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.
«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти «магістр»

на тему: Математична модель для оцінювання складності об'єктно-орієнтованого проектування Java-застосунків з використанням метрик RFC та CBO та програма для її реалізації

Виконав: студент групи 6151м
_____ Бризгалов М.В.
(підпис, ПІБ)
Керівник роботи:
доцент кафедри ПЗАС, к.т.н, доцент.
(посада, науковий ступень вчене звання)
_____ Макарова Л.М.
(підпис, ПІБ)

Миколаїв – 2025 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)
« 27 » жовтня 2025 р.

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

Студенту Бризгалову Максиму Володимировичу
(Прізвище, ім'я, по батькові)

1. Тема роботи: Математична модель для оцінювання складності об'єктно-орієнтованого проектування Java-застосунків з використанням метрик RFC та CBO та програма для її реалізації

Керівник роботи доцент кафедри ПЗАС, к.т.н, доцент Макарова Л.М.

Затверджені наказом ректора № 1222-УЧ від « 15 » 10 2025 р.

2. Термін подання роботи: 08.12.2025 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 та методика випробувань програмного забезпечення)

5. Перелік презентаційних матеріалів Титульний слайд, Мета та завдання дослідження, Об'єкт та предмет дослідження, Наукова новизна та Практичне значення одержаних результатів, Аналіз існуючих моделей та методів, що застосовуються для для оцінювання складності об'єктно-орієнтованого проектування, Обґрунтування актуальності дослідження, Побудова математичної моделі для оцінювання складності об'єктно-орієнтованого проектування Java-застосунків з використанням метрик RFC та CBO, Постановка задачі на розробку програми для оцінювання складності об'єктно-орієнтованого проектування Java-застосунків. Діаграма варіантів використання, Технічний проєкт 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 27.10.2025 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	28.10.2025	ДП*
2.	Підготовка розділу з огляду літератури та обґрунтування необхідності проведення досліджень за обраною темою	30.10.2025	ДП*
3.	Підготовка розділу (ів) за результатами власних досліджень	01.11.2025	ДП*
4.	Підготовка розділу з розробки програмного забезпечення	26.11.2025	
5.	Підготовка організаційно-економічного розділу	28.11.2025	
6.	Підготовка розділу з охорони праці	02.12.2025	
7.	Підготовка розділу з охорони навколишнього середовища	03.12.2025	
8.	Підготовка розділу ВИСНОВКИ	04.12.2025	
9.	Оформлення списку використаних джерел та додатків	05.12.2025	
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.)	08.12.2025	не пізніше, ніж за 14 діб до захисту (згідно п.4.1 зазначеного Порядку)
11.	Підготовка презентації та доповіді	12.12.2025	
12.	Попередній захист роботи на засіданні кафедри ПЗАС	15.12.2025	
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	19.12.2025	

* - за результатами (розділ звіту) дослідницької практики (ДП), яка була з 01.09.2025 по 26.10.2025 р.

Студент _____ Бризгалов М.В.
(підпис) (ПІБ)

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

РЕФЕРАТ

Бризгалов Максим Володимирович

«Математична модель для оцінювання складності об’єктно-орієнтованого проектування Java-застосунків з використанням метрик RFC та CBO та програма для її реалізації»

Кваліфікаційна робота на здобуття освітнього рівня магістра зі спеціальності 121 «Інженерія програмного забезпечення». Національний університет кораблебудування імені адмірала Макарова. Миколаїв, 2025 р.

Обсяг роботи: 126 стор., 23 табл., 17 рис., 59 використане джерело, 5 додатків.

Актуальність теми роботи. У сучасних умовах розробки програмного забезпечення особливо важливим є своєчасне виявлення та аналіз відхилень у характеристиках програмних продуктів, що можуть свідчити про надмірну складність архітектури. Метрики RFC (Response for a Class) та CBO (Coupling Between Objects) є одними з ключових показників складності об’єктно-орієнтованих програм. Використання математичних моделей для виявлення викидів на основі цих метрик дозволяє відстежувати аномалії проєктування програмного забезпечення, що відповідає актуальним завданням інженерії ПЗ та сучасним вимогам до якості програмних продуктів.

Мета та завдання дослідження. Мета дослідження полягає у підвищенні достовірності оцінювання складності об’єктно-орієнтованого проектування Java-застосунків за рахунок побудови математичної моделі для прогнозування та розробці ПЗ для її реалізації, що забезпечить більш високу достовірність прогнозування складності на ранніх етапах розробки.

