beta_i = \frac{\sum(Z_{x_i} - \overline{Z_{x_i}})(Z_y - \overline{Z_y})}{\sum(Z_{x_i} - \overline{Z_{x_i}})^2}, \quad (2.11)$$

$$\beta_0 = \overline{Z_y} - \beta_1 * \overline{Z_{x_\iota}} - \beta_2 * \overline{Z_{x_\iota}} - \beta_3 * \overline{Z_{x_\iota}}, \quad (2.12)$$

де *i* дорівнює від 1 до *n*.

Рівняння нелінійної регресії після зворотного перетворення, яке дозволяє повернутися до початкового масштабу даних:

$$\hat{Y} = 10^{\beta_0} * {X_1}^{\beta_1} * {X_2}^{\beta_2} * {X_3}^{\beta_3}. \quad (2.13)$$

Оскільки емпіричні дані, що використовуються для аналізу проєктів, зазвичай не мають нормального розподілу, часто обирають нелінійні регресійні моделі.

$$\hat{Y} = 10^{\varepsilon+\beta_0} * {X_1}^{\beta_1} * {X_2}^{\beta_2} * {X_3}^{\beta_3}. \quad (2.14)$$

Перевірка якості моделі

Для оцінки якості побудованої нелінійної регресійної моделі використовуються такі метрики: коефіцієнт детермінації R^2, $R^2_{Adjusted}$, середня абсолютна відносна похибка (*MMRE*) та *PRED*(0,25). Нижче наведені формули для кожної метрики.

Коефіцієнт детермінації (R^2)

R^2 показує, яка частина дисперсії залежної змінної пояснюється моделлю [29]. Формула:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}, \quad (2.15)$$

де: y_i – фактичне значення залежної змінної,

$\hat{y}_i$ – значення, передбачене моделлю,

$\bar{y}_i$ – середнє значення залежної змінної,

N – кількість спостережень.

$R^2 \in [0,1]$, де ближче до 1 означає кращу якість моделі.

Для кращого розуміння якості моделі із різною кількістю предикторів використовується скоригований коефіцієнт детермінації $R^2_{Adjusted}$, який враховує кількість незалежних змінних у моделі. Формула:

$$R^2_{Adjusted} = 1 - \frac{(1-R^2)*(N-1)}{N-p-1}, \quad (2.16)$$

де: R^2– стандартний коефіцієнт детермінації,

N – кількість спостережень (розмір вибірки),

p – кількість незалежних змінних у моделі.

Середня абсолютна відносна похибка (*MMRE*) оцінює середню величину відносної похибки між фактичними та передбаченими значеннями [30]. Формула:

$$MMRE = \frac{1}{N}\sum_{I=1}^{N}\frac{|y_i-\hat{y}_i|}{y_i}, \quad (2.17)$$

де всі позначення ті самі, що й вище. Чим менше значення *MMRE*, тим точніша модель (рекомендоване значення < 0,25).

PRED(0,25) визначає частку спостережень, для яких відносна похибка менша або дорівнює 25%[31]. Формула:

$$PRED(0,25) = \frac{\textit{кількість спостережень із } \frac{|y_i-\hat{y}_i|}{y_i} \leq 0.25}{N} \quad (2.18)$$

Ця метрика відображає стабільність моделі. Чим ближче значення до 1, тим краще.

2.2 Збір та попередня обробка даних метрик проектів, що використовують 2D графічні рушії

Дані для побудови нелінійної регресійної моделі, що оцінює розмір програмних застосунків з 2D графічними рушіями, були зібрані з 108 проекту на GitHub і аналізувалися за допомогою інструменту SourceMeter, як це було розглянуто в підрозділі 1.3. Метрики NC (кількість класів), NM (кількість методів) та DIT (глибина дерева наслідування) використовувалися для оцінки розміру застосунку в тисячах рядків коду (KLOC). Фрагмент даних наведено в таблиці 1.1.

Щоб визначити, яку модель слід застосовувати, необхідно перевірити вихідний набір даних на відповідність багатовимірному нормальному розподілу Гауса. Для перевірки нормальності розподілу вхідних даних насамперед слід застосувати критерії Мардіа і додатково обчислити квадрати відстаней Махаланобіса для вихідного набору даних згідно з формулами (2.1), (2.3) та (2.4).

Після перевірки даних на нормальність за допомогою критерію оцінки асиметрії β1 та ексцесу β2, було встановлено, що дані не відповідають нормальному розподілу. Результати перевірки:

β_2=179,22, що значно перевищує критичне значення 28,5828.

β_1=139,97, що також перевищує критичне значення 40,00

Це свідчить про суттєві відхилення від нормальності у розподілі даних.

Потім визначаємо квантіль розподілу Пірсона для рівня значущості 0,005. Якщо квадрат відстані Махаланобіса для проєкту перевищує цей квантіль, можна зробити висновок про відсутність нормального розподілу даних. У таблиці 1.1 наведено фрагмент набору даних, де NC відповідає змінній X_1, NM – змінній X_2,

DIT – змінній X_3, а KLOC – змінній Y. D^2 – розрахований квадрат відстані Махаланобіса для вихідних даних.

Оскільки є такі дані, для яких D^2 більший за квантіль розподілу Пірсона [32] χ^2 = 14,86, дані не мають Гаусівський розподіл [33], тому потрібно будувати нелінійну регресійну модель. При побудові нелінійної регресійної моделі потрібно використовувати метод нормалізуючих перетворень. Спочатку потрібно знайти і вилучити викиди із вихідних даних, для чого використовуємо формулу (2.2). Для нормалізації даних використовується десятковий логарифм (2.7), що допомагає зменшити розкид значень і вплив викидів.

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

2. Розраховуємо коваріаційну матрицю S для 4 змінних на нормалізованих даних.

3. Знаходимо зворотну матрицю S^{-1}.

4. Для кожного спостереження i розраховуємо квадрат відстані Махаланобіса для нормалізованих даних.

6. Для виявлення викидів використовуємо критерій χ^2. Якщо обчислене значення тестової статистики χ^2 (позначимо його як χ^2_{di}) згідно з формулою (2.5), перевищує критичне значення χ^2 згідно з формулою (2.6), для заданого рівня значущості та кількості ступенів свободи (позначимо його як $\chi^2_{\text{критичне}}$), то поточний спостережуваний набір даних слід вважати викидом.

7. Повторяємо кроки 2 – 6 поки не буде вилучено усі викиди.

Після видалення викидів залишилась вибірка із 103 застосунків. Квантіль розподілу Пірсона для 103 спостережень із рівнем значущості 0,005 та 4 ступенями вільності склав 14,86, тому можна зробити висновок, що передобробка даних завершена, оскільки $Di^2 \max = 12{,}91$.

2.3 Побудова математичної моделі для оцінювання розміру програмних застосунків, що використовують 2D графічні рушії

Після попередньої обробки даних і видалення викидів було сформовано вибірку, що включає 103 застосунків. Дані були розподілені на навчальну та тестову вибірки для побудови й оцінювання моделі. Навчальна вибірка становила 60% від загального обсягу (59 застосунків) і використовувалася для побудови моделі. Тестова вибірка, що становила 40% (44 застосунків), застосовувалася для перевірки якості моделі.

Розподіл вибірок здійснено з урахуванням збереження мінімальних і максимальних значень метрик, що забезпечило репрезентативність даних та точність оцінювання моделі.

Оскільки емпіричні дані не мають нормального розподілу, для вирішення задачі прогнозування розміру необхідно побудувати рівняння нелінійної регресії. У цьому процесі доцільно застосувати метод нормалізуючих перетворень, використовуючи десятковий логарифм для зменшення розкиду значень і підвищення достовірності моделі. Для нормалізації даних використовується десятковий логарифм згідно з формулою (2.7), що допомагає зменшити розкид значень і вплив викидів та використовуємо формулу (2.8) для повернення нормалізованого значення до початкового маштабу. Використовуючи формулу (2.9), було побудовано лінійне рівняння регресії на нормалізованих даних. Остаточна нелінійна регресійна модель зворотно перетворена згідно з формулою (2.10),(2.11),(2.12), та (2.13).

Після обчислень були отримані наступні значення коефіцієнтів: $\beta_0 = -1{,}7566$, β_1 = 0,5425, β_2 = 0,5816, β_3 = -0,1212.

Таким чином, остаточно нелінійна регресійна модель згідно з формулою (2.14) має вигляд:

$$\hat{Y} = 10^{\varepsilon - 1{,}7566,} * X_1{}^{0{,}5425} * X_2{}^{0{,}5816} * X_3{}^{-0{,}1212}.$$

Розрахунок параметрів якості для побудованої нелінійної регресійної моделі здійснюється для оцінювання її достовірності та адекватності. Для оцінки якості моделі використані метрики: коефіцієнт детермінації згідно з формулою (2.15), скоригований коефіцієнт детермінації згідно з формулою (2.16), MMRE згідно з формулою (2.17) і PRED(0,25) згідно з формулою (2.18). Аналіз якості моделі було проведено з використанням таких критеріїв:

– R^2 (коефіцієнт детермінації) – показник, що відображає, яку частку варіації залежної змінної пояснює модель;

– *MMRE* (Mean Magnitude of Relative Error) – середня величина відносної похибки, яка показує середнє відхилення передбачених значень від фактичних;

– *PRED* (0,25) – відсоток прогнозів, які мають відносну похибку меншу або рівну 25%.

Результати оцінювання для навчальної та тестової вибірок:

– Навчальна вибірка: R^2 = 0,9302, $R^2_{Adjusted}$=0,9264, *MMRE* = 0,2438, *PRED*(0,25) = 0,5254.

– Тестова вибірка: R^2 = 0,8459, $R^2_{Adjusted}$= 0,8338, *MMRE* = 0,2630, *PRED*(0,25) = 0,5476.

Розроблена нелінійна регресійна модель демонструє задовільні результати прогнозування як на навчальній, так і на тестовій вибірках. Значення R^2, яке наближається до 0,9, вказує на те, що модель ефективно пояснює варіацію залежної змінної KLOC. Низькі показники *MMRE* та задовільний рівень *PRED*(0,25) підтверджують, що у моделі задовільні показники і вона є надійним інструментом для оцінювання розміру застосунків, що використовують 2D графічні рушії.

3 РОЗРОБКА ПРОЕКТУ ПРОГРАМИ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ЗАСТОСУНКІВ, ЩО ВИКОРИСТОВУЮТЬ 2D ГРАФІЧНІ РУШІЇ

3.1 Постановка задачі на розробку програми для оцінювання розміру застосунків, що використовують 2D графічні рушії

З метою створення інструменту для точного оцінювання програмних застосунків, які використовують 2D графічні рушії, визначено наступну задачу: розробити програмне забезпечення, здатне прогнозувати розмір застосунків на основі спеціалізованої математичної моделі. Модель повинна враховувати специфіку цих застосунків, використовуючи такі ключові метрики: NC (кількість класів), NM (середня кількість методів на клас), DIT (глибина дерева наслідування) і KLOC (тисячі рядків коду), забезпечуючи достовірне прогнозування.

На основі аналізу предметної області та математичних моделей сформульовано вимоги до розроблюваного програмного забезпечення, яке отримало назву «AppSight Analyzer».

Програмне забезпечення «AppSight Analyzer» орієнтоване на автоматизацію процесу оцінювання розміру застосунків, створених із використанням 2D графічних рушіїв. Основна мета системи – спрощення процесу планування та управління розробкою програмних продуктів за рахунок автоматизованого прогнозування розміру проектів. Функціональність ПЗ має охоплювати два ключові завдання:

– Створення нової моделі оцінювання розміру – побудова регресійної моделі на основі введених даних (NC, NM, DIT, KLOC) для прогнозування розміру застосунків.

– Оцінювання розміру із застосуванням існуючої моделі – перевірка прогнозних значень для конкретного застосунку за допомогою раніше створеної моделі.

Функціональність ПЗ організована у формі двох основних сценаріїв.

Сценарій 1: Побудова нової моделі

Обробка даних:

– Завантаження набору даних у форматі CSV зі метриками NC, NM, DIT, KLOC;

– Попередня підготовка даних: нормалізація, видалення аномалій;

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

Побудова моделі:

– Формування лінійної регресійної моделі з використанням методу лів;

– Побудова нелінійної регресійної моделі із застосуванням 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 розміру на основі існуючої моделі

Введення даних:

– Введення значень метрик NC, NM, DIT для оцінюваного застосунку.

– Перевірка допустимості введених значень, включаючи аналіз відстані Махаланобіса для визначення відповідності моделі.

Прогнозування:

– Використання моделі для розрахунку розміру застосунку. Виведення результатів:

– Побудова довірчих інтервалів та інтервалів прогнозування.

– Надання користувачу прогнозних значень розміру та звіту про відповідність введених метрик моделі.

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:

– Для створення моделі та її використання: набір даних у форматі CSV, що містить метрики NC, NM, DIT,KLOC.

Для оцінювання розміру:

– Значення NC, NM, DIT метрик для конкретного проекту.

2. Вихідні дані

Для побудови нової моделі:

– Параметри нелінійної регресійної моделі;

– Показники якості моделі (коефіцієнт детермінації R^2, *MMRE*, *PRED*(0,25));

Для оцінювання якості:

– Прогнозний розмір застосунку.

– Результати перевірки припустимості введених метрик.

Ці вимоги формують основу для проектування системи, що дозволить підвищити ефективність оцінювання розміру застосунків із використанням 2D графічних рушіїв і забезпечить точність прогнозування на ранніх етапах розробки.

3.2 Архітектура програми для оцінювання розміру застосунків, що використовують 2D графічні рушії

Для створення програмного забезпечення була обрана архітектура MVC (Model-View-Controller). Це рішення обґрунтоване її здатністю ефективно розподіляти завдання, що забезпечує покращену підтримку, масштабованість і повторне використання коду. Застосування MVC в аналізі розміру застосунків із використанням 2D графічних рушіїв дозволяє досягти високого рівня організованості системи завдяки її трьом ключовим компонентам: моделі (Model), представленню (View) і контролеру (Controller). Основні функції цих компонентів такі:

Модель (Model):

– Відповідає за управління даними та бізнес-логікою.

– Здійснює отримання та обробку даних із бази даних, включаючи проєктні характеристики, метрики та параметри якості регресійної моделі.

– Забезпечує цілісність і узгодженість даних, надаючи механізми доступу до них і можливості їхньої модифікації.

Представлення (View):

– Зосереджено на логіці відображення інформації користувачеві.

– Забезпечує візуалізацію даних у вигляді графіків, таблиць або інших елементів інтерфейсу користувача.

– Дозволяє користувачеві взаємодіяти з системою, надаючи дані у зручному й інтуїтивно зрозумілому форматі.

Контролер (Controller):

– Виконує роль посередника між моделлю та представленням.

– Обробляє запити користувача, оновлює стан системи через модель і передає зміни представленню.

– Забезпечує синхронізацію даних між іншими компонентами, управляючи логікою взаємодії.

Взаємодія між цими компонентами може бути детально проілюстрована за допомогою UML-діаграм, що демонструють структурну організацію системи та механізми зв’язку між її елементами.

Основні переваги використання архітектури MVC:

– Чітке розділення обов’язків між компонентами сприяє модульності та полегшує підтримку системи.

– Повторне використання моделі та контролера на різних платформах знижує витрати на розробку.

– Легке розширення функціоналу шляхом модифікації або додавання окремих компонентів без втручання в роботу інших.

– Зручність модульного тестування кожного компонента, що гарантує його коректність незалежно від інших частин системи [34].

Таким чином, архітектура MVC є оптимальним вибором для створення програмного забезпечення, що потребує гнучкості та високої організованості структури.

Діаграми відіграють критичну роль у візуалізації складних процесів, перетворюючи абстрактні концепції на зрозумілі графічні зображення. Для створення подальших діаграм було використано плагін PlantUML [35].

PlantUML - потужний інструмент з відкритим вихідним кодом, який дозволяє генерувати різноманітні типи діаграм за допомогою простої текстової мови. Він підтримує UML-діаграми, схеми бізнес-процесів, мережеві та структурні діаграми, надаючи розробникам та фахівцям зручний спосіб графічного представлення інформації.

Перевагою PlantUML є його простота: замість складного графічного редагування користувач може описати діаграму текстом, і плагін автоматично

створить відповідне зображення. Такий підхід значно спрощує процес документування та підвищує ефективність командної роботи.

На рисунку 3.1 представлено загальну діаграму компонентів програмного забезпечення для оцінювання розміру застосунків, що використовують 2D графічні рушії.

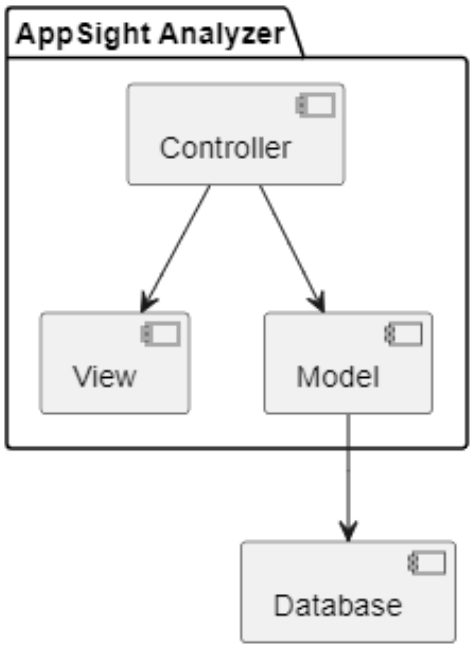


Рисунок 3.1 – Діаграма компонентів програмного забезпечення для оцінювання розміру застосунків, що використовують 2D графічні рушії

Ця схема демонструє архітектуру програмного забезпечення за моделлю MVC та її інтеграцію з локальною базою даних. Контролер взаємодіє як із представленням, так і з моделлю, яка зберігається у базі даних [36]. На рисунку 3.2 показано ланцюжок взаємодії між компонентами програмного забезпечення, яке призначене для оцінювання розміру застосунків, що використовують 2D графічні рушії.