Об’єкт дослідження: Процес побудови математичної моделі у вигляді еліпса прогнозування для оцінювання складності об’єктно-орієнтованого проектування Java-застосунків з використанням метрик RFC та CBO.

Предмет дослідження: Математична модель у вигляді еліпса прогнозування для оцінювання складності об’єктно-орієнтованого проектування Java-застосунків з використанням метрик RFC та CBO.

Методи дослідження: Для вирішення визначених завдань у кваліфікаційній роботі використано методи математичної статистики та аналізу, теорії ймовірностей, об’єктно-орієнтованого проєктування. Ці методи застосовувались для побудови еліпса прогнозування для оцінювання складності об’єктно-орієнтованого проектування Java-застосунків з використанням метрик RFC та CBO.

Наукова новизна: Наукова новизна роботи полягає в удосконаленні математичної моделі на основі еліпса прогнозування для оцінювання складності об’єктно-орієнтованого проектування Java-застосунків з використанням метрик RFC та CBO, метода нормалізації Бокса-Кокса та видалення викидів із

використанням відстанні Махаланобіса та еліпсів прогнозування, що дозволяє підвищити достовірність та більш точно виявляти архітектурні аномалії.

Практичне значення одержаних результатів: Запропоноване програмне забезпечення забезпечує автоматизовану оцінку складності Java-застосунків. Його використання дозволяє зменшити ризики, пов'язані з надмірною архітектурною складністю, оптимізувати процеси проєктування і підтримки програмного забезпечення, а також підвищити ефективність роботи розподілених команд розробників.

Апробація результатів дослідження: Основні положення і результати досліджень пройшли апробацію на "Proceedings of the 13-th International Conference on Information Control Systems & Technologies (ICST 2025)" (24-26 вересня 2025р., м. Одеса) та три тези: VI Всеукраїнській науково-практичній інтернет-конференції "Інформаційні технології: моделі, алгоритми, системи (ITMAS-2025)" (16-17 листопада 2025 р., м. Миколаїв), матеріали XIII Міжнародної науково-практичної конференціі (24–26 вересень 2025 р., Одеса), матеріали IX Міжнародної науково-технічної конференції ком'ютерне моделювання та оптимизація складних систем, 2025, С.70 (24–26 вересень 2025 р., Одеса)

Публікації: За результатами досліджень опубліковано три тези та одна наукові праця, а саме матеріали конференції.

Ключові слова: Математична модель, виявлення викидів, програмна метрика, Java-застосунок, еліпс прогнозування, перетворення Бокса–Кокса, відстань Махаланобіса.

ABSTRACT

Bryzghalov Maksym Volodimirovich

«Mathematical model for estimating the complexity of object-oriented design of Java applications using the RFC and CBO metrics and a software for its implementation»

A Master's qualification work for obtaining the educational level of Master in specialty 121 "Software Engineering". Admiral Makarov National University of Shipbuilding. Mykolaiv, 2025.

Volume of work: 126 pages, 23 tables, 17 figures, 59 references, 5 appendices.

Relevance of the topic of the work: In modern software development, the timely detection and analysis of deviations in software product characteristics that may indicate excessive architectural complexity is particularly important. The metrics RFC (Response for a Class) and CBO (Coupling Between Objects) are among the key indicators of object-oriented software complexity. The use of mathematical models for outlier detection based on these metrics makes it possible to identify software design anomalies, which corresponds to current challenges in software engineering and modern quality requirements for software products.

Purpose and objectives of the study: The purpose of this study is to increase the reliability of assessing the complexity of object-oriented design of Java applications by constructing a mathematical model for prediction and developing software for its implementation, which will provide higher reliability of complexity prediction at the early stages of development.

The object of the study: The process of constructing a mathematical model in the form of a prediction ellipse for assessing the complexity of object-oriented design of Java applications using the RFC and CBO metrics.

Subject of the study: A mathematical model in the form of a prediction ellipse for assessing the complexity of object-oriented design of Java applications using the RFC and CBO metrics.

Methods of the study: To solve the defined tasks, the qualification work employs methods of mathematical statistics and analysis, probability theory, and object-oriented design. These methods were used to construct a prediction ellipse for assessing the complexity of object-oriented design of Java applications using the RFC and CBO metrics.

Scientific novelty of the results obtained: The scientific novelty of the work lies in the improvement of a mathematical model based on a prediction ellipse for assessing the complexity of object-oriented design of Java applications using the RFC and CBO metrics, the Box–Cox normalization method, and outlier removal using the Mahalanobis distance and prediction ellipses, which increases reliability and enables more accurate detection of architectural anomalies.

Practical significance of the obtained results: The proposed software provides automated complexity assessment of Java applications. Its use reduces risks associated

with excessive architectural complexity, optimizes software design and maintenance processes, and improves the efficiency of distributed development teams.

Testing of the research results: The main provisions and results of the study were approved and presented at the Proceedings of the 13th International Conference on Information Control Systems & Technologies (ICST 2025) (September 24–26, 2025, Odesa) and in three sets of theses: the VI All-Ukrainian Scientific and Practical Internet Conference “Information Technologies: Models, Algorithms, Systems (ITMAS-2025)” (November 16–17, 2025, Mykolaiv); the proceedings of the XIII International Scientific and Practical Conference (September 24–26, 2025, Odesa); and the proceedings of the IX International Scientific and Technical Conference “Computer Modeling and Optimization of Complex Systems”, 2025, p. 70 (September 24–26, 2025, Odesa).

Publications: Based on the research results, three conference abstracts and one scientific publication (conference proceedings) were published

Keywords: Mathematical model, outlier detection, software metric, Java application, prediction ellipse, Box–Cox transformation, Mahalanobis distance.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах динамічного розвитку інформаційних технологій та активного використання об’єктно-орієнтованих підходів у Java-програмуванні, розробники все частіше зіштовхуються з проблемою зростання складності програмних застосунків. Збільшення масштабів програмного коду та його взаємозв’язків створює потребу у більш точних методах аналізу та прогнозування архітектурних характеристик [1]. Важливу роль у цьому процесі відіграють метрики програмного забезпечення, зокрема ті, що відображають рівень зв’язності та відповідальності класів, оскільки саме вони найкраще характеризують складність структури застосунків [2].

Попри наявність значної кількості підходів до оцінювання якості проєктування ПЗ, сучасна інженерія програмного забезпечення все ще не має універсальної системи показників, яка б комплексно враховувала взаємозалежності між метриками. Це обмежує можливості точного прогнозування складності проєктів та своєчасного виявлення критичних аномалій у проєктуванні. Тому створення математичної моделі, здатної інтегрувати декілька ключових показників у єдину систему оцінювання, є важливим кроком до підвищення ефективності управління розробкою.

Додатковою проблемою стає те, що чимало програмних продуктів створюються у стислі терміни, з орієнтацією насамперед на виконання бізнес-завдань. У таких випадках вибір технологій часто ґрунтується не на технічному обґрунтуванні, а на зовнішніх факторах або попередньому досвіді команд [3]. Це може призвести до надмірного ускладнення системи, проблем із супроводом та підвищення витрат на подальшу підтримку. Особливої актуальності набуває ця проблема у відкритих проєктах, де до процесу розробки залучені розподілені команди з різних регіонів [4].

Сучасні методи аналізу складності програмного забезпечення пропонують різні моделі, але більшість з них недостатньо враховує багатофакторні залежності між метриками. Саме тому постає необхідність у вдосконаленні підходів до оцінювання складності об’єктно-орієнтованого проєктування, зокрема, таких, що ґрунтуються на побудові двох еліпсів прогнозування згідно із [5]. Це дає змогу не лише виявляти аномальні значення, але й класифікувати проєкти за рівнем складності. Реалізація такої моделі підвищить достовірність оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків.

Мета і завдання дослідження. Мета дослідження полягає у підвищенні достовірності оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO за рахунок побудови відповідної математичної моделі для прогнозування та розробці програми для її реалізації, що забезпечить більш високу достовірність прогнозування складності на ранніх етапах розробки.

Для досягнення цієї мети необхідно вирішити такі завдання:

– дослідити наявні методи оцінювання складності об’єктно-орієнтованого проєктування ПЗ;

– визначити ключові метрики для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків;

– дослідити відкриті джерела, написані мовою програмування Java, які можуть служити основою для створення моделі;

– побудувати математичну модель для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків;

– протестувати побудовану математичну модель для перевірки її адекватності;

– реалізувати програму для побудови математичної моделі у вигляді еліпсів прогнозування для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO;

– провести тестування розробленої програми з метою забезпечення достовірності прогнозування.