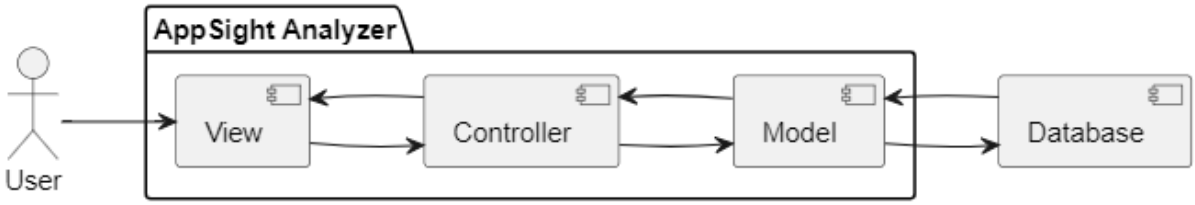


Рисунок 3.2 – Діаграма взаємодії компонентів програмного забезпечення для оцінювання розміру застосунків, що використовують 2D графічні рушії

Ця діаграма ілюструє архітектуру взаємодії компонентів, яка включає наступні етапи:

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

2. Представлення (View) передає введені користувачем дані Контролеру (Controller) і отримує оновлений стан для відображення.

3. Контролер (Controller) координує потік даних між Представленням (View) та Моделлю (Model). Він приймає дані від Представлення, обробляє їх і передає в Модель, а також отримує оновлені дані від Моделі та передає їх назад у Представлення.

4. Модель (Model) відповідає за бізнес-логіку застосунку та зберігає його стан. Вона взаємодіє з Базою Даних (Database) для читання та запису даних.

5. База Даних (Database) зберігає постійні дані застосунку.

3.3 Розробка ескізного проекту програми для оцінювання розміру застосунків, що використовують 2D графічні рушії

На основі аналізу вимог до програмного забезпечення був розроблений ескізний проєкт програми. Спочатку була побудована діаграма варіантів використання, яка всебічно відображає основні функціональні сценарії та

взаємодію користувачів з системним рішенням для прогнозування розміру застосунків на базі 2D графічних рушіїв. [37].

На рисунку 3.3 представлена діаграма варіантів використання для аналізу розміру застосунків, які використовують 2D графічні рушії.

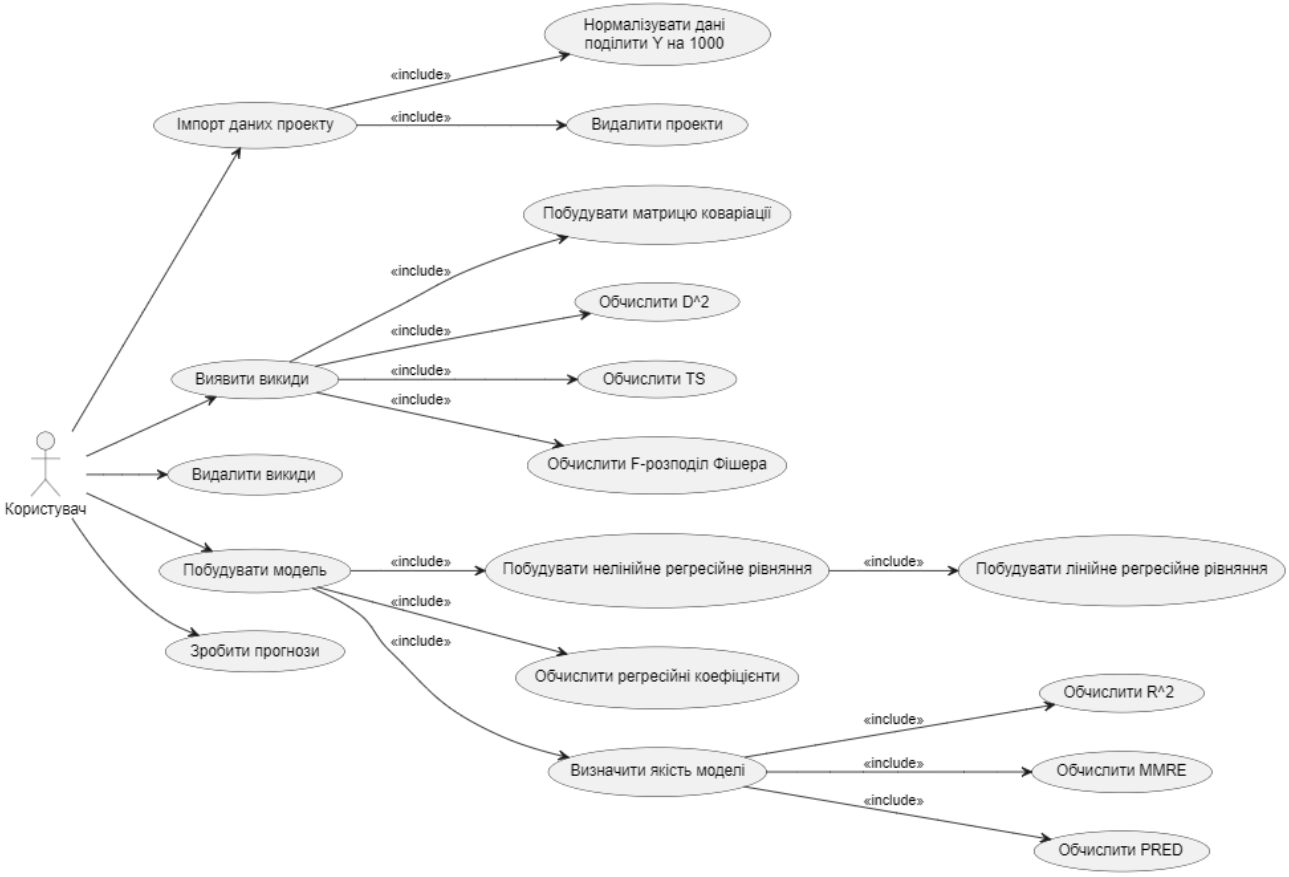


Рисунок 3.3 – Діаграма варіантів використання «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії

Ця UML-діаграма ілюструє ключові варіанти використання, що стосуються аналізу даних і створення системи прогнозування розміру застосунків на основі регресійної моделі. У діаграмі виділені наступні основні процеси:

Імпорт даних про проєкт – завантаження вихідних даних метрик застосунків, що використовують 2D графічні рушії, у форматі .csv для

подальшого аналізу. Цей процес включає розподіл даних на навчальну (60%) та тестову (40%) вибірки.

Огляд ітерацій видалення викидів – виявлення аномальних значень у даних, які можуть вплинути на точність моделі. Цей процес включає видалення викидів за допомогою коваріаційної матриці, квадратів відстані Махаланобіса, тестової статистики, F-розподілу Фішера.

Побудова регресійної моделі – створення як лінійної, так і нелінійної регресійної моделі на основі навчальної вибірки (60%). Цей процес передбачає розрахунок коефіцієнтів регресії.

Оцінювання якості моделі – аналіз отриманої регресійної моделі за допомогою таких критеріїв, як R^2, *MMRE, PRED*(0,25).

Здійснення прогнозів – отримання прогнозних значень якості проєктів на основі побудованої регресійної моделі, використовуючи метрики NC, NM та DIT.

Таким чином, діаграма детально описує ключові етапи аналізу даних і побудови прогнозної моделі для оцінювання розміру застосунків.

У процесі розробки програмного забезпечення «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії, було створено ряд документів для формалізації поведінки та структури системи. Серед них була діаграма діяльності, яка ілюструє сценарії основних варіантів використання [38]. Також були розроблені специфікації до діаграми використання, деякі з них представлені далі.

Специфікація 1: Імпорт даних проекту

Короткий опис:

Даний випадок використання забезпечує користувачу можливість імпортувати та підготувати дані проекту для подальшого аналізу та моделювання.

Передумови:

– Дані мають бути підготовлені у прийнятному форматі для імпорту.

– Користувач має доступ до інтерфейсу системи.

Основний потік:

– Користувач завантажує масив даних до системи.

– Система виконує нормалізацію даних шляхом поділу значення Y на 1000.

– Система автоматично перевіряє коректність імпортованих даних та виводить повідомлення про потенційні невідповідності.

– Користувач видаляє зайві або неінформативні проекти за допомогою функціоналу фільтрації.

– Система формує готовий до аналізу набір нормалізованих даних.

Альтернативний потік:

Якщо дані містять помилки, система повідомляє користувача та пропонує вручну виправити або перезавантажити дані.

Постумови:

Підготовлений, нормалізований набір даних доступний для подальшого статистичного аналізу.

Специфікація 2: Виявлення та видалення викидів

Короткий опис:

Цей випадок використання спрямований на очищення масиву даних від статистичних викидів для підвищення точності моделювання.

Передумови:

– Дані мають бути імпортовані та нормалізовані.

– Користувач має доступ до функціоналу аналізу даних.

Основний потік:

– Система обчислює матрицю коваріації для оцінки взаємозв'язків між змінними.

– Система розраховує відстань D^2 для визначення відхилень точок від центру розподілу.

– Виконується тестова статистика TS та порівнюється з F-розподілом Фішера.

– Система ідентифікує значущі викиди та пропонує їх видалення.

– Користувач підтверджує автоматичне або manual видалення викидів.

Альтернативний потік:

Якщо система не виявляє статистично значущих викидів, очищення не проводиться, і аналіз продовжується.

Постумови:

Очищений набір даних доступний для подальшого моделювання.

Специфікація 3: Побудова моделі

Короткий опис:

Цей випадок використання забезпечує створення регресійної моделі для прогнозування на основі імпортованих даних.

Передумови:

– Масив даних підготовлений та очищений від викидів.

– Користувач має доступ до функціоналу побудови моделей.

Основний потік:

– Користувач обирає тип моделі (нелінійна чи лінійна).

– Система обчислює регресійні коефіцієнти для обраної моделі.

– Система оцінює якість моделі через показники R^2, *MMRE* та *PRED*.

– У разі відповідності встановленим критеріям модель зберігається як готова до прогнозування.

Альтернативний потік:

Якщо модель не задовольняє критерії якості, користувач може повторити моделювання з іншими параметрами або типом моделі.

Постумови:

Побудована регресійна модель з розрахованими показниками якості доступна для прогнозування та аналізу.

Представлені специфікації охоплюють ключові процеси: імпорт даних, виявлення та видалення статистичних викидів, а також побудову прогностичної моделі. Кожна специфікація детально описує логіку взаємодії користувача з системою, висвітлюючи як основні, так і альтернативні сценарії роботи.

На рисунку 3.4 представлено діаграму діяльності «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії.

Процес починається з імпорту проектних даних, які необхідні для аналізу. Далі виконується видалення викидів: для кожного спостереження розраховуються квадрати відстаней Махаланобіса для нормалізованих даних. На основі цих відстаней обчислюється тестова статистика, після чого визначається критичне значення за допомогою *F*-розподілу Фішера. Якщо тестова статистика перевищує критичне значення, викиди видаляються, і процес повторюється, доки всі викиди не будуть вилучені.

Після видалення викидів дані діляться на навчальну (60%) і тестову (40%) вибірки, і будується рівняння лінійної регресії. Коефіцієнти регресії розраховуються методом найменших квадратів, а потім проводиться зворотне перетворення для отримання рівняння нелінійної регресії. Після цього оцінюється якість побудованої моделі.

Останнім етапом є здійснення прогнозування якості на основі створеної регресійної моделі. В цьому процесі розраховуються оцінка розміру застосунку за метриками NC, NM і DIT. На цьому етапі процес завершується.

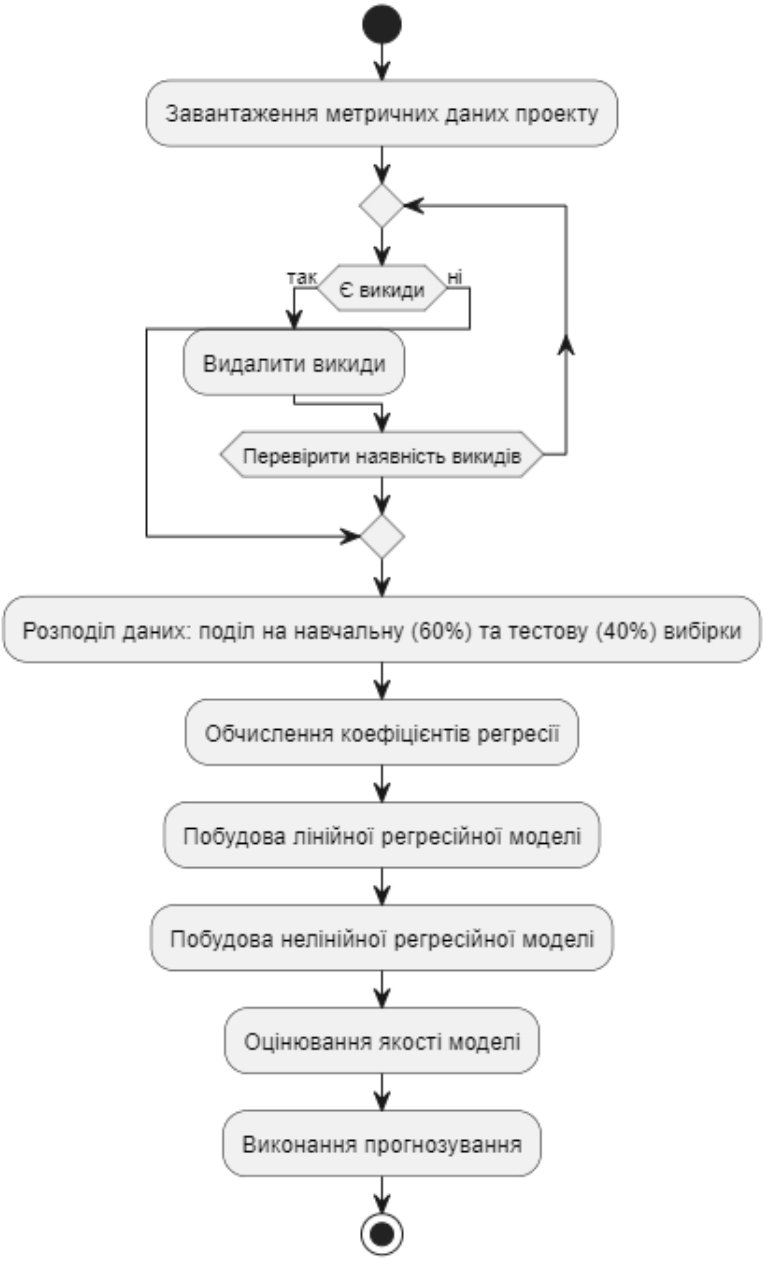


Рисунок 3.4 – Діаграма діяльності «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії

На рисунку 3.5 представлено діаграму діяльності «AppSight Analyzer» для розділення вибірки на дві для побудови та тестування моделі.