Об’єкт дослідження: процес побудови математичної моделі у вигляді еліпсів прогнозування для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO.

Предмет дослідження: математична модель у вигляді еліпсів прогнозування для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO.

Методи дослідження: Для вирішення визначених завдань у кваліфікаційній роботі використано методи математичної статистики, теорії ймовірностей, об’єктно-орієнтованого проєктування. Ці методи застосовувались для побудови еліпсів прогнозування для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO.

Наукова новизна: Наукова новизна роботи полягає в удосконаленні математичної моделі на основі еліпсів прогнозування для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO, за рахунок використання нормалізуючого перетворення Бокса-Кокса та видалення викидів з урахуванням квадрату відстані Махаланобіса, що дозволяє підвищити достовірність оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків та більш точно виявляти архітектурні аномалії порівняно з існуючими моделями.

Практичне значення одержаних результатів: Запропонована програма забезпечує автоматизоване оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків. Її використання дозволяє раннє виявленя ризиків, пов’язаних з надмірною архітектурною складністю, оптимізувати процеси проєктування і підтримки програмного забезпечення, а також підвищити ефективність роботи розподілених команд розробників.

Особистий внесок здобувача. Всі отримані результати є результатом самостійної роботи автора. У роботах, які були опубліковані у співавторстві, автором виконано: [6, 7] – збір та обробка метрик програмних застосунків на Java, пошук викидів; [8] – оцінювання складності на основі побудованої математичної моделі; [9] – розробка програми для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків.

Апробація результатів дослідження. Основні положення і результати досліджень пройшли апробацію на XIII Міжнародній науково-практичній конференції «Інформаційні управляючі системи і технології (ІУСТ-ОДЕСА-2025) » (24–26 вересень 2025 р., м. Одеса), IX Міжнародній науково-технічній конференції «Комп'ютерне моделювання та оптимізація складних систем (КМОСС 2025)» (5–7 листопада 2025 р., м. Дніпро), VI Всеукраїнській науково-практичній інтернет-конференції "Інформаційні технології: моделі, алгоритми, системи (ITMAS-2025)" (16-17 листопада 2025 р., м. Миколаїв).

Публікації: За результатами досліджень опубліковано чотири наукові праці, а саме: 1 – стаття, 3 – матеріали конференції.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА МОДЕЛЕЙ ДЛЯ ОЦІНЮВАННЯ СКЛАДНОСТІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОЄКТУВАННЯ

1.1 Аналіз структурних властивостей класів застосунків з відкритим кодом на Java

Аналіз структурних властивостей класів Java-застосунків у відкритих проєктах є важливим для розуміння їхньої внутрішньої організації та взаємозв'язків, що необхідно для побудови точних моделей оцінювання складності об'єктно-орієнтованого проєктування. У цьому контексті ключовими аспектами семантики класів є наступне.

Взаємозалежності класів. Дослідження зв'язків між класами та їх взаємодії дозволяє оцінити рівень зв'язності програмного продукту [10]. Побудова графів залежностей і аналіз внутрішніх взаємозв'язків допомагає виявляти критичні компоненти та потенційні вузькі місця архітектури.

Використання патернів проєктування. Виявлення застосованих дизайн-патернів дозволяє зрозуміти архітектурну структуру програми, визначити ступінь повторного використання коду та оцінити вплив патернів на складність програмного продукту [11]. Аналіз коду на предмет використання шаблонів проєктування, таких як Singleton, Factory або Strategy, сприяє точнішій оцінці архітектурної якості.

Абстракція та інтерфейси. Аналіз рівня абстрагування класів і застосування інтерфейсів показує гнучкість архітектури та можливості масштабування програмного продукту [12]. Використання інтерфейсів та абстрактних класів дозволяє створювати більш модульну та підтримувану програму, що зменшує залежність між компонентами.

Метадані та анотації. Вивчення наявності анотацій і метаданих у класах дозволяє враховувати додатковий функціонал, реалізований через рефлексію, і визначати потенційні точки складності в програми [13, 14]. Анотації можуть додавати поведінкові характеристики до класів та методів, впливаючи на складність тестування і підтримки коду.

Додатково, відкриті репозиторії часто містять компактні або демонстраційні проєкти з одним файлом або окремим модулем. Такі ресурси дозволяють швидко тестувати нові підходи, аналізувати ефективність алгоритмів і стиль програмування, а також формують емпіричну базу для побудови математичних моделей оцінювання складності проєктування. Ці невеликі проєкти слугують цінним джерелом даних для перевірки достовірності моделі.

Узагальнюючи, дослідження семантики класів у відкритих Java-проєктах дозволяє виділити основні фактори, які визначають складність об’єктно-орієнтованого проєктування, і забезпечує фундамент для подальшого формалізованого аналізу та математичного моделювання.

1.2 Аналіз моделей та метрик для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Оцінювання складності проєктування програмного забезпечення на ранніх етапах розробки є ключовим чинником для планування ресурсів, управління проєктом та прогнозування його ефективності. Метрики програмного забезпечення відображають кількісні характеристики структури та взаємодії компонентів програм і дозволяють порівнювати проєкти між собою, оцінювати трудовитрати, ймовірність появи помилок і загальну складність архітектури [15].

Будь-які контрольовані метричні характеристики мають аналізуватися у взаємозв’язку між собою або залежно від конкретного завдання. Гібридні

показники, що поєднують різні метрики, також залежать від базових метрик і не можуть бути універсальними.

Кількісні метрики. Спочатку застосовувалися для оцінки трудовитрат за проєктом, однак із розвитком мов програмування виникли проблеми через різницю між логічними та фізичними рядками коду. Логічні рядки відображають кількість команд у програмі і краще показують фактичну складність коду, хоча значною мірою залежать від мови програмування та стилю кодування [16]. Крім того, для оцінювання складності об’єктно-орієнтованих програм використовуються метрики класів, зокрема NCL (Number of Classes) – кількість класів у проєкті. Ця метрика дозволяє оцінити масштаб архітектури та рівень її складності, адже більша кількість класів часто свідчить про складніші взаємозв’язки між компонентами програми.

Об'єктно-орієнтовані метрики. Цей клас метрик виник разом із об'єктно-орієнтованим програмуванням. Найбільш відомими наборами є метрики Мартіна та метрики Чидамбера і Кемерера, до яких належать WMC, RFC, CBO та інші [1]. У контексті оцінювання складності проєктування Java-застосунків застосування метрик RFC та CBO у співвідношенні до кількості класів (NCL) дозволяє отримати відносні показники для побудови статистично обґрунтованих моделей складності.

Метрики надійності. Ці метрики враховують кількість помилок та дефектів у програмі, включаючи структурні зміни між перевірками, помилки під час перегляду коду та тестування. Для великих проєктів показники зазвичай нормалізуються на тисячу рядків коду, що дозволяє порівнювати рівень надійності різних програм [16].

Гібридні метрики. Застосовуються унікально під кожен проєкт та об’єднують різні показники для аналізу специфічних аспектів складності, наприклад, використання модулів монорепозиторія у великих програмах [17].

Розглянемо детальніше метрики об'єктно-орієнтованого класу, зокрема набір метрик Чідамбера і Кемерера. Основні метрики, що впливають на оцінку складності:

– Зважені методи класу (WMC). Ця метрика визначає складність класу шляхом підсумовування складностей усіх його методів. Має наступну формулу [18]:

$$WMC = \sum_{i=1}^{n} C_i,$$

де n – кількість методів у класі;

C_i – складність і-го методу.

Зі зростанням кількості методів у класі його застосування стає дедалі специфічнішим, що обмежує можливість багаторазового використання. Тому метрика WMC повинна мати низьке значення.

– Недолік зв'язності у методах (LCOM). Ця метрика показує, наскільки методи не пов'язані один з одним через властивості. Якщо всі методи звертаються до однакових властивостей, LCOM = 0 [19].

– Зв’язність між класами (CBO). Метрика відображає ступінь залежності компонентів програми та їх взаємодію, що прямо впливає на підтримуваність та розширюваність проєкту. CBO визначає кількість інших класів, з якими даний клас взаємодіє [20]:

$$CBO = |\{\text{кількість класів, які користуються цим класом}\}|.$$

Високе значення CBO свідчить про сильну залежність від інших класів, із чого виходить, що його складніше модифікувати.

– Відгук на класс (RFC). Показує кількість методів, які можуть бути викликані з боку класу, і використовується для оцінки потенційних точок

складності та ризику помилок. Для визначення загальної формули метрики необхідно привести поняття множини відгуку класу (RS), формула для визначення якого є [21]:

$$RS = \{M\}U_{all_i}\{R_i\},$$

де $\{R_i\}$ – множина методів, які викликає і-й метод;

$\{M\}$ – множина методів класу.