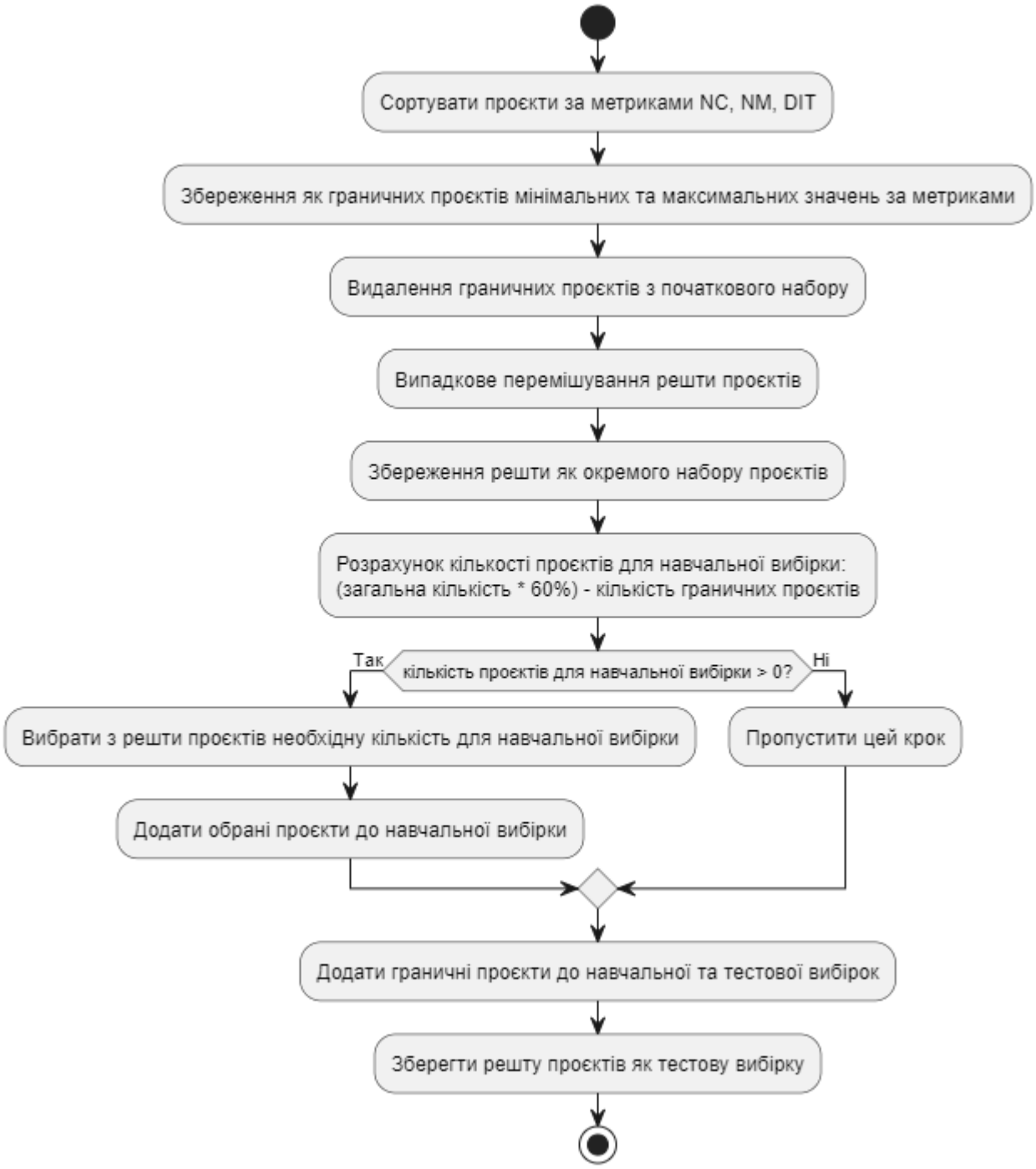


Рисунок 3.5 – Діаграма діяльності «AppSight Analyzer» для розділення вибірки на дві для побудови та тестування моделі

Алгоритм розподіляє список проєктів на навчальну та тестову вибірки відповідно до заданого співвідношення (60%). Він також забезпечує, щоб у

навчальній і тестовій вибірках завжди були представлені крайні значення (мінімальні та максимальні) для кожної з визначених метрик. Спочатку створюється локальна копія початкового списку проєктів, щоб уникнути змін у вихідних даних. Список проєктів сортується за кожною метрикою окремо: NC, NM, DIT. З відсортованих списків вибираються проєкти з найменшими та найбільшими значеннями метрик, які вважаються важливими, оскільки вони відображають крайні випадки даних. Формується множина з мінімальних та максимальних проєктів для кожної метрики, що гарантує унікальність об’єктів. З основного списку видаляються всі проєкти, які входять до множини крайніх значень. Решта проєктів перемішуються випадковим чином для забезпечення випадкового розподілу. Визначається кількість проєктів, необхідних для навчальної вибірки (60%), враховуючи співвідношення та вже вибрані крайні проєкти. Навчальна вибірка складається з крайніх проєктів та частини випадково вибраних проєктів зі залишку. Інші проєкти формують тестову вибірку.

Прототипи інтерфейсу користувача є важливим етапом проектування програмних додатків та веб-сервісів. Вони слугують попередньою версією майбутнього інтерфейсу, яка дозволяє команді розробників та дизайнерів ретельно продумати структуру, послідовність взаємодії та зручність використання продукту. Створення прототипів допомагає визначити основні функціональні елементи, перевірити логіку навігації та передбачити можливі проблеми до початку безпосередньої розробки. Існує кілька рівнів прототипування – від найпростіших паперових ескізів до інтерактивних макетів, які максимально наближені до кінцевого продукту. Такий підхід значно скорочує витрати на розробку, оскільки дає змогу виявити та виправити недоліки на ранніх стадіях проектування, а також забезпечує ефективну комунікацію між членами команди та замовниками [39].

Прототипи інтерфейсу користувача зображені на рисунках 3.6 – 3.9.

Рисунок 3.6 – Прототип головного екрана «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії

Головне вікно має нижню панель з розділами «Проекти та метрики», «Коваріаційна матриця», «Виявлення викидів», «Видалення викидів» і «Регресійний аналіз». У верхній частині розташована основна область, де відображаються завантажені ПЗ. Кожен проект містить нумерацію, URL та метрики:

- Y – KLOC (Thousands of Lines of Code);
- X1 – NC (Number of Classes);
- X2 – NM (Number of Methods Per Class);
- X3 – DIT (Depth of Inheritance Tree).

Рисунок 3.7 – Прототип екрана «Коваріаційна матриця» «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії

Вкладка «Коваріаційна матриця» відображає коваріаційну матрицю та обернену коваріаційну матрицю. Використовуючи обернену коваріаційну матрицю ми вираховуємо квадрат відстаней Махолонобіса.

Рисунок 3.8 – Прототип екрана «Виявлення викидів» «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії

Вкладка «Виявлення викидів» відображає: квадрат відстані Махаланобіса, тестову статистику, *F*-розподіл Фішера (alpha = 0,005). Ця вкладка дозволяє дізнатися про викіди.

Рисунок 3.9 – Прототип екрана «Регресійний аналіз» «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії

Вкладка «Виявлення викидів» відображає рівняння лінійної регресії, рівняння нелінійної регресії. Ця вкладка дозволяє виконувати регресійний аналіз, для оцінювання розміру застосунків за допомогою метрик NC, NM та DIT. Знизу наведена якість нелінійної моделі.

3.4 Розробка техничного проекту програми для оцінювання розміру застосунків, що використовують 2D графічні рушії

Після детального аналізу початкового проекту було успішно розроблено комплексний технічний проєкт програмного забезпечення «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії. У контексті дослідження архітектурних рішень системи «AppSight Analyzer» розглянемо її статичну структуру, яка відображає взаємозв'язки та організацію ключових класів [40]. На рисунку 3.10 представлена статична модель «AppSight Analyzer» у вигляді діаграми класів.

Ця UML діаграма класів ілюструє систему, яка аналізує метрики програмних проєктів і створює регресійну модель для прогнозування KLOC.

Опис класів

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

– SoftwareProject: клас, що представляє окремий програмний проєкт. Містить інформацію про назву проєкту, його URL та об'єкт класу Metrics, який зберігає метрики проєкту.

– Metrics: клас, що зберігає різні метрики проєкту, такі як NC, NM, DIT та KLOC.

– Outliers: клас, що відповідає за обробку даних проєктів. Нормалізує дані, розраховує коваріаційну матрицю та її обернену, визначає відстані Махаланобіса, тестову статистику та видаляє викиди.

– RegressionModel: клас, що представляє регресійну модель. Розділяє проєкти на навчальну та тестову вибірки, обчислює коефіцієнти регресії, оцінює якість моделі та здійснює прогнози розміру проєктів.

– ModelMetrics: клас, який зберігає показники якості моделі, такі як коефіцієнт детермінації (R^2), середню абсолютну похибку (*MMRE*) та прогнозоване значення (*PRED*(0,25)).

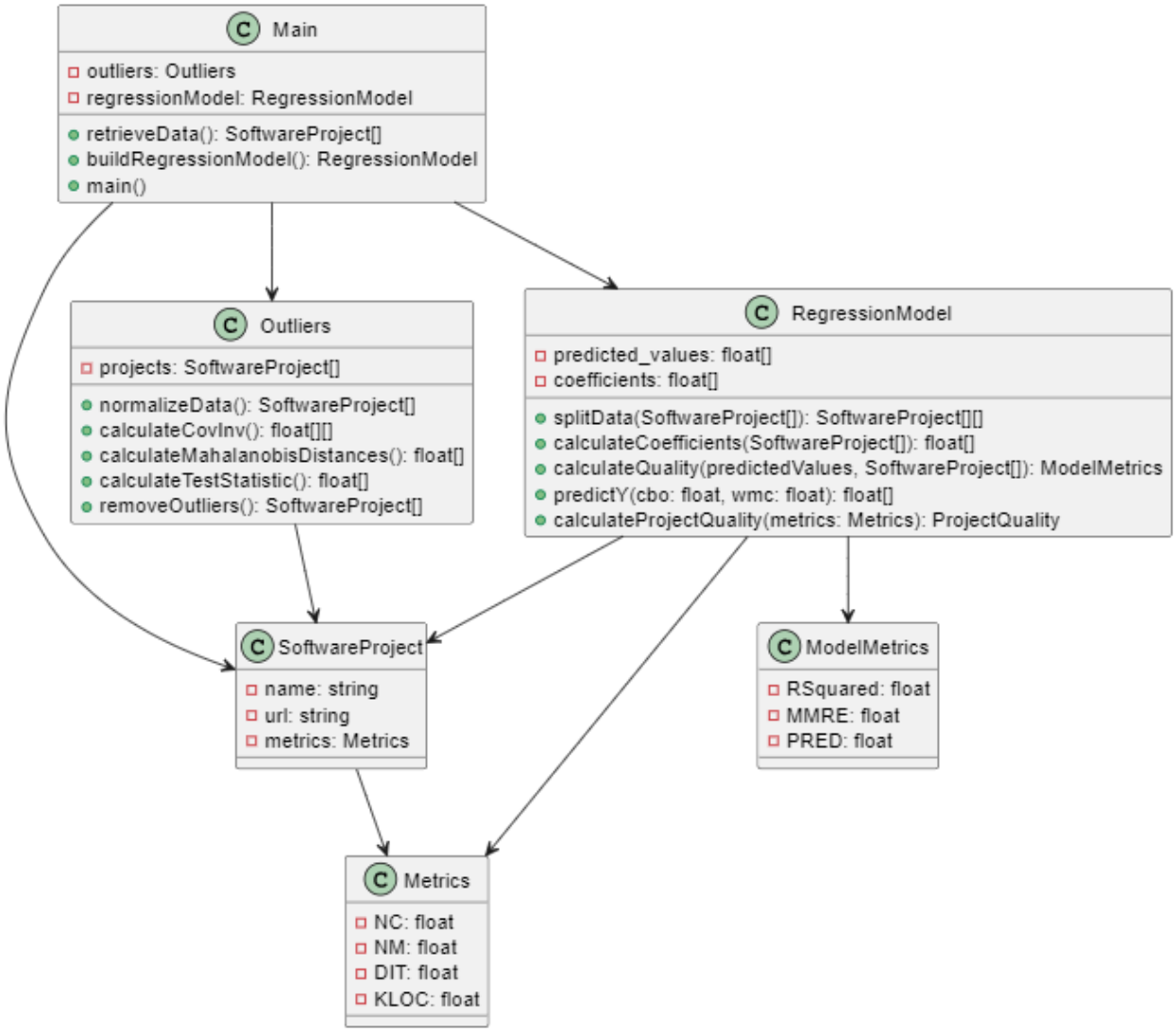


Рисунок 3.10 – Діаграма класів «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії

Зв'язки між класами

– Залежність: клас Main залежить від класів Outliers і RegressionModel, що означає використання цих класів для виконання своєї роботи.

– Агрегація: клас SoftwareProject містить об'єкт класу Metrics, що вказує на зв'язок «має».

– Асоціація: інші зв'язки на діаграмі показують взаємозв'язки між класами, але не обов'язково означають, що один клас містить інший.

Діаграма послідовностей нижче ілюструє процес побудови регресійної моделі для оцінювання розміру програмного забезпечення, а також розрахунку та відображення метрик розміру моделі [41]. У цьому процесі задіяні користувач (User), інтерфейс користувача (UI), основний модуль (Main), обробник даних (DataHandler), модуль регресійної моделі (RegressionModel) та модуль оцінювання метрик якості моделі (ModelQualityMetrics).

Процес починається з того, що користувач ініціює побудову моделі через інтерфейс користувача. Далі основний модуль запитує очищені дані у обробника даних. Після отримання даних модуль RegressionModel спочатку обчислює лінійні коефіцієнти, а потім використовує їх для побудови нелінійної моделі. Далі проводиться оцінка моделі з розрахунком ключових метрик якості: R^2, *MMRE* та *PRED*(0,25). Завершальним етапом є передача розрахованих метрик до інтерфейсу користувача для їх візуалізації. Діаграма послідовностей представлена на рисунку 3.11.

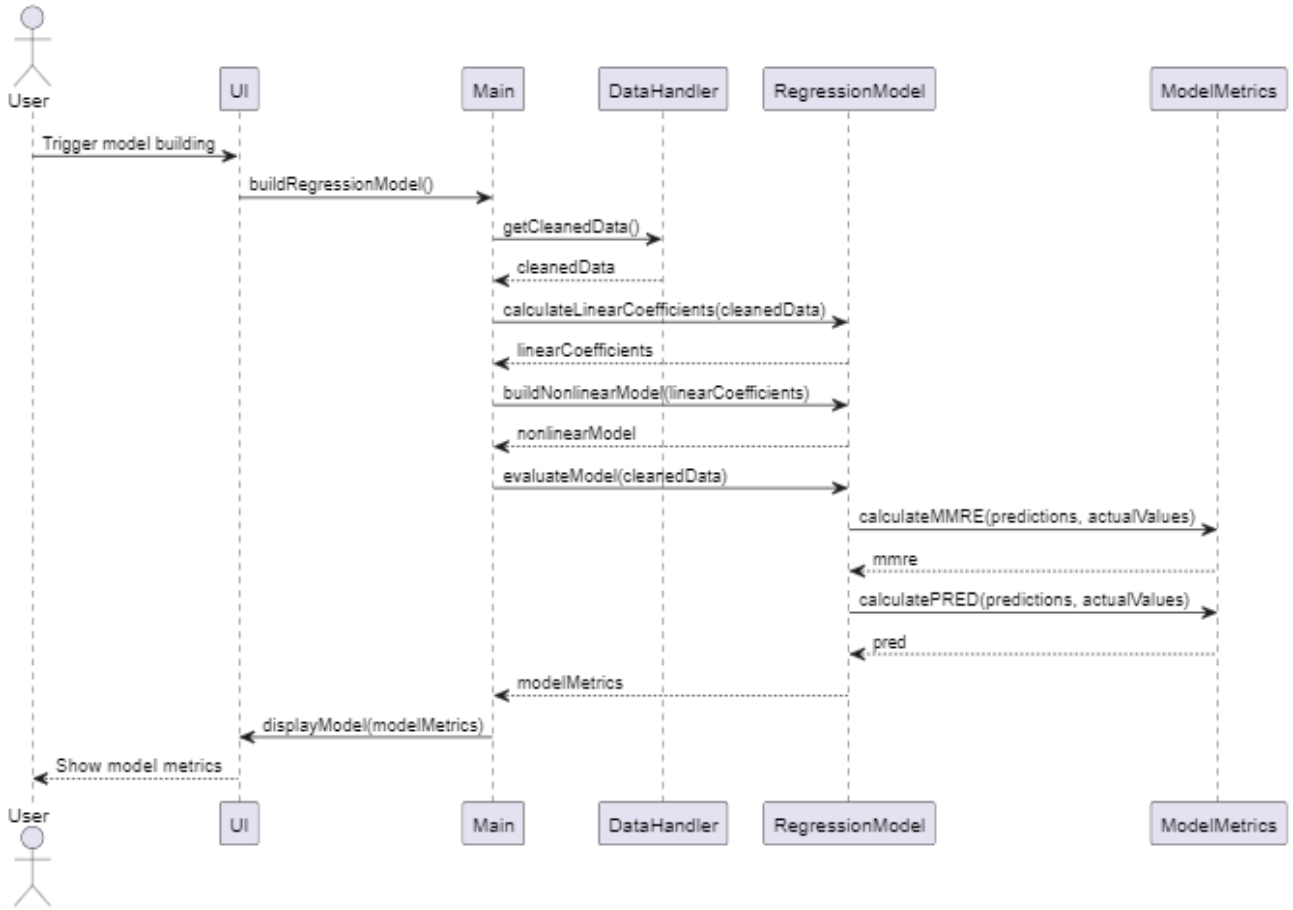


Рисунок 3.11 – Діаграма послідовностей «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії

На діаграмі видно, що взаємодія між компонентами системи чітко структурована. Кожен модуль виконує свою конкретну задачу: обробка даних, розрахунок коефіцієнтів, побудова моделей та обчислення метрик. Метрики R^2, *MMRE* та *PRED*(0,25) забезпечують кількісну оцінку достовірності прогнозів моделі, що дозволяє оцінити якість побудованої моделі. Візуалізація цих метрик через інтерфейс користувача завершує процес, роблячи результати доступними та зрозумілими для користувача. Такий підхід забезпечує послідовність операцій, розподіл відповідальності між модулями та точність розрахунків.

3.5 Розробка робочого проекту програми для оцінювання розміру застосунків, що використовують 2D графічні рушії

Середовищем розробки для реалізації програмного забезпечення «AppSight Analyzer» буде Visual Studio Code [42] – легкий, але потужний редактор коду, розроблений компанією Microsoft. Він забезпечує підтримку численних мов програмування, а також багатий функціонал для редагування коду, налагодження та інтеграції з іншими інструментами, що робить його ідеальним для розробки складних програмних рішень.

Для створення інтерфейсу програми буде застосована система дизайну Material Design [43], розроблена Google [44]. Ця система використовується для розробки веб-сайтів, мобільних додатків і інших цифрових продуктів, забезпечуючи створення сучасних, чітких і зручних для користувача інтерфейсів. Вона відповідає принципам зручності та інтуїтивності, що сприяє кращому користувацькому досвіду.

Фреймворк Flutter стане основою для розробки програмного забезпечення «AppSight Analyzer». Flutter – це потужний і відкритий кросплатформений фреймворк, який дозволяє створювати мобільні, веб- і настільні додатки на основі однієї кодової бази. Завдяки інтеграції з Material Design, Flutter дозволяє створювати елегантні та зручні інтерфейси користувача, які виглядають однаково добре на різних пристроях. Щоб реалізувати проект, потрібно буде завантажити та встановити Flutter SDK, що забезпечить доступ до необхідних інструментів і команд для створення, побудови та запуску проекту [45].

Мова програмування Dart буде використовуватись разом з Flutter для написання коду додатка «AppSight Analyzer». Файли Dart (.dart) використовуватимуться для створення компонентів інтерфейсу користувача, а також для обробки взаємодії користувача з програмою. Dart є

високопродуктивною мовою, що оптимізована для роботи з Flutter, і забезпечує простоту в написанні коду та ефективність у виконанні [46].

Використання Flutter для розробки цього програмного забезпечення має кілька ключових переваг. Одна кодова база дозволяє зменшити витрати на розробку та обслуговування додатка, оскільки програму можна запускати на різних платформах (iOS, Android, Windows, Linux, MacOS і Web) без необхідності переписування коду. Це значно скорочує час розробки та полегшує підтримку та масштабування програмного продукту.

Офіційне сховище пакетів для Dart і Flutter – це pub. Цей менеджер пакетів дозволяє розробникам легко знаходити, встановлювати та управляти залежностями, що використовуються в проекті. Завдяки цьому, додавання нових можливостей і функцій стає зручним і швидким процесом [47].

Процес ініціалізації та налаштування проекту передбачає кілька етапів. Спочатку потрібно завантажити та встановити Flutter SDK, що включає всі необхідні інструменти для роботи. Потім створюється новий проект за допомогою команди «flutter create», яка генерує базову структуру файлів і каталогів. Далі у файл «pubspec.yaml» додаються необхідні залежності, серед яких пакети, що полегшують завантаження даних, обробку CSV-файлів (file_picker, csv), а також пакет для керування станом програми (provider).

Наступним кроком є написання коду на Dart, що включатиме створення компонентів інтерфейсу, реалізацію функціоналу для завантаження даних, видалення викидів та побудову регресійної моделі. Завдяки використанню бібліотек та пакетів з pub.dev, процес розробки стане ще зручнішим.

Після завершення основної розробки, додаток буде зібраний для платформи Windows за допомогою команди «flutter build windows». Ця команда створює оптимізовані ресурси, які можна використовувати для розгортання на комп’ютерах з операційною системою Windows.

В результаті використання всіх зазначених технологій та інструментів, програмне забезпечення «AppSight Analyzer» буде готове до роботи. Воно надасть користувачеві можливість завантажувати дані, що містять інформацію про метрики NC, NM, DIT,KLOC і на основі цих даних створювати регресійну модель, що дозволяє прогнозувати розмір застосунків, які використовують 2D графічні рушії.

Завдяки обраним технологіям, «AppSight Analyzer» стане зручним, потужним та універсальним інструментом для аналізу та прогнозування, що спрощує роботу розробників і забезпечує високу продуктивність на всіх основних платформах. Головна сторінка застосунку представлена на рисунку 3.12.

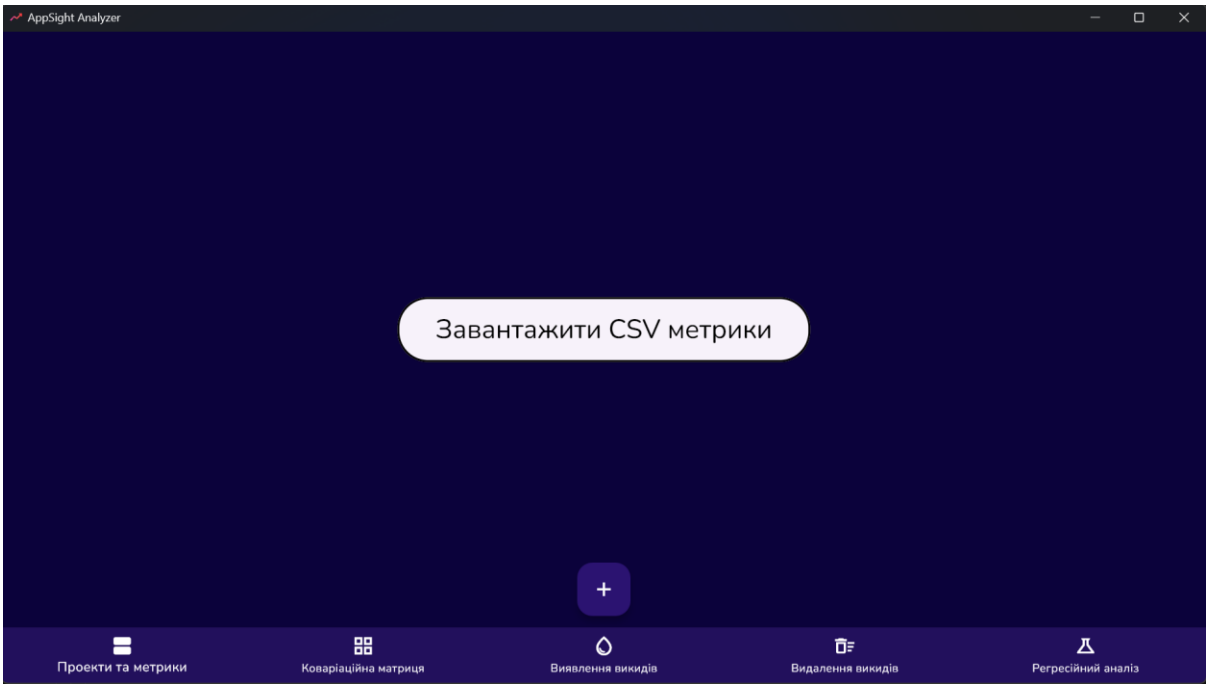


Рисунок 3.12 – Головна сторінка застосунку «AppSight Analyzer»

На головній сторінці можна побачити кнопку із завантаженням даних із .csv файлу, нижні компоненти меню, такі як:

– Проекти та метрики – перегляд завантажених проектів з метриками.

– Коваріаційна матриця – перегляд коваріаційної матриці та оберненої коваріаційної матриці.

– Виявлення викидів – перегляд виявлених викидів.

– Видалення викидів – список проектів з викидами для видалення.

– Регресійний аналіз – прогнозування та якість нелінійної регресійної моделі.

Після того, як данні були завантаженні, їх обробляє застосунок та відображає у вкладці «Проекти та метрики». На рисунку 3.13 представлено вкладку «Проекти та метрики» із завантаженими даними. Користувач може завантажити їх у форматі .csv.

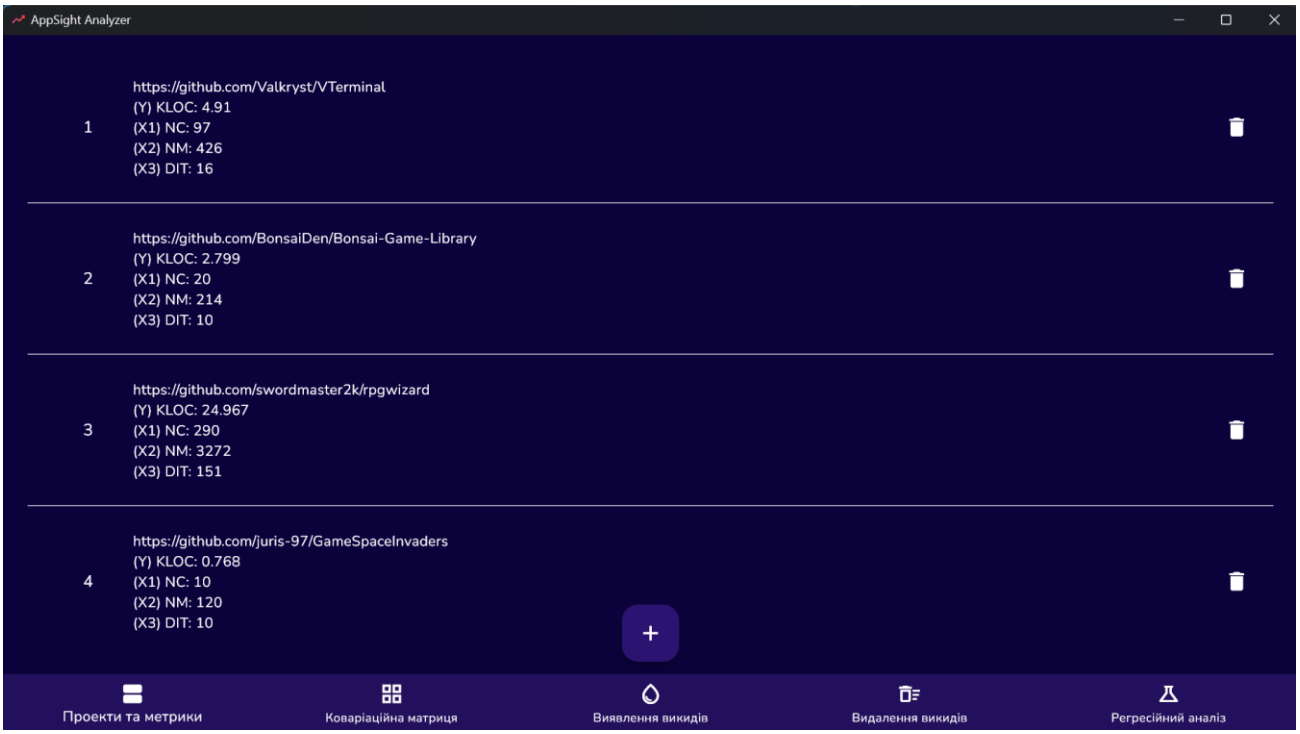
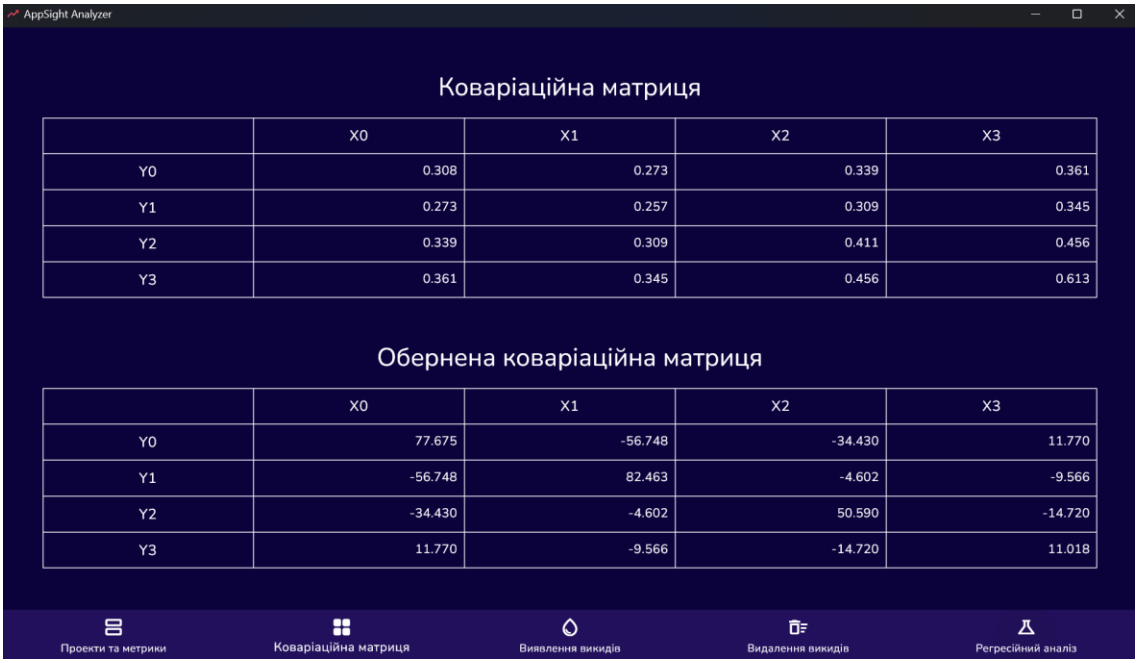


Рисунок 3.13 – Вкладка «Проекти та метрики» застосунку «AppSight Analyzer»

Користувач може переглянути коваріаційну матрицю та обернену коваріаційну матрицю. На рисунку 3.14 представлено вкладку «Коваріаційна матриця».

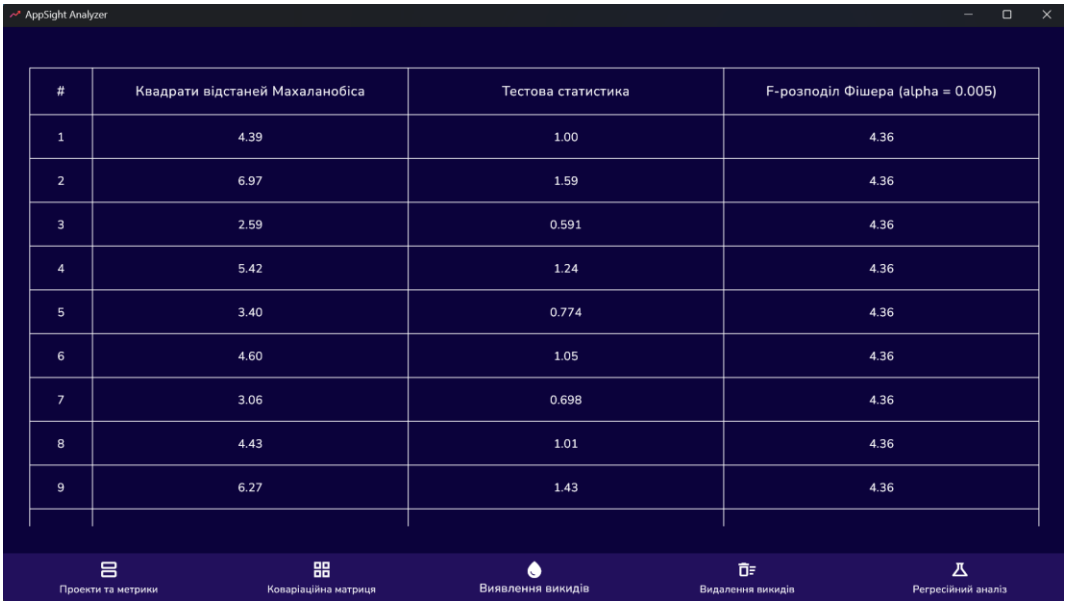


	X0	X1	X2	X3
Y0	0.308	0.273	0.339	0.361
Y1	0.273	0.257	0.309	0.345
Y2	0.339	0.309	0.411	0.456
Y3	0.361	0.345	0.456	0.613

	X0	X1	X2	X3
Y0	77.675	-56.748	-34.430	11.770
Y1	-56.748	82.463	-4.602	-9.566
Y2	-34.430	-4.602	50.590	-14.720
Y3	11.770	-9.566	-14.720	11.018

Рисунок 3.14 – Вкладка «Коваріаційна матриця» застосунку «AppSight Analyzer»

Користувач може переглядати таблицю з значеннями квадрата відстані Махалонобіса, тестової статистики та *F*-розподілу Фішера. Якщо значення тестової статистики більше за *F*-розподілу Фішера, то це свідчить про викид. На рисунку 3.15 представлена вкладка «Видалення викидів»



#	Квадрати відстаней Махаланобіса	Тестова статистика	F-розподіл Фішера (alpha = 0.005)
1	4.39	1.00	4.36
2	6.97	1.59	4.36
3	2.59	0.591	4.36
4	5.42	1.24	4.36
5	3.40	0.774	4.36
6	4.60	1.05	4.36
7	3.06	0.698	4.36
8	4.43	1.01	4.36
9	6.27	1.43	4.36

Рисунок 3.15 – Вкладка «Видалення викидів» застосунку «AppSight Analyzer»

На рисунку 3.16 представлена вкладка «Регресійний аналіз» (побудована лінійна та нелінійна регресійна модель із завантажених даних та її якість, без викидів).

Рисунок 3.16 – Вкладка «Регресійний аналіз» застосунку «AppSight Analyzer»

Під час тестування програмного забезпечення «AppSight Analyzer» для аналізу розміру застосунків, що використовують 2D графічні рушії, були проведені наступні тести для перевірки відповідності вимогам 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.

Для оцінки якості програмного рішення проведено комплексне тестування його функціональних характеристик та продуктивності. Результати тестування представлено в таблиці 3.1.

Таблиця 3.1 – Результати тестування програмного забезпечення

№	Дія	Очікуваний результат	Результат
1	Завантаження даних з CSV файлу	Дані успішно завантажені та відображаються у вкладці «Проекти та метрики».	Успішно
2	Виявлення та видалення викидів	Викиди успішно виявляються та видаляються та відображаються у вкладці «Виявлення викидів» та успішно видаляються у вкладці «Видалення викидів».	Успішно
3	Побудова регресійної моделі	Регресійна модель успішно побудована та відображається у вкладці «Регресійний аналіз».	Успішно
4	Здійснення прогнозів	Прогнози успішно здійснюються та відображаються у вкладці «Регресійний аналіз».	Успішно

У ході тестування програмного забезпечення «AppSight Analyzer» всі функціональні вимоги були перевірені та підтверджені. Завантаження даних із CSV-файлу виконувалося коректно, виявлення та видалення викидів відбувалися без помилок, побудова моделі здійснювалася відповідно до заданих параметрів, прогнози були успішними.

Було виконано випробування розробленого ПЗ «AppSight Analyzer» згідно Програми та методики, наведеної у додатку Д.

Таким чином, програмне забезпечення «AppSight Analyzer» продемонструвало успішне проходження тестування та випробування і може бути ефективно використане для оцінювання розміру застосунків, які використовують 2D графічні рушії.

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВІД РОЗРОБКИ І ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ЗАСТОСУНКІВ, ЯКІ ВИКОРИСТОВУЮТЬ 2D ГРАФІЧНІ РУШІЇ

Створене програмне забезпечення для оцінювання розміру застосунків, які використовують 2D графічні рушії, відповідає сучасним вимогам IT-ринку. Зростаюча популярність 2D графічних рушіїв зумовлює необхідність у спеціалізованих інструментах, які сприяють підвищенню ефективності та якості розробки програмних продуктів.