Метрика RFC дорівнює кількості методів у багатьох відгуках, тобто дорівнює потужності цієї множини. Формула метрики RFC [21]:

$$RFC = |RS|.$$

– NCL (Number of Classes in the System, кількість класів у програмі) – це метрика, яка відображає загальну кількість класів, що беруть участь у проєкті або модулі.

Сьогодні існує широкий спектр алгоритмічних підходів для оцінювання складності об’єктно-орієнтованого проєктування, що відрізняються використанням різних параметрів та методів аналізу. Серед них можна виділити регресійні, експертні, параметричні моделі та комбіновані підходи.

У межах цього дослідження пропонується використовувати математичну модель на основі трансформованого еліпса прогнозування, що дозволяє оцінювати складність об’єктно-орієнтованого проєктування Java-застосунків з урахуванням взаємозв’язку ключових метрик. Зокрема, застосовується відношення RFC/NCL та CBO/NCL, що забезпечує опосередковане вимірювання складності програми через нормалізовані показники класів.

На основі результатів дослідження, еліпс прогнозування ефективний для виявлення відхилень у багатовимірних даних, проте його класична реалізація

передбачає нормальний розподіл даних, який для метрик Java-застосунків не характерний [22, 23]. Тому запропонована модель використовує відносні метрики та нормалізацію Бокса-Кокса, яка трансформує вихідні дані та наближає їх до умов, необхідних для коректної побудови еліпса прогнозування.

1.3 Аналіз існуючих моделей оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків та обґрунтування необхідності дослідження

Оцінювання складності архітектури програмного забезпечення є важливим завданням, оскільки дозволяє виявляти потенційні аномалії, що впливають на підтримуваність, якість коду та ризики подальшого розвитку проєкту. Для цього застосовуються різні математичні та статистичні моделі, які дозволяють аналізувати зв’язки між ключовими метриками складності.

У цьому дослідженні було проаналізовано 302 Java-застосунки з відкритим кодом, на основі яких проводиться побудова трансформованого еліпса прогнозування. Планується використання відносного перетворення метрик RFC та CBO шляхом ділення на NCL для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків.

У цьому дослідженні розглядається математична модель на основі еліпса прогнозування, побудована за метриками RFC та CBO, що дозволяє виявляти випадки аномальної складності архітектури. Цей підхід дозволяє враховувати кореляцію між метриками та ефективно працює з двовимірними даними.

У той же час для порівняння ефективності даного підходу розглянемо подібну математичну модель. У статті [24] запропоновано удосконалений метод статистичної обробки двовимірних метрик, що базується на нормалізуючому перетворенні Бокса–Кокса та побудові еліпсів прогнозування на основі χ^2 та F-розподілів. Ця модель дозволяє надійно визначати аномальні значення,

враховуючи відхилення емпіричних даних від нормального розподілу, а також оцінювати ступінь поведінкової відмінності окремих елементів загальної вибірки.

Водночас модель, представлена в статті [24], не використовує відносне перетворення із застосуванням NCL, і у зв’язку з цим варіативність даних для моделей може бути дуже різний. Наприклад, у нашому випадку значення CBO та RFC без відносного перетворення можуть перевищувати 1000, що виходить за діапазон, для якого була побудована модель у статті [24]. Це свідчить про те, що існує сенс створити нову модель, адаптовану до наших даних.

2 МАТЕМАТИЧНА МОДЕЛЬ ДЛЯ ОЦІНЮВАННЯ СКЛАДНОСТІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОЄКТУВАННЯ JAVA-ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ МЕТРИК RFC ТА CBO

2.1 Математична модель для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків

Якщо вихідні дані для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків не відповідають нормальному розподілу, це ускладнює застосування відомих статистичних методів і може призводити до неточних результатів. Щоб підвищити достовірність аналізу, перед побудовою математичної моделі необхідно виконати нормалізацію даних.

У нашій моделі перед нормалізацією даних, метрики RFC та CBO перетворюються на відносні шляхом ділення на NCL, тобто дані мають вигляд RFC/NCL та CBO/NCL.

Використовувати модель з даними допустимо в обмежених випадках, а саме коли випадкові величини будуть мати гаусівський розподіл. У разі якщо дані не гаусівські потрібно застосувати нормалізуючі перетворення.