Розробка орієнтована на програмістів, які працюють із 2D графічними рушіями, а також на компанії, що займаються створенням застосунків із використанням таких рушіїв. Очікується, що програмний продукт буде затребуваним завдяки своїм унікальним можливостям автоматизованого аналізу та оцінювання програмного коду.

4.1 Розрахунок витрат на розробку програмного забезпечення для оцінювання розміру застосунків, які використовують 2D графічні рушії

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 (7 пачок)	700
Заправлення картриджа до принтера	800
CD-диски (2 шт.)	200
Література	3000
Непередбачені витрати	200
Разом	4900

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	35 000
Додаткова заробітна плата (премії тощо)	3 000
Відрахування на соціальні заходи (22%)	7 700
Загальногосподарські витрати	3 500
Витрати на допоміжні матеріали	2 000
Витрати на електроенергію	3 500
Разом	51 200

Далі, після завершення попередніх етапів, можемо переходити до оцінки економічної ефективності.

4.2 Прогнозовані доходи та оцінка економічної ефективності

Програмне забезпечення буде реалізовано за ліцензійною моделлю з ціною однієї ліцензії 500 грн. Прогнозований обсяг продажів на перший рік становить 1000 ліцензій. Очікуваний дохід: 1000 × 500 = 500 000 грн.

Чиста приведена вартість (NPV) [49]:

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

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

$$NPV = \frac{500\,000}{(1+0.1)^1} - 56\,100 = 456\,545 - 56\,100 = 400\,445 \text{ грн.}$$

Рентабельність інвестицій (ROI) [50]:

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

$$ROI = \frac{500\,000 - 51\,200}{51\,200} \times 100\% = 8{,}76\%.$$

Період окупності:

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

Щомісячний дохід (рівномірний розподіл): $500\,000/12 = 41\,666$ грн.

Період окупності $= 51\,200 \,/\, 41\,666 = 1{,}22$ місяців.

Економічний аналіз підтверджує перспективність проекту: програмне забезпечення може повністю окупити витрати вже за 1,22 місяця та гарантує стабільний прибуток у майбутньому.

5 ОХОРОНА ПРАЦІ

5.1 Загальні питання охорони праці

Охорона праці є невід'ємною складовою трудової діяльності, яка має на меті забезпечення безпеки, здоров'я та благополуччя працівників у процесі виконання ними своїх обов'язків [51]. Це багатогранний напрямок, який охоплює правові, соціально-економічні, організаційно-технічні, санітарно-гігієнічні та лікувально-профілактичні заходи. Ефективна система охорони праці сприяє створенню комфортних і безпечних умов для виконання роботи, мінімізації ризиків та підвищенню продуктивності працівників.

Основними завданнями охорони праці є вирішення інженерно-технічних та соціальних питань [52].

Інженерно-технічний напрямок спрямований на впровадження технологій і методів, що знижують ризики, пов'язані з трудовим процесом. Сюди входять розробка безпечних робочих процесів, використання сучасного обладнання, що відповідає стандартам безпеки, а також забезпечення працівників засобами індивідуального захисту. Наприклад, автоматизація небезпечних операцій або встановлення систем моніторингу виробничих умов дозволяють суттєво знизити ймовірність нещасних випадків чи професійних захворювань. Крім того, важливу роль відіграє регулярне технічне обслуговування обладнання, своєчасна заміна застарілих механізмів та впровадження інноваційних рішень, які роблять робочий процес безпечнішим.

Соціальний аспект охорони праці орієнтований на забезпечення справедливості, компенсацій та підтримки працівників. Він охоплює компенсаційні виплати у випадку отримання травм чи професійних захворювань, страхування працівників від нещасних випадків, створення нормативно-правової бази, що регулює трудові відносини, а також захист прав працівників від

дискримінації та несправедливого ставлення. Важливо, щоб усі заходи соціальної підтримки були спрямовані не лише на відшкодування шкоди, а й на створення умов, які дозволяють уникнути таких ситуацій у майбутньому.

Законодавче регулювання у сфері охорони праці є основою для забезпечення безпеки на робочих місцях. В Україні ключовим нормативним документом є Закон «Про охорону праці», який визначає права та обов'язки всіх учасників трудового процесу [53]. Цей закон встановлює вимоги щодо організації робочого середовища, проведення навчання та інструктажу з техніки безпеки, обов’язкового використання засобів індивідуального захисту, періодичного медичного огляду працівників тощо. Важливо зазначити, що контроль за виконанням законодавчих норм здійснюють спеціальні державні органи, які також відповідають за розробку рекомендацій і стандартів для вдосконалення системи охорони праці.

Сьогодні охорона праці вимагає комплексного підходу, який включає спільну відповідальність роботодавців, працівників і держави. Роботодавці повинні забезпечувати безпечні умови праці, інвестувати в сучасні технології та проводити регулярне навчання персоналу. Працівники, у свою чергу, мають дотримуватись встановлених правил безпеки та проходити необхідні інструктажі. Держава виконує роль гаранта, який встановлює законодавчі норми, здійснює контроль і сприяє розвитку культури безпеки.

Особливу увагу варто приділяти постійному вдосконаленню системи охорони праці. Сучасні умови виробництва динамічно змінюються, і це вимагає адаптації заходів безпеки до нових викликів. Впровадження цифрових технологій, таких як системи моніторингу небезпеки або автоматизовані платформи аналізу ризиків, є важливим кроком у цьому напрямку. Регулярне оновлення навчальних програм і підвищення обізнаності працівників також сприяють зменшенню виробничих ризиків.

5.2 Дослідження потенційних ризиків та небезпек під час роботи з моніторами

Робота з моніторами є важливою складовою сучасного офісного середовища, яка значно підвищує ефективність виконання завдань у багатьох сферах діяльності. Водночас тривала робота за комп’ютером супроводжується потенційними ризиками, які можуть негативно впливати на фізичне і психологічне здоров’я працівників. Саме тому дослідження цих ризиків є надзвичайно актуальним і вимагає особливої уваги з боку роботодавців та спеціалістів з охорони праці.

Тривале використання моніторів пов’язане з низкою проблем, які можуть впливати на зір, опорно-руховий апарат, психоемоційний стан та загальне самопочуття працівників [54]. Однією з найбільш поширених небезпек є порушення зору, які виникають через необхідність довготривалого фокусування погляду на екрані. До основних проявів цих порушень належать втома очей, сухість або подразнення, розмитість зору та підвищена чутливість до світла. Причинами таких проблем є недостатнє освітлення робочого місця, неправильне налаштування яскравості та контрасту дисплея, а також нерегулярність перерв у роботі.

Не менш важливим є вплив роботи з моніторами на опорно-руховий апарат [55]. Багато працівників змушені проводити тривалий час у статичній позі, що може викликати болі у шиї, плечах, спині та зап’ястках. Неправильна організація робочого місця, зокрема невідповідна висота столу чи стільця, а також незручне розташування клавіатури чи миші, значно посилюють ці ризики. Частими наслідками такої діяльності є порушення постави та розвиток тунельного синдрому зап’ястка, який виникає через постійне навантаження на кисті.

Крім фізичних аспектів, слід звернути увагу на психологічні ризики, які можуть виникати внаслідок тривалої роботи з моніторами. Постійна

зосередженість, монотонність завдань та суворі дедлайни нерідко призводять до емоційного вигорання, підвищеного рівня стресу та зниження концентрації уваги. Такі стани не лише знижують продуктивність працівників, але й негативно впливають на їх загальне самопочуття.

Окремо варто зазначити можливий вплив електромагнітного випромінювання, яке генерують моніторами. Хоча сучасні дисплеї відповідають міжнародним стандартам безпеки, тривалий контакт з такими пристроями може спричиняти дискомфорт, особливо за умов наявності інших несприятливих факторів, таких як стрес чи недостатнє освітлення.

Для зменшення негативного впливу роботи з моніторами на здоров’я працівників необхідно приділити особливу увагу організації робочого місця. Важливо використовувати ергономічні меблі, які забезпечують комфортну посадку, правильну підтримку спини та рук. Монітор слід розміщувати на рівні очей та на оптимальній відстані від користувача, що зазвичай становить від 50 до 70 сантиметрів. Освітлення в приміщенні має бути достатньо яскравим, але не створювати відблисків на екрані.

Дотримання режиму праці та відпочинку також є важливим аспектом профілактики. Рекомендується робити короткі перерви кожні 50–60 хвилин, під час яких варто виконувати вправи для очей та легкі фізичні розминки. Це дозволяє знизити напруження м’язів і покращити кровообіг, запобігаючи розвитку фізичного дискомфорту.

Технологічні рішення також можуть сприяти покращенню умов роботи з моніторами. Використання програмного забезпечення для фільтрації синього світла та налаштування параметрів яскравості і контрасту екрана відповідно до рівня освітлення дозволяють зменшити навантаження на очі. Регулярний моніторинг стану здоров’я працівників, зокрема консультації з офтальмологом та ортопедом, є важливим кроком для своєчасного виявлення та усунення проблем.

Зменшення психоемоційного навантаження досягається завдяки створенню комфортного психологічного клімату в колективі, забезпеченню підтримки з боку керівництва та оптимальному плануванню робочого часу. Такі заходи сприяють підвищенню загального рівня задоволеності працею та зниженню стресу.

Робота з моніторами, попри численні переваги, може бути пов’язана з ризиками для здоров’я працівників. Однак їх своєчасне виявлення та вжиття відповідних заходів дозволяють мінімізувати негативний вплив. Поєднання ергономічного підходу, раціональної організації робочого місця, використання сучасних технологій та регулярного медичного моніторингу є ключем до створення безпечних і комфортних умов праці.

5.3 Способи протидії та мінімізації негативного впливу під час роботи з моніторами

Зменшення впливу негативних факторів під час роботи з екранними пристроями є важливим завданням, яке потребує комплексного підходу. Одним із найефективніших методів є впровадження заходів для оптимізації робочого місця. Для цього необхідно забезпечити відповідність меблів ергономічним вимогам, а також правильно розташувати обладнання. Зокрема, екран монітора повинен знаходитися на рівні очей працівника, на відстані не менше 50–70 сантиметрів, що дозволяє знизити навантаження на шию та очі. Особливу увагу слід приділити освітленню робочого місця, уникаючи прямого або відбитого світла, яке може викликати небажані відблиски на екрані. Крім того, важливо коректно налаштувати яскравість та контраст дисплея, адаптуючи їх до умов зовнішнього освітлення.

Регулярні перерви в роботі відіграють значну роль у підтриманні здоров’я працівників [56]. Кожні 50–60 хвилин варто робити короткі перерви тривалістю 5–10 хвилин. Під час цих пауз рекомендовано виконувати вправи для очей, які зменшують напругу та покращують кровообіг, а також фізичні розминки для м’язів спини, шиї та рук. Це сприяє зниженню втоми та попередженню розвитку проблем опорно-рухового апарату.

Застосування сучасних технологій є ще одним дієвим методом боротьби з негативним впливом роботи за комп’ютером. Наприклад, використання програмного забезпечення для фільтрації синього світла допомагає зменшити навантаження на зір, особливо у вечірній час. Крім того, варто звертати увагу на технічні характеристики моніторів і вибирати моделі з низьким рівнем випромінювання та функціями, які знижують втому очей.

Медичний моніторинг працівників є важливим аспектом профілактики ризиків. Регулярні медичні огляди допомагають виявляти проблеми на ранніх стадіях і вживати необхідних заходів. У разі появи дискомфорту або перших симптомів втоми очей чи болю в спині слід звертатися за консультацією до офтальмолога чи ортопеда. Важливим є також навчання працівників правильній організації робочого процесу та використанню обладнання. Інформування щодо правил ергономіки, основних симптомів професійних захворювань і методів їх запобігання допомагає знизити ризик негативних наслідків.

Для зменшення психоемоційного навантаження варто створювати комфортні умови праці. Роботодавці можуть організовувати психологічну підтримку працівників, забезпечувати можливість гнучкого графіка або змінюваності діяльності для уникнення монотонності. Планування робочого часу з урахуванням необхідності регулярного відпочинку та дотримання нормального балансу між роботою та особистим життям також сприяє покращенню загального самопочуття співробітників.

Комплексний підхід до вирішення проблем, пов’язаних із роботою з екранними пристроями, дозволяє значно знизити їхній негативний вплив на здоров’я працівників. Впровадження рекомендацій щодо організації робочого місця, дотримання режиму праці та відпочинку, використання сучасних технологій та регулярний медичний контроль є необхідними умовами для забезпечення безпеки та підвищення ефективності трудової діяльності.

6 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

6.1 Загальні питання охорони навколишнього середовища

Сучасні інформаційні технології справляють значний вплив на навколишнє середовище, і цей вплив має двоїстий характер. З одного боку, розвиток ІТ-сфери сприяє зниженню споживання ресурсів та оптимізації виробничих процесів, що має позитивні наслідки для екологічної ситуації. Автоматизація бізнес-процесів, впровадження цифрових технологій і розвиток кросплатформних рішень дозволяють значно скоротити витрати на матеріальні ресурси, знизити обсяги відходів і зменшити потребу у використанні паперу. Наприклад, завдяки цифровізації і розвитку дистанційної роботи зменшується необхідність у транспортуванні людей і товарів, що, своєю чергою, сприяє економії енергетичних ресурсів і знижує викиди шкідливих речовин у атмосферу. Також інноваційні підходи в управлінні бізнесом і виробництвом сприяють оптимізації використання енергії та ресурсів, що у свою чергу зменшує негативний вплив на природу [57].

З іншого боку, швидкий розвиток інформаційних технологій має свої негативні наслідки, серед яких слід відзначити збільшення обсягів електронних відходів та підвищене енергоспоживання. Серверні ферми та центри обробки даних, які підтримують цифрові послуги та інфраструктуру, потребують величезної кількості електроенергії, що навантажує енергосистему і сприяє збільшенню викидів парникових газів. Крім того, виробництво сучасного електронного обладнання – смартфонів, комп’ютерів, серверів – пов’язане з екологічними витратами через необхідність видобутку рідкісних металів та використання токсичних матеріалів, які шкодять довкіллю. Витрати, пов’язані з видобутком, переробкою і транспортуванням цих матеріалів, а також утилізація

електронних відходів, створюють додаткове навантаження на природні ресурси та екосистеми. В результаті, хоча сучасні технології надають безліч переваг у вигляді ефективності та зручності, їх вплив на навколишнє середовище залишається неоднозначним і потребує подальшого обдумування та відповідних кроків для зменшення негативних наслідків.

6.2 Створення програмного забезпечення з урахуванням екологічних принципів

Одним із ключових напрямків зменшення негативного впливу на довкілля є створення енергоефективного програмного забезпечення. В цьому контексті варто відзначити платформу Flutter, яка дозволяє розробляти кросплатформні додатки, що є відмінним прикладом сучасного підходу до програмування. Використання такої архітектури дозволяє уникнути дублювання коду для різних операційних систем, що, в свою чергу, сприяє зниженню навантаження на процесори та зменшенню енергоспоживання як під час розробки, так і при подальшій експлуатації додатків.

Створення програмного забезпечення з урахуванням екологічних принципів має значний потенціал для зниження загального енергоспоживання. Наприклад, оптимізація алгоритмів і зменшення кількості обчислювальних операцій сприяють зниженню споживаної енергії як на кінцевих пристроях (мобільних телефонах, комп’ютерах), так і на серверах, що в свою чергу зменшує негативний вплив на навколишнє середовище. Ефективне програмне забезпечення допомагає не лише знижувати енергетичні витрати, але й знижувати викиди парникових газів, оскільки менше енергії витрачається на обробку даних і виконання програм [58].

Додатково, використання кросплатформних рішень значно скорочує необхідність створення та підтримки тестових і нативних додатків для різних платформ. Це, в свою чергу, призводить до економії ресурсів на всіх етапах розробки, тестування та оновлення програмного забезпечення. Такий підхід не тільки знижує витрати часу та коштів, але й мінімізує вплив на екологію, сприяючи більш сталому розвитку IT-індустрії та створенню програмних рішень, що відповідають сучасним екологічним стандартам.

Для зменшення негативного впливу інформаційних технологій на навколишнє середовище розроблено кілька ключових стратегій. Однією з основних є оптимізація програмного коду. Створення коду, який ефективно використовує ресурси пристроїв, сприяє зниженню енергоспоживання як під час розробки, так і при подальшій експлуатації програмного забезпечення. Важливим етапом є впровадження енергоефективних методів тестування, що включають автоматизовані системи, які знижують кількість зайвих перевірок і таким чином економлять енергію та ресурси.

Крім того, слід активно впроваджувати технології віртуалізації для зниження потреби у фізичних серверах. Віртуалізація дозволяє значно зменшити енергетичні витрати та знизити витрати на обслуговування, що є критично важливим для оптимізації ресурсів у великих дата-центрах. Завдяки цьому, компанії можуть ефективніше використовувати апаратні ресурси, зменшуючи енергоспоживання та витрати на енергетичну підтримку серверного обладнання.

Ще одним важливим кроком є мінімізація використання паперу у всіх аспектах розробки та тестування програмного забезпечення. Перехід до електронного документообігу дозволяє знизити обсяги відходів та зберегти природні ресурси, такі як дерево, і зменшити витрати на транспортування та зберігання паперових документів.

Важливо також впроваджувати використання відновлювальних джерел енергії для забезпечення роботи серверів. Перехід дата-центрів на «зелену»

енергетику значно знижує залежність від традиційних джерел енергії та допомагає знизити викиди вуглекислого газу. Використання сонячних панелей, вітряних турбін і інших альтернативних джерел дозволяє компаніям і організаціям не тільки знижувати свій екологічний слід, але й підтримувати загальний баланс екосистеми.

Всі ці стратегії є важливою частиною концепції «зеленого IT», яка спрямована на інтеграцію сталих і екологічно безпечних практик у сферу інформаційних технологій. Впровадження таких підходів не тільки допомагає знизити витрати на енергію та зберегти екологію, але й створює можливості для більш ефективного управління ресурсами, сприяючи сталому розвитку індустрії.

6.3 Екологічні стандарти у циклі життя програмного забезпечення та їх вплив на навколишнє середовище

Життєвий цикл програмного забезпечення включає три основні стадії: розробку, використання та утилізацію, і кожна з них має свій екологічний вплив. На етапі розробки значна кількість енергії витрачається на тестування, обробку даних і програмування, що передбачає використання потужних комп’ютерних систем і серверів. Для зменшення впливу на довкілля важливо впроваджувати енергоефективні технології та використовувати джерела відновлювальної енергії, а також енергоощадне обладнання ще на цьому етапі розробки.

На стадії використання програмне забезпечення повинно бути спроектоване таким чином, щоб забезпечити мінімальне споживання енергії кінцевими пристроями, такими як смартфони, ноутбуки, комп’ютери та інші електронні гаджети. Оптимізація коду, зменшення навантаження на процесор під час виконання завдань і реалізація енергозберігаючих режимів роботи можуть

значно знизити енергоспоживання. Це важливо для зменшення вуглецевого сліду, а також для покращення ефективності роботи обладнання.

На етапі утилізації необхідно враховувати екологічні аспекти при знятті з експлуатації старих пристроїв і серверів [59]. Правильна утилізація електронного обладнання сприяє зменшенню обсягу електронних відходів і мінімізує вплив на навколишнє середовище. Переробка електронних компонентів дозволяє повторно використовувати цінні матеріали, знижуючи потребу у видобутку нових ресурсів та зменшуючи відходи, що потрапляють на звалища. Використання таких підходів допомагає знижувати шкоду для довкілля, забезпечуючи стале використання ресурсів і підтримку екологічної рівноваги.

Одним із ключових аспектів забезпечення сталого розвитку у сфері інформаційних технологій є впровадження міжнародних екологічних стандартів, які допомагають знижувати негативний вплив на довкілля та забезпечувати ефективне використання ресурсів. Визнані норми, як-от ISO 14001, який встановлює вимоги до систем екологічного менеджменту, відіграють важливу роль у створенні ефективних практик управління екологічною діяльністю на всіх етапах життєвого циклу програмного забезпечення – від планування та розробки до експлуатації та утилізації [60]. Ці стандарти допомагають організаціям інтегрувати екологічні принципи в повсякденну діяльність, підтримуючи таким чином збереження природних ресурсів і зменшення забруднення навколишнього середовища.

Важливо також використовувати системи оцінки, такі як EPEAT, яка дозволяє оцінювати та сертифікувати продукцію з точки зору її екологічної ефективності [61]. Вона включає в себе кілька критеріїв, що оцінюють вплив продуктів на довкілля, зокрема енергоефективність, матеріали, що використовуються, і можливість переробки. Цей підхід сприяє тому, щоб розробники і користувачі обирали технології, що відповідають найвищим екологічним стандартам.

Принципи «зеленого IT» не обмежуються лише енергозбереженням; вони також охоплюють використання відновлюваних джерел енергії, зменшення викидів парникових газів, скорочення обсягів електронних відходів і повторне використання матеріалів. Впровадження енергоефективних рішень, таких як оптимізація програмного коду, що знижує навантаження на процесори та інші компоненти, а також зменшення кількості обчислювальних операцій, є важливим кроком у цьому процесі. Програмне забезпечення має бути спроектоване так, щоб мінімізувати енергоспоживання в процесі його виконання, особливо при виконанні завдань, які не потребують значної обчислювальної потужності.

Також необхідно приділяти увагу утилізації та повторному використанню електронних пристроїв. При виведенні з експлуатації старих серверів і комп'ютерних систем важливо забезпечити їхню належну утилізацію. Переробка електронного обладнання не лише знижує обсяг відходів, а й дозволяє зберігати цінні матеріали, що можуть бути повторно використані у виробництві нових пристроїв. Це дозволяє зменшити витрати на видобуток нових матеріалів і сприяє зниженню викидів CO2.

Впровадження екологічних стандартів у розробку програмного забезпечення є важливою складовою частиною сталого розвитку IT-галузі. Такі заходи, як інтеграція відновлюваних джерел енергії, використання енергоефективних компонентів і систем, а також створення програмних рішень, що оптимізують ресурси, забезпечують зменшення негативного впливу на екологію. Коли організації впроваджують ці стратегії, вони не тільки дотримуються глобальних стандартів і норм, але й отримують вигоди у вигляді зниження витрат на енергію та збереження корпоративної репутації в умовах зростаючих екологічних вимог і громадської свідомості.

Врахування екологічних аспектів у створенні програмного забезпечення підтримує створення більш сталих технологічних рішень, які сприяють збереженню ресурсів планети, знижують викиди шкідливих речовин та

покращують умови для життя майбутніх поколінь. Завдяки цьому галузь інформаційних технологій може стати важливим елементом глобальних зусиль по боротьбі зі змінами клімату і підтримці екологічного балансу [62].

ВИСНОВКИ

У цій кваліфікаційній роботі було вдосконалено математичну модель для оцінювання розміру програмних застосунків, що працюють на 2D графічних рушіях. Модель використовує метрики кількість класів (NC), кількість методів у класі (NM) та глибину дерева успадкування (DIT) для прогнозування розміру у тисячах рядків коду (KLOC). Аналіз сучасних підходів показав, що традиційні моделі, розроблені для мов програмування Java, C# та PHP, не відповідають вимогам застосунків на 2D графічних рушіях через значні відмінності в їхній архітектурі та діапазонах метрик. Це підкреслило необхідність створення нової спеціалізованої моделі для таких застосунків.

Було проведено збір та аналіз даних із 108 відкритого проєкту, що використовує 2D графічні рушії. Після ретельної обробки даних та видалення викидів, було сформовано вибірку з 103 застосунків, яка послужила основою для побудови моделі.

Загалом, розроблена модель дозволяє ефективно оцінювати розмір програмних продуктів, що працюють на 2D графічних рушіях, що є важливим для підвищення достовірності оцінювання обсягу відповідного ПЗ і для покращення планування його розробки. Модель показала свою ефективність на тестових даних, демонструючи високу достовірність прогнозування.

Окрім того, було створено програму, яке автоматизує процес оцінювання розміру застосунків, що використовують 2D графічні рушії. Ця програма дозволяє збирати дані, аналізувати їх, а також будувати регресійну модель. Вона включає в себе зазначені вище метрики, що допомагають у більш достовірному прогнозуванні розміру ПЗ, і може стати практичним інструментом для розробників, які працюють із застосунками на 2D графічних рушіях.

Також було виконано розрахунок економічної ефективності від розробки та впровадження програми для оцінювання розміру застосунків, що використовують 2D графічні рушії, та розглянуто питання з охорони праці і охорони навколишнього середовища.

Мета роботи досягнута, усі завдання виконано в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Liu J. Design and development of 2D game 'Adventurous spirit' // Proceedings of the 2023 International Conference on Machine Learning and Automation. Dalian, China, February 2024. P. 102–109. URL: https://www.researchgate.net/publication/378435471_Design_and_development_of_2D_game_'Adventurous_spirit'

2. Software Metrics [Електронний ресурс]. URL: https://www.javatpoint.com/software-engineering-software-metrics (дата звернення: 01.10.2024).

3. Філіпчик А.А. Методи оцінки трудомісткості програмних проектів за методикою COCOMO II // Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення. Тернопіль, 2021. С. 47–48. URL: http://openarchive.nure.ua/handle/document/15065.

4. Макарова Л.М., Приходько Н.В., Кудін, О.О. Побудова нелінійної регресійної моделі для оцінювання розміру веб-додатків, реалізованих мовою Java. Вісник ХНТУ, 2019. 2(69), 145-153. URL: https://kntu.net.ua/ukr/content/download/82035/475659/file/Вісник%20№2(69).pdf

5. Петриченко С.А., Пухалевич А.В. Багатофакторна нелінійна регресійна модель для оцінювання розміру вебзастосунків, що створюються з використанням мови програмування C# // *Матеріали IV Всеукраїнської науково-практичної інтернет конференції "Інформаційні технології: моделі, алгоритми, системи (ITMAS – 2023)" (30-31 жовтня 2023 р., м. Миколаїв)*. Миколаїв: НУК імені адмірала Макарова, 2023. С. 40–42. URL: https://itconf.nuos.edu.ua/2023/wp-content/uploads/sites/6/NUOS-ITMAS-2023-Proceedings.pdf.

6. Латанська Л.О., Кольцов А.В. Нелінійна регресійна модель для оцінювання кількості строк коду веб-застосунків, що створюються з

використанням PHP фреймворку Symfony. // Матеріали III Всеукраїнської науково-практичної інтернет конференції *“Інформаційні технології: моделі, алгоритми, системи (ITMAS – 2022)”. (26-28 жовтня 2022 р. , м. Миколаїв).* Миколаїв: НУК імені адмірала Макарова, 2022. С. 10–11. URL: https://itconf.nuos.edu.ua/2022/wp-content/uploads/sites/5/NUOS-ITMAS-2022-Proceedings.pdf.

7. Приходько С.Б., Сербулов Г.В. Нелінійна регресійна модель для оцінювання розміру програмного забезпечення інформаційних систем з кодом на PHP // Матеріали II Всеукраїнської науково-практичної інтернет конференції *“Інформаційні технології: моделі, алгоритми, системи (ITMAS – 2021)”. (26-28 жовтня 2021 р. , м. Миколаїв).*Миколаїв, НУК імені адмірала Макарова, 2021. С. 13–14. URL: https://itconf.nuos.edu.ua/2021/wp-content/uploads/sites/4/NUOS-ITMAS-2021-Proceedings.pdf.

8. How to Estimate the Size of a Project [Електронний ресурс]. URL: https://www.digileaf.com/blog/project-management/estimate-size-project/(дата звернення: 04.10.2024).

9. Камінський С.С., Бризгалов М.В., Макарова Л.М. Математична модель для оцінювання розміру програмних застосунків, що використовують 2D графічні рушії // Матеріали V Всеукраїнської науково-практичної інтернет конференції *“Інформаційні технології: моделі, алгоритми, системи (ITMAS – 2024)”. (30-31 жовтня 2024 р. , м. Миколаїв).* Миколаїв, НУК імені адмірала Макарова, 2024. С. 69–70. URL: https://itconf.nuos.edu.ua/2024/wp-content/uploads/sites/7/NUOS-ITMAS-2024-Proceedings.pdf.

10. Камінський С.С., Макарова Л.М. Розробка програмного забезпечення для оцінювання розміру застосунків,що використовують 2D графічні рушії. *// XV Міжнародна науково-технічна конференція “Інновації в суднобудуванні та океанотехніці”.(26–27 вересня 2024 р. , м. Миколаїв).* Миколаїв, НУК імені

адмірала Макарова, 2024. С. 607–609. URL: https://nuos.edu.ua/wp-content/uploads/2024/10/Materiali-konferencii.pdf.

11. Software Measurement and Metrics [Електронний ресурс]. URL: https://www.geeksforgeeks.org/software-measurement-and-metrics/ (дата звернення: 04.10.2024).

12. WMC, CBO, RFC, LCOM, DIT, NOC - 'The Chidamber and Kemerer Metrics' [Електронний ресурс]. URL: http://www.virtualmachinery.com/sidebar3.htm (дата звернення: 07.10.2024).

13. Aanchal S.K. Metrics for Software Components in Object Oriented Environments: A Survey // International Journal of Scientific Research in Computer Science and Engineering. 2013. Vol. 1, Iss. 2. P. 1–5. URL: https://www.isroset.org/pub_paper/IJSRCSE/ISROSET-IJSRCSE-00043.pdf.

14. Depth of Inheritance Tree (DIT) [Електронний ресурс]. URL: https://dcm.dev/docs/metrics/class/depth-of-inheritance-tree (дата звернення: 10.10.2024).

15. Lines of Code (LOC) in Software Engineering [Електронний ресурс]. URL: https://www.geeksforgeeks.org/lines-of-code-loc-in-software-engineering/ (дата звернення: 06.10.2024).

16. MVC [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (дата звернення: 10.10.2024).

17. Abstract Window Toolkit (AWT) [Електронний ресурс]. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/awt/ (дата звернення: 12.10.2024).

18. Бібліотека рефлексивної генерації Swing-форм [Електронний ресурс]. URL: https://habr.com/articles/126579/ (дата звернення: 14.10.2024).

19. Java 2D API [Електронний ресурс]. URL: https://www.oracle.com/java/technologies/java-2d-api.html (дата звернення: 14.10.2024).

20. Що таке Java? [Електронний ресурс]. URL: https://aws.amazon.com/what-is/java/ (дата звернення: 14.10.2024).

21. GitHub [Електронний ресурс]. URL: https://github.com (дата звернення: 24.10.2024).

22. SourceMeter [Електронний ресурс]. URL: https://sourcemeter.com (дата звернення: 25.10.2024).

23. Multivariate Normality Testing (Mardia) [Електронний ресурс]. URL: https://real-statistics.com/multivariate-statistics/multivariate-normal-distribution/multivariate-normality-testing/ (дата звернення: 20.10.2024).

24. 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.

25. Відстань Махаланобіса [Електронний ресурс]. URL: https://askom.gitbooks.io/-/content/23_rasstoyanie_mahalanobisa.html (дата звернення: 20.10.2024).

26. Коваріаційна матриця та її вибіркова оцінка [Електронний ресурс]. URL:https://stud.com.ua/151093/ekonomika/kovariatsiyna_matritsya_vibirkova_otsinka (дата звернення: 21.10.2024).

27. Як розрахувати спостережуване значення тестової статистики? [Електронний ресурс]. URL:https://simpatiya.figa.cx.ua/vzaiemodiya/yak-rozrakhuvati-sposterezhuvane-znachennya-testovoi-statistiki.html (дата звернення: 21.10.2024).

28. F Distribution Tables [Електронний ресурс]. URL: http://www.socr.ucla.edu/Applets.dir/F_Table.html (дата звернення: 22.10.2024).

29. Коефіцієнт детермінації (R у квадраті) [Електронний ресурс]. URL: https://uk.economy-pedia.com/11038615-coefficient-of-determination-r-squared (дата звернення: 23.10.2024).

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

31. Jayakumar K., Abran A. Estimation Models for Software Functional Test Effort // Journal of Software Engineering and Applications. 2017. Vol. 10. No. 4. P. 338-353. DOI: https://doi.org/10.4236/jsea.2017.104020.

32. Критерії Пірсона та Стьюдента – застосування та відмінності [Електронний ресурс]. URL: https://diploms.kiev.ua/uk/kriteriyi-pirsona-ta-styudenta-zastosuvannya-ta-vidminnosti/ (дата звернення: 27.10.2024).

33. Що таке нормальний розподіл у статистиці? [Електронний ресурс]. URL:https://www.houseofmath.com/uk/encyclopedia/statystyka-ta-ymovirnist/imovirnist-i-kombinatoryka/rozpodily-ymovirnostey/shcho-take-normalnyy-rozpodil-u-statystytsi (дата звернення: 29.10.2024).

34. Що таке діаграма компонентів UML в OOAD? Позначення, приклад [Електронний ресурс]. URL: https://www.guru99.com/uk/component-diagram-uml-example.html (дата звернення: 1.11.2024).

35. PlantUML [Електронний ресурс]. URL: https://plantuml.com (дата звернення: 1.11.2024).

36. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти [Електронний ресурс]. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 1.11.2024).

37. Варіанти використання та сценарії (Use Cases and Scenarios) [Електронний ресурс]. URL: https://www.maxzosim.com/use-cases-and-scenarios/ (дата звернення: 2.11.2024).

38. Діаграма діяльності в UML: символ, компоненти та приклад [Електронний ресурс]. URL: https://www.guru99.com/uk/uml-activity-diagram.html (дата звернення: 3.11.2024).

39. Як протестувати інтерфейс на користувачах, якщо ви це робите вперше [Електронний ресурс]. URL: https://cases.media/en/article/yak-protestuvati-interfeis-na-koristuvachakh-yaksho-vi-ce-robite-vpershe?srsltid=A (дата звернення: 3.11.2024).

40. Що таке діаграма класів UML і найкращий творець діаграм класів UML [Електронний ресурс]. URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/ (дата звернення: 3.11.2024).

41. Діаграма послідовності (Sequence Diagrams) [Електронний ресурс]. URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 4.11.2024).

42. Visual Studio Code [Електронний ресурс]. URL: https://code.visualstudio.com (дата звернення: 5.11.2024).

43. Material Design [Електронний ресурс]. URL: https://m3.material.io (дата звернення: 5.11.2024).

44. Google [Електронний ресурс]. URL: https://www.google.com.ua/?hl=uk (дата звернення: 5.11.2024).

45. Flutter [Електронний ресурс]. URL: https://flutter.dev (дата звернення: 7.11.2024).

46. Яка мова використовує Flutter і що в ній особливе [Електронний ресурс]. URL: https://spacelab.ua/articles/kakoy-yazyk-ispolzuyet-flutter-i-chto-v-nem-osobennogo/ (дата звернення: 10.11.2024).

47. dart pub [Електронний ресурс]. URL: https://dart.dev/tools/pub/cmd (дата звернення: 10.11.2024).

48. Що таке тестування ПЗ, його етапи, види, інструменти [Електронний ресурс]. URL: https://university.sigma.software/what-is-software-testing/ (дата звернення: 15.11.2024).