Для оцінки відхилення розподілу від нормального, використовуються показники багатовимірної асиметрії та ексцесу за Мардіа [25]:

$$\beta_{1,k} = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}[(X_i - \bar{X})^T S_N^{-1}(X_j - \bar{X})]^3, \quad (2.1)$$

$$\beta_{2,k} = \frac{1}{N}\sum_{i=1}^{N}[(X_i - \bar{X})^T S_N^{-1}(X_i - \bar{X})]^2, \quad (2.2)$$

де $X - k$-вимірний вектор змінних, $X = (X_1, X_2, \ldots, X_k)$;

$S_N -$ вибіркова коваріаційна матриця, яка обчислюється за формулою [26]

$$S_N = \frac{1}{N}\sum_{i=1}^{NN}(X_i - \bar{X})(X_i - \bar{X})^T, \quad (2.3)$$

де $\bar{X} -$ вектор середніх значень.

Для перевірки багатовимірної нормальності набору даних за тестом Мардіа використаємо два окремі критерії, що базуються на асиметрії та ексцесі. Обидві умови мають виконуватися, щоб припущення про багатовимірну нормальність вважалося справедливим. Тестова статистика для $\beta_{1,k}$ та $\beta_{2,k}$ визначається за співвідношенням:

$$\frac{n*\beta_{1,k}}{6} \leq \chi^2, \quad (2.4)$$

де χ^2 – це верхній квантиль χ^2-розподілу зі ступенями свободи $k(k+1)(k+2)/6$, а $\alpha = 0{,}005$ є рівнем значущості.

Для частини, що стосується ексцесу, статистика $\beta_{2,k}$ порівнюється з квантилем $1-\alpha$ нормального розподілу N із математичним сподіванням $\mu = k(k+2)$ та дисперсією $\sigma^2 = 8k(k+2)/N$:

$$\beta_{2,k} \leq N_{1-\alpha}(\mu, \sigma^2). \quad (2.5)$$

У цьому дослідженні нормалізація здійснюється за допомогою біваріантного нормалізуючого перетворення Бокса-Кокса для показників RFC/NCL та CBO/NCL, що дозволяє скоригувати початковий розподіл даних і підготувати їх для побудови трансформованого еліпса прогнозування [27]:

$$Z = \begin{cases} \frac{X^{\lambda}-1}{\lambda}, if\ \lambda \neq 0; \\ \log(X), if\ \lambda = 0, \end{cases} \quad (2.6)$$

де X приймає значення відповідних стовпців даних RFC/NCL та CBO/NCL;

Оптимальний вектор параметрів $\lambda=[\lambda_1, \lambda_2]$ оцінюється за допомогою методу максимальної правдоподібності (MLE), причому функція правдоподібності включає логарифм визначника коваріаційної матриці нормалізованих даних та перетворення Якобі [28]:

$$L(\lambda) = -\frac{n}{2} log * det\big(Cov(Z)\big) - \sum_{j=1}^{p}(\lambda_j - 1)\sum_{i=1}^{n} log\ (X_{ij}), \qquad (2.7)$$

де $L(\lambda) -$ функція правдоподібності;

$n -$ кількість спостережень;

$p -$ число змінних;

$Cov(Z) -$ коваріаційна матриця нормалізованих даних;

$X_{ij} -$ значення початкових даних для i-го спостереження таj-ї змінної.

Окрім припущень про розподіл, якість вибірки покращується шляхом видалення аномальних точок (викидів). Виявлення таких точок здійснюється за допомогою квадрату відстані Махаланобіса [29]:

$$d_i^2 = (Z_i - \bar{Z})S_Z^{-1}(Z_i - \bar{Z}), \qquad (2.8)$$

де $\bar{Z} -$ вектор середніх значень нормалізованих змінних $Z = (Z_1, Z_2, \ldots, Z_k)^T$;

$S_Z -$ зміщена вибіркова матриця коваріації для нормалізованих даних.

Статистичний поріг на основі F-розподілу для рівня довіри $1-\alpha$ обчислюється за формулою [30]:

$$T = \frac{k(n^2-1)}{n(n-k)} F_{1-\alpha}(k, n-k), \qquad (2.9)$$

де $n -$ число спостережень;

k – кількість змінних;

α – рівень значущості;

$F_{1-\alpha}$ – квантильна функція *F*-розподілу.

Якщо d_i^2, i=1,2,…,n перевищує статистичний поріг, точка вважається викидом.