49. Що таке чиста поточна вартість (NPV)? [Електронний ресурс]. URL: https://finances.in.ua/shcho-take-chysta-potochna-vartist-npv/ (дата звернення: 17.11.2024).

50. Що таке ROI, та Де порахувати рентабельність інвестицій [Електронний ресурс]. URL: https://netpeak.net/uk/blog/shcho-take-roi-ta-de-porakhuvati-rentabel-nist-investitsiy/ (дата звернення: 17.11.2024).

51. Про охорону праці: Закон України від 15.11.2024 № 2694-XII [Електронний ресурс]. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 09.11.2024).

52. Основні функції служби охорони праці [Електронний ресурс]. URL: https://dsp.gov.ua/faq/osnovni-funktsii-sluzhby-okhorony-pratsi/ (дата звернення: 19.11.2024).

53. 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).

54. Комп'ютер і до сліпоти може довести [Електронний ресурс]. URL: https://armyinform.com.ua/2020/08/30/kompyuter-i-do-slipoty-mozhe-dovesty/ (дата звернення: 23.11.2024).

55. Здоров'я користувача ПК: опорно-руховий апарат [Електронний ресурс]. URL: https://i.factor.ua/ukr/journals/bk/2009/february/issue-4/article-100268.html (дата звернення: 23.11.2024).

56. Перерви під час роботи за комп'ютером: чи входять в тривалість робочої зміни [Електронний ресурс]. URL: https://buhplatforma.com.ua/news/24275-perervi-pd-chas-roboti-za-kompyuterom-chi-vhodyat-v-trivalst-robocho-zmni (дата звернення: 24.11.2024).

57. Про охорону навколишнього природного середовища : Закон України від 15.11.2024 № 1264-XII [Електронний ресурс]. URL: https://zakon.rada.gov.ua/laws/show/1264-12#Text (дата звернення: 24.11.2024).

58. Як розробити ЕКОполітику з нуля? [Електронний ресурс]. URL: https://ecolog-ua.com/news/yak-rozrobyty-ekopolityku-z-nulya (дата звернення: 25.11.2024).

59. Як відповідально утилізувати електротехніку? Кілька лайфхаків і важливих правил [Електронний ресурс]. URL: https://kosmotech.com.ua/yak-vidpovidalno-utylizuvaty-elektrotehniku-kilka-lajfhakiv-i-vazhlyvyh-pravyl/ (дата звернення: 26.11.2024).

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

61. Overview of the EPEAT Ecolabel [Електронний ресурс]. URL: https://epeat.net/about-epeat (дата звернення: 29.11.2024).

62. 10 КРОКІВ для протидії зміні клімату [Електронний ресурс]. URL: https://ecoaction.org.ua/10-krokiv.html (дата звернення: 1.12.2024).

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «AppSight Analyzer»

Вступ

Повна назва програмного продукту: Програмне забезпечення «AppSight Analyzer» для визначення розміру застосунків, що використовують 2D графічні рушії.

Коротка назва: «AppSight Analyzer».

Сфера застосування: компанії, що займаються розробкою застосунків, що використовують 2D графічні рушії.

1. Підстави для розробки

Підставою для розробки «AppSight Analyzer» 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 розміру застосунків, що використовують 2D графічні рушії.

2.2 Функціональне призначення

Функціональним призначенням «AppSight Analyzer» є завантаження та перегляд метрик застосунків, що використовують 2D графічні рушії, побудова регресійної моделі для оцінки розміру проектів та прогноз розміру.

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 (NC, NM, DIT,KLOC);

– 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, такими, як R^2, *MMRE, PRED*(0,25);

– використання моделі для оцінювання розміру проекту на основі заданих NC, NM та DIT;

3.1.2 Вимоги до організації вхідних та вихідних даних

Дані про метрики та модель зберігаються у локальній базі даних.

Вхідні дані:

– файл у форматі CSV, що містять метрики застосунків, що використовують 2D графічні рушії: NC, NM, DIT,KLOC;

– метрики NC, NM, DIT для визначення розміру.

Вихідні дані:

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

– викиди;

– параметри побудованої лінійної моделі;

– показники якості нелінійної регресійної моделі;

– показники якості тестової вибірки для побудованої моделі;

– інтервали прогнозування;

– прогнозні значення розміру 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);

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 використовує 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;

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	15.09.2024
2	Розробка архітектури ПЗ	25.09.2024
3	Розробка проекту ПЗ	15.10.2024
4	Реалізація та тестування ПЗ	1.11.2024
5	Випробування ПЗ	5.11.2024

7. Порядок контролю та приймання

Контроль за аналізом та проектуванням кожного компонента програмного забезпечення має здійснюватися на всіх етапах розробки з урахуванням вимог, визначених у технічному завданні. Кожен етап розробки повинен бути завершений у встановлені терміни та узгоджений із замовником для забезпечення своєчасності та відповідності очікуванням.

Прийом робіт здійснюється відповідно до затвердженої програми та методики випробувань і оформлюється протоколом, що фіксує результати тестування. У разі виявлення дефектів під час приймальних випробувань складається акт про знайдені помилки, який підписують представники обох сторін – замовника і розробника, а також затверджується керівником організації-замовника та організації-розробника.

Розробник зобов'язаний протягом строку, що не перевищує 2 тижні, усунути зазначені помилки і повідомити замовника про готовність повторного проведення перевірки. Це забезпечує високий рівень контролю якості, своєчасне виявлення та усунення недоліків, а також сприяє досягненню високих стандартів якості програмного продукту.

ДОДАТОК Б – КОД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «AppSight Analyzer»

main.dart

```
import 'dart:async';
import 'dart:io';

import 'package:csv_metrics/app/app.dart';
import 'package:csv_metrics/services/logger.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:flutter_web_plugins/url_strategy.dart';
import 'package:window_size/window_size.dart';

final _log = AppLogger().logger;

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

    if (Platform.isWindows || Platform.isLinux || Platform.isMacOS) {
      setWindowTitle('AppSight Analyzer');
      // setWindowMaxSize(const Size(1920, 1440));
      setWindowMinSize(const Size(1280, 720));
    }

    usePathUrlStrategy();

    registerErrorHandlers();

    runApp(const App());
  }, (error, stackTrace) {
    _log.e(
      'Error',
      error: error,
      stackTrace: stackTrace,
    );
  });
}

void registerErrorHandlers() {
  FlutterError.onError = (details) {
    _log.e(
      'FlutterError',
      error: details.exception,
      stackTrace: details.stack,
```

```
    );
  };

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

outliers.py

```
from dataclasses import dataclass
import pandas as pd
import numpy as np
import os
from typing import List, Tuple
from scipy.stats import f
import random

@dataclass(frozen=True)
class Project:
    url: str
    x1: float
    x2: float
    x3: float
    y: float
    zx1: float = 0.0
    zx2: float = 0.0
    zx3: float = 0.0
    zy: float = 0.0
    ds: float = 0.0
    ts: float = 0.0

def data_path(filename: str) -> str:
    """Return the full path to the data file."""
    return os.path.join(os.path.dirname(__file__), os.path.join('data', filename))

def retrieve_data(filename: str = "100.csv") -> List[Project]:
    """Load data from CSV file and return a list of Project objects."""
    df = pd.read_csv(data_path(filename))
    return [Project(
        url=row['URL'],
        x1=row['NCL'],
        x2=row['NM'],
```

```
        x3=row['DIT'],
        y=row['LOC'],
    ) for _, row in df.iterrows()]

def normalize_data(projects: List[Project]) -> List[Project]:
    """Normalize the data using logarithmic transformation."""
    normalized_projects = []

    def log10(value):
        """Calculate log base 10, return 0 for values <= 0."""
        if value <= 0:
            return 0
        return np.log10(value)

    for project in projects:
        # Apply logarithmic normalization
        zx1 = log10(project.x1)
        zx2 = log10(project.x2)
        zx3 = log10(project.x3)
        zy = log10(project.y)

        normalized_projects.append(Project(
            url=project.url,
            x1=project.x1,
            x2=project.x2,
            x3=project.x3,
            y=project.y,
            zx1=zx1,
            zx2=zx2,
            zx3=zx3,
            zy=zy
        ))

    return normalized_projects

def projects_to_array(projects: List[Project]) -> np.ndarray:
    """Convert a list of Project objects to a numpy array."""
    return np.array([[p.zx1, p.zx2, p.zx3, p.zy] for p in projects])

def calculate_cov_inv(Z: np.ndarray) -> np.ndarray:
    """Calculate the inverse of the covariance matrix."""
    n = Z.shape[0]
    Z_centered = Z - np.mean(Z, axis=0)
    cov = (Z_centered.T @ Z_centered) / n
    return np.linalg.inv(cov)

def calculate_mahalanobis_distances(Z: np.ndarray, cov_inv: np.ndarray) -> np.ndarray:
    """Calculate Mahalanobis distances."""
    Z_centered = Z - np.mean(Z, axis=0)
```

```
    return np.sum(Z_centered @ cov_inv * Z_centered, axis=1)

def calculate_test_statistic(n: int, mahalanobis_distances: np.ndarray) -> np.ndarray:
    """Calculate the test statistic."""
    return ((n - 4) * n / ((n**2 - 1) * 4)) * mahalanobis_distances

def determine_outliers(projects: List[Project], alpha: float = 0.05) -> Tuple[List[int], List[Project]]:
    """Determine outliers using Mahalanobis distance."""
    Z = projects_to_array(projects)
    n = Z.shape[0]
    cov_inv = calculate_cov_inv(Z)
    mahalanobis_distances = calculate_mahalanobis_distances(Z, cov_inv)
    test_statistic = calculate_test_statistic(n, mahalanobis_distances)

    new_projects = []
    for i, project in enumerate(projects):
        new_projects.append(Project(
            url=project.url,
            x1=project.x1,
            x2=project.x2,
            x3=project.x3,
            y=project.y,
            zx1=project.zx1,
            zx2=project.zx2,
            zx3=project.zx3,
            zy=project.zy,
            ts=test_statistic[i],
            ds=mahalanobis_distances[i]
        ))


    fisher_f = f.ppf(1 - alpha, 4, n - 4)
    print(fisher_f)

    outliers = np.where(test_statistic > fisher_f)[0].tolist()
    for i in outliers:
        print(f"Removed project: Zy = {new_projects[i].zy:.4f} Zx1 = {new_projects[i].zx1:.4f} Zx2 =
{new_projects[i].zx2:.4f} Zx3 = {new_projects[i].zx3:.4f} (TS: {new_projects[i].ts:.4f})")

    return outliers, new_projects

def remove_outliers(projects: List[Project]) -> List[Project]:
    """Remove outliers"""
    outliers, projects = determine_outliers(projects)
    return [p for i, p in enumerate(projects) if i not in outliers]

def iterative_outlier_removal(projects: List[Project]) -> List[Project]:
    """Iteratively remove outliers until no more outliers are found."""
    iteration = 1
```

```
    while True:
        print(f"\nStarting iteration {iteration}")
        new_projects = remove_outliers(projects)

        if len(new_projects) == len(projects):
            print("No more outliers found. Stopping iterations.")
            break

        projects = new_projects
        iteration += 1

    return projects

def split_data(projects: List[Project], train_ratio: float = 0.6) -> Tuple[List[Project], List[Project]]:
    """
    Split the data into training and testing sets, ensuring that the training set
    includes the min and max values, and the testing set covers the remaining range.
    """
    # Sort projects by each metric
    sorted_by_cbo = sorted(projects, key=lambda p: p.x1)
    sorted_by_wmc = sorted(projects, key=lambda p: p.x2)
    sorted_by_sloc = sorted(projects, key=lambda p: p.y)
    sorted_by_noc = sorted(projects, key=lambda p: p.x3)

    # Get min and max projects for each metric
    min_max_projects = set([
        sorted_by_cbo[0], sorted_by_cbo[-1],
        sorted_by_wmc[0], sorted_by_wmc[-1],
        sorted_by_sloc[0], sorted_by_sloc[-1],
        sorted_by_noc[0], sorted_by_noc[-1]
    ])

    # Remove min and max projects from the main list
    remaining_projects = [p for p in projects if p not in min_max_projects]

    # Shuffle the remaining projects
    random.shuffle(remaining_projects)

    # Calculate the number of additional projects needed for the training set
    n_train = int(len(projects) * train_ratio) - len(min_max_projects)

    # Split the remaining projects
    train_projects = list(min_max_projects) + remaining_projects[:n_train]
    test_projects = remaining_projects[n_train:]

    return train_projects, test_projects

def save_to_csv(projects: List[Project], filename: str):
    """Save the list of projects to a CSV file."""
```

```
    df = pd.DataFrame([
        {
            'URL': p.url,
            'NC': p.x1,
            'NM': p.x2,
            'DIT': p.x3,
            'LOC': p.y,
            'ZX1': p.zx1,
            'ZX2': p.zx2,
            'ZX3': p.zx3,
            'ZY': p.zy
        } for p in projects
    ])
    df.to_csv(data_path(filename), index=False)

def main():
    projects = retrieve_data()
    print(f"Initial number of data points: {len(projects)}")

    normalized_projects = normalize_data(projects)
    final_projects = iterative_outlier_removal(normalized_projects)

    print(f"\nFinal number of data points after outlier removal: {len(final_projects)}")

    # Split the data into training and testing sets
    train_projects, test_projects = split_data(final_projects)

    print(f"\nNumber of training data points: {len(train_projects)}")
    print(f"Number of testing data points: {len(test_projects)}")

    # Save the training and testing sets to CSV files
    # save_to_csv(train_projects, 'train_data.csv')
    # save_to_csv(test_projects, 'test_data.csv')
    # print("\nData has been split and saved to 'train_data.csv' and 'test_data.csv'")

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

regression_model.dart

```
import 'dart:math';

import 'package:csv_metrics/constants.dart';
import 'package:csv_metrics/models/interval/model_interval.dart';
import 'package:csv_metrics/models/metrics/metrics.dart';
import 'package:csv_metrics/models/model_quality/model_quality.dart';
import 'package:csv_metrics/services/fisher.dart';
import 'package:csv_metrics/services/student.dart';
```

```
import 'package:flutter/material.dart';
import 'package:ml_linalg/linalg.dart';

class RegressionModel {
  RegressionModel(this.projects) {
    x1 = projects.map((e) => e.ncl!).toList();
    x2 = projects.map((e) => e.nm!).toList();
    x3 = projects.map((e) => e.dit!).toList();
    Y = projects.map((e) => e.loc!).toList();
    normalizeData();

  }

  void normalizeData() {
    Zx1 = List.filled(n, 0);
    Zx2 = List.filled(n, 0);
    Zx3 = List.filled(n, 0);
    Zy = List.filled(n, 0);

    // Применяем логарифмическую трансформацию по основанию 10
    for (var i = 0; i < projects.length; i++) {
      Zx1[i] = logBase10(x1[i]);
      Zx2[i] = logBase10(x2[i]);
      Zx3[i] = logBase10(x3[i]);
      Zy[i] = logBase10(Y[i]);
    }
  }

  final List <Metrics> projects;
  late final List<double> x1;
  late final List<double> x2;
  late final List<double> x3;
  late final List<double> Y;
  late List<double> Zx1;
  late List<double> Zx2;
  late List<double> Zx3;
  late List<double> Zy;

  double logBase10(double value) {
    if (value <= 0) return 0;
    return log(value) / log(10);
  }

  int get n => projects.length;
  double get x1Avg => x1.reduce((a, b) => a + b) / n;
  double get x2Avg => x2.reduce((a, b) => a + b) / n;
  double get x3Avg => x3.reduce((a, b) => a + b) / n;
  double get yAvg => Y.reduce((a, b) => a + b) / n;
  double get Zx1Avg => Zx1.reduce((a, b) => a + b) / n;
```

```
double get Zx2Avg => Zx2.reduce((a, b) => a + b) / n;
double get Zx3Avg => Zx3.reduce((a, b) => a + b) / n;
double get ZyAvg => Zy.reduce((a, b) => a + b) / n;

List<List<double>> calculateCovarianceMatrix() {
  List<List<double>> cov = List.generate(4, (_) => List.filled(4, 0));

  List<List<double>> values = [
    Zy,
    Zx1,
    Zx2,
    Zx3,
  ];

  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
      double sum = 0;
      for (int k = 0; k < n; k++) {
        sum += (values[i][k] - values[i].reduce((a, b) => a + b) / n) *
            (values[j][k] - values[j].reduce((a, b) => a + b) / n);
      }
      cov[i][j] = sum / n;
    }
  }

  return cov;
}

List<List<double>> invertMatrix(List<List<double>> matrix) {
  int n = matrix.length;

  List<List<double>> identityMatrix =
      List.generate(n, (i) => List.generate(n, (j) => i == j ? 1.0 : 0.0));

  List<List<double>> a = matrix.map((row) => row.toList()).toList();

  for (int i = 0; i < n; i++) {
    double diagElement = a[i][i];
    if (diagElement == 0) {
      throw Exception('Matrix is not invertible');
    }
    for (int j = 0; j < n; j++) {
      a[i][j] /= diagElement;
      identityMatrix[i][j] /= diagElement;
    }

    for (int k = 0; k < n; k++) {
      if (k == i) continue;
      double factor = a[k][i];
```

```
      for (int j = 0; j < n; j++) {
        a[k][j] -= factor * a[i][j];
        identityMatrix[k][j] -= factor * identityMatrix[i][j];
      }
    }
  }

  return identityMatrix;
}

List<double> calculateMahalanobisDistances(List<List<double>> covInv) {
  List<double> distances = List.filled(n, 0);

  List<List<double>> values = [
    Zy,
    Zx1,
    Zx2,
    Zx3,
  ];

  for (int i = 0; i < n; i++) {
    List<double> row = List.filled(4, 0);
    for (int j = 0; j < 4; j++) {
      for (int k = 0; k < 4; k++) {
        row[j] += covInv[j][k] *
            (values[k][i] - values[k].reduce((a, b) => a + b) / n);
      }
    }

    for (int j = 0; j < 4; j++) {
      distances[i] +=
          row[j] * (values[j][i] - values[j].reduce((a, b) => a + b) / n);
    }
  }

  return distances;
}

List<double> calculateTestStatistics(List<double> distances) {
  List<double> testStatistics = List.filled(n, 0);

  for (int i = 0; i < n; i++) {
    testStatistics[i] = (n - 4) * n / ((pow(n, 2) - 1) * 4) * distances[i];
  }

  return testStatistics;
}

Future<double?> calculateFisherFDistribution() {
```

```
  return Fisher.inv(
    alpha: alpha,
    df1: 4,
    df2: n - 4,
  );
}

Future<List<int>> determineOutliers(List<double> testStatistics) async {
  double? f = await calculateFisherFDistribution();
  if (f == null) {
    debugPrint('Failed to calculate Fisher F distribution');
    return [];
  }
  List<int> outliers = [];
  for (int i = 0; i < n; i++) {
    if (testStatistics[i] > f) {
      outliers.add(i);
    }
  }
  return outliers;
}

List<double> calculateRegressionCoefficients() {
  int n = Y.length;

  Matrix X = Matrix.fromRows([
    for (int i = 0; i < n; i++) Vector.fromList([1, Zx1[i], Zx2[i], Zx3[i]]),
  ]);

  Matrix yMatrix = Matrix.column(Zy);

  Matrix X_transpose = X.transpose();
  Matrix X_transpose_X = X_transpose * X;
  Matrix X_transpose_X_inv = X_transpose_X.inverse();
  Matrix X_transpose_y = X_transpose * yMatrix;
  Matrix b = X_transpose_X_inv * X_transpose_y;

  List<double> coefficients = b.getColumn(0).toList();

  return coefficients;
}

List<double> calculatePredictedValues(List<double> b) {
  double b0 = b[0];
  double b1 = b[1];
  double b2 = b[2];
  double b3 = b[3];

  List<double> Zy_hat = List.filled(n, 0);
```

```
    for (int i = 0; i < n; i++) {
      Zy_hat[i] = b0 + b1 * Zx1[i] + b2 * Zx2[i] + b3 * Zx3[i];
    }

    return Zy_hat;
  }

  List<double> calculateYHat(List<double> predictedValues) {

    return predictedValues
        .map((y) => pow(10, y))
        .toList()
        .cast<double>();
  }

  ModelQuality calculateModelQuality() {
    final YHatOriginal =
calculateYHat(calculatePredictedValues(calculateRegressionCoefficients()));
    final YOriginal = Zy.map((y) =>  pow(10,y)).toList();

    final residuals = List.generate(n, (i) => YOriginal[i] - YHatOriginal[i]);
    final meanY = YOriginal.reduce((a, b) => a + b) / n;

    final SSRes = residuals.map((r) => r * r).reduce((a, b) => a + b);
    final SSTot =
        YOriginal.map((y) => pow(y - meanY, 2)).reduce((a, b) => a + b);

    final rSquared = 1 - (SSRes / SSTot);
    final mmre = residuals
          .map((r) => r.abs() / YOriginal[residuals.indexOf(r)])
          .reduce((a, b) => a + b) /
        n;
    final pred = residuals
          .where((r) => r.abs() / YOriginal[residuals.indexOf(r)] < 0.25)
          .length /
        n;
    return ModelQuality(
      rSquared: rSquared,
      mmre: mmre,
      pred: pred,
    );
  }

  Future<List<ModelInterval>> calculateLinearIntervals() async {
    List<ModelInterval> intervals = [];

    final q = await Student.inv2T(alpha: alpha / 2, df: n - 4);
    if (q == null) {
      debugPrint('Failed to calculate Student T distribution');
```

```
  return [];
 }

 final ZyHat = calculatePredictedValues(calculateRegressionCoefficients());
 final sy = sqrt(
  (1 / (n - 4)) *
     Zy.map((value) => pow(value - ZyHat[Zy.indexOf(value)], 2))
        .reduce((value, element) => value + element),
 );

 for (int i = 0; i < n; i++) {
  final value = 1 / n +
     (pow(Zx1[i] - Zx1Avg, 2) +
           pow(Zx2[i] - Zx2Avg, 2) +
           pow(Zx3[i] - Zx3Avg, 2)) /
        (Zx1.map((value) => pow(value - Zx1Avg, 2))
              .reduce((value, element) => value + element) +
           Zx2.map((value) => pow(value - Zx2Avg, 2))
              .reduce((value, element) => value + element) +
           Zx3.map((value) => pow(value - Zx3Avg, 2))
              .reduce((value, element) => value + element));

  intervals.add(
    ModelInterval(
     index: i + 1,
     calculatedValues: ZyHat[i],
     lowerConfidenceLimit: ZyHat[i] - q * sy * sqrt(value),
     upperConfidenceLimit: ZyHat[i] + q * sy * sqrt(value),
     lowerPredictionLimit: ZyHat[i] - q * sy * sqrt(1 + value),
     upperPredictionLimit: ZyHat[i] + q * sy * sqrt(1 + value),
    ),
  );
 }
 return intervals;
}
Future<List<ModelInterval>> calculatenonLinearIntervals() async {
 List<ModelInterval> intervals = [];

 final q = await Student.inv2T(alpha: alpha / 2, df: n - 4);
 if (q == null) {
  debugPrint('Failed to calculate Student T distribution');
  return [];
 }

 final yHat = calculateYHat(
  calculatePredictedValues(calculateRegressionCoefficients()),
 );
 final sy = sqrt(
  (1 / (n - 4)) *
```

```
        Y
            .map((value) => pow(value - yHat[Y.indexOf(value)], 2))
            .reduce((value, element) => value + element),
    );

    for (int i = 0; i < n; i++) {
      final value = 1 / n +
          (pow(x1[i] - x1Avg, 2) +
                pow(x2[i] - x2Avg, 2) +
                pow(x3[i] - x3Avg, 2)) /
              (x1
                  .map((value) => pow(value - x1Avg, 2))
                  .reduce((value, element) => value + element) +
                x2
                  .map((value) => pow(value - x2Avg, 2))
                  .reduce((value, element) => value + element) +
                x3
                  .map((value) => pow(value - x3Avg, 2))
                  .reduce((value, element) => value + element));

      intervals.add(
        ModelInterval(
          index: i + 1,
          calculatedValues: yHat[i],
          lowerConfidenceLimit: yHat[i] - q * sy * sqrt(value),
          upperConfidenceLimit: yHat[i] + q * sy * sqrt(value),
          lowerPredictionLimit: yHat[i] - q * sy * sqrt(1 + value),
          upperPredictionLimit: yHat[i] + q * sy * sqrt(1 + value),
        ),
      );
    }
    return intervals;
  }
  num predictY(double x1, double x2, double x3) {
    final b = calculateRegressionCoefficients();
    return pow(10, b[0])*pow(x1, b[1])*pow(x2, b[2])*pow(x3, b[3]);
  }
}
```

ДОДАТОК В – ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «AppSight Analyzer»

Програмне забезпечення «AppSight Analyzer» розроблене для автоматизації процесу оцінювання розміру застосунків, які працюють на основі 2D графічних рушіїв. Цей інструмент дозволяє здійснювати очищення даних від викидів і створювати математичну модель, що використовується для прогнозування розміру програмного забезпечення.

Призначення та функціональні можливості:

1. Завантаження метрик.

Програма працює з файлами формату CSV, що містять значення метрик: NC (Number of Classes), NM (Number of Methods Per Class), DIT (Depth of Inheritance Tree) та KLOC (Thousands of Lines of Code). 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 виконує поетапне видалення аномальних спостережень із даних. Значення метрик піддаються нормалізації за допомогою десяткового логарифмування. На основі нормалізованих даних розраховуються квадрат відстані Махаланобіса і коваріаційна матриця. Для кожного спостереження обчислюється тестова статистика. Викиди визначаються шляхом порівняння отриманого значення T_S із критичним значенням $F_{крит}$. Цей процес повторюється до тих пір, поки всі викиди не будуть повністю видалені.

4. Побудова математичної моделі:

– Розрахунок параметрів моделі, коефіцієнтів b_0, b_1, b_2, b_3.

– Використання нелінійної регресії для прогнозування метрики KLOC на основі значень NC, NM та DIT.

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 розміру застосунків:

– Визначення розміру застосунків на основі побудованої математичної моделі.

7. Збереження результатів:

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

Мова розробки та технології:

«AppSight Analyzer» був повністю створений на основі платформи Flutter з використанням мови програмування Dart. Flutter забезпечує реалізацію інтерфейсу користувача та обробку даних в єдиній екосистемі, що дозволяє створювати кросплатформні рішення для мобільних пристроїв, десктопних систем і веб-додатків. Dart використовується для розробки всіх алгоритмів, включаючи обробку метрик, поділ даних на вибірки, побудову математичної моделі та перевірку 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.

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.

AppSight Analyzer є ефективним інструментом для розробників, який спрощує процес оцінювання розміру застосунків, що використовують 2D графічні рушії.

ДОДАТОК Г – ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «AppSight Analyzer»

1. Завантаження та встановлення

– Завантажте виконуваний файл AppSightAnalyzer.

– Розпакуйте архів та запустіть файл AppSightAnalyzer.exe

2. Завантаження даних

– Натисніть кнопку «Завантажити CSV файл» або Кнопку «+» нижче на 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.

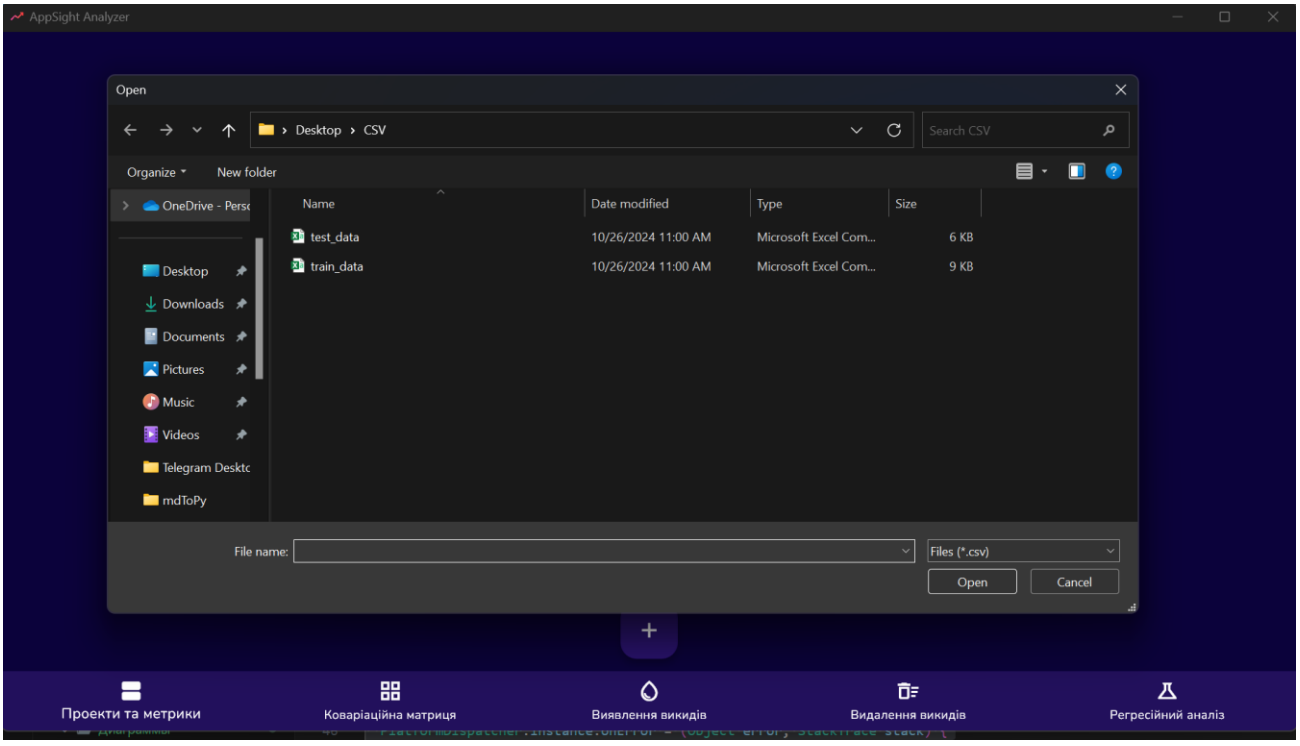


Рисунок Г.1 – Завантаження даних у «AppSight Analyzer»

3. Перегляд та аналіз даних

– Використовуйте меню, що знаходиться знизу, для переходу між різними вкладками з даними.

– Вкладка «Проекти та метрики» – перегляд завантажених проектів.

– Вкладка «Коваріаційна матриця» – перегляд коваріаційної матриці та обратної коваріаційної матриці.

– Вкладка «Виявлення викидів» – перегляд виявлених викидів.

– Вкладка «Видалення викидів» – видалення знайдених викидів.

– Вкладка «Регресійний аналіз» – прогнозування розміру за допомогою метрик NC, NM та DIT.

4. Використання побудованої моделі

– Перейдіть до вкладки «Регресійний аналіз».

– Введіть бажане значення NC в поле «Enter X1 (NC)».

– Введіть бажане значення NM в поле «Enter X2 (NM)».

– Введіть бажане значення DIT в поле «Enter X3 (DIT)».

– Змінюйте значення та переглядайте розмір.

На рисунку Г.2 представлено використання побудованої моделі.

Рисунок Г.2 – Використання побудованої моделі у «AppSight Analyzer»

ДОДАТОК Д – ПРОГРАМА І МЕТОДИКА ВИПРОБУВАНЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ «AppSight Analyzer»

1. Об’єкт випробування

Об'єктом випробування є програмне забезпечення «AppSight Analyzer», яке призначене для оцінювання розміру застосунків, що працюють на 2D графічних рушіях. Це програмне рішення застосовує математичну модель, яка використовує метрики NC, NM та DIT для прогнозування та аналізу обсягу програмного коду KLOC.

2. Мета випробування

Метою випробування програмного забезпечення «AppSight Analyzer» є перевірка правильності його функціонування, надійності математичної моделі, а також відповідності встановленим вимогам щодо продуктивності та стабільності. Основні 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

Під час випробування програмного забезпечення «AppSight Analyzer» функціональні характеристики перевіряються на відповідність вимогам, зазначеним у розділі «Вимоги до функціональних характеристик» технічного завдання, яке наведене в 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) Код програми.

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 «AppSight Analyzer» включає два етапи: перевірку відповідності програмної документації вимогам та оцінку відповідності функціональних характеристик програмного забезпечення.

6 Методи випробувань

6.1 Методика перевірки відповідності програмної документації встановленим вимогам.

Під час перевірки вимог до програмної документації потрібно здійснити аналіз наявності всіх необхідних документів і їх відповідності встановленим вимогам щодо змісту та формату. Додатково потрібно перевірити відповідність документів інформаційним вимогам, включаючи наявність усіх потрібних даних і детального опису програми, а також відповідність методики тестування

заявленим вимогам. Потрібно оцінити, чи інструкція користувача містить чіткі та зрозумілі інструкції щодо роботи з програмним забезпеченням. Для перевірки структури та оформлення документів потрібно звернути увагу на наявність заголовків, нумерації сторінок, змісту, а також потрібно оцінити читабельність тексту та його відповідність граматичним і орфографічним нормам. Усі документи потрібно перевірити на відповідність стандартам і забезпечення зрозумілої структури.

6.2 Методика перевірки відповідності функціональним характеристикам програмного продукту.

Під час перевірки програмного забезпечення «AppSight Analyzer», призначеного для оцінювання розміру застосунків на основі 2D графічних рушіїв, необхідно забезпечити виконання таких етапів:

– Завантаження метрик із файлу CSV. Перевірка коректності завантаження даних метрик (NC, NM, DIT, KLOC) з файлів формату CSV.

– Розподіл даних на навчальну та тестову вибірки. Тестування автоматичного розподілу завантажених даних на навчальну та тестову вибірки для побудови та перевірки моделі.

– Видалення викидів з вибірки. Оцінка ефективності ідентифікації та видалення аномальних значень із застосуванням відстані Махаланобіса та нормалізації.

– Побудова математичної моделі. Перевірка правильності розрахунку параметрів моделі (b_0, b_1, b_2, b_3) та прогнозування метрики KLOC на основі значень NC, NM та DIT.

– Тестування моделі на тестовій вибірці. Оцінка якості моделі за показниками R^2, *MMRE*, *PRED*(0,25) на основі тестової вибірки.

– Оцінювання розміру застосунків.

– Експорт результатів у форматі CSV. Випробування можливості експорту оброблених даних та результатів аналізу у форматі CSV з правильною структурою.

– Обробка некоректних даних. Перевірка стабільності роботи програми 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	2	3
1	Завантаження метрик із файлу CSV	Дані метрик (NC, NM, DIT, KLOC) 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, b_3) розраховано
5	Прогнозування значення KLOC на основі метрик NC,NM та DIT	Прогноз відповідає розрахунковим значенням
6	Тестування моделі на тестовій вибірці	Показники R^2, *MMRE*, *PRED*(0,25) відповідають очікуваним значенням
7	Експорт результатів у форматі CSV	Дані збережено у вказаному форматі, структура файлу відповідає вимогам
8	Обробка некоректних даних або порожнього файлу	Програма видає відповідне попередження або повідомлення про помилку