Після нормалізації та видалення викидів для очищених даних будуються два багатовимірні еліпси прогнозування із різною точністю [5]. Математично еліпс прогнозування для нормалізованих даних описується формулою:

$$(X_i - \bar{X})^T S_N^{-1} (X_i - \bar{X}) = \chi^2_{k,\alpha} \quad (2.10)$$

або через *F*-розподіл:

$$(X_i - \bar{X})^T S_N^{-1} (X_i - \bar{X}) = \frac{k(n^2-1)}{n(n-k)} F_{1-\alpha}(k, n-k) \quad (2.11)$$

де X — вектор нормалізованих змінних;

$\bar{X}$ – вектор середніх значень;

S_N– вибіркова коваріаційна матриця, розрахована за формулою (2.3);

n – число спостережень;

k – кількість змінних;

α – рівень значущості.

Наступним кроком застосовується обернене перетворення, що дозволяє отримати два трансформованих еліпса прогнозування для початкових метрик RFC/NCL та CBO/NCL. Це забезпечує точне виявлення аномальних значень у даних (викидів), навіть якщо вони не відповідають гаусівському розподілу [7]:

$$\frac{[\psi_1(X_1)-m_{Z_1}]^2}{S_{Z_1}^2}+\frac{[\psi_2(X_2)-m_{Z_2}]^2}{S_{Z_2}^2}-\frac{2S_{Z_1Z_2}[\psi_1(X_1)-m_{Z_1}][\psi_2(X_2)-m_{Z_2}]}{S_{Z_1}^2S_{Z_2}^2}=\frac{2(N^2-1)(S_{Z_1}^2S_{Z_2}^2-S_{Z_1Z_2}^2)}{N(N-2)S_{Z_1}^2S_{Z_2}^2}F_{2,N-2,a}, \quad (2.12)$$

де $\psi -$ нормалізуюче перетворення за допомогою двовимірної нормалізації Бокса-Кокса;

$m_Z -$ вектор середніх значень $m_Z = (m_{Z_1}, m_{Z_2})^T -$ матриця коваріації;

$S_{Z_i}^2 -$ коваріаційна матриця.

Класифікація складності проєкту здійснюється на основі порівняння його відстані Махаланобіса з порогами отриманих еліпсів прогнозування.

Трансформовані еліпси прогнозування можна використовувати для аналізу нових проєктів, особливо у випадках появи викидів або якщо дані виходять за межі визначених регіонів еліпса.

Низька складність (Low) – проєкти, що розташовані всередині внутрішньої еліптичної області.

Висока складність (High) – проєкти, що розташовані між внутрішнім та зовнішнім еліпсами.

Викиди (Outlier) – проєкти, які не потрапляють у жодну з еліптичних областей, що вказує на аномальні або нестандартні показники метрик.

2.2 Попередня обробка даних для побудови математичної моделі для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Для побудови моделі автор зібрав набір даних метрик коду для 140 Java-застосунків ігрових рушіїв, розміщених на платформі GitHub [31]. Ці дані були отримані за допомогою статичного аналізу вихідного коду [32]. 88 метрик було взято з джерела [23] та 74 – із джерела [33]. Загалом, набір даних містив 302 рядки. До нього входять показники CBO/NCL та RFC/NCL. Ці метрики можуть бути

визначені вже на ранньому етапі планування проєкту на основі концептуальної моделі застосунку. Фрагмент із зібраних 140 проєктів наведено у таблиці 2.1, а описова статистика початкового набору даних наведена у таблиці 2.2.

Таблиця 2.1 – Фрагмент емпіричних даних зібраних із Github

Назва	CBO	RFC	NCL	CBO/NCL	RFC/NCL
2D-Game-Engine	85	241	24	3,5417	10,0417
2DGameEngineJava2D	55	244	18	3,0556	13,5556
2DGameEngine-Java	19	154	8	2,375	19,25
2dEngine	42	217	20	2,1	10,85
2dGameEngine	63	185	21	3	8,8095
2DGLEngine	25	91	13	1,9231	7
2dGameEngineJava	10	45	8	1,25	5,625
2dTilebased-Platforming-Engine	44	237	15	2,93331	15,8
2d-racing-game-engine	256	655	60	4,26617	10,9167
2D-Game-Enigine-	99	331	34	2,9118	9,7353
2D-Java-Platformer-Game	83	211	42	1,9762	5,0238
Ace-Game-Engine	189	327	55	3,4364	5,9455
2DShooterEngine	94	212	32	2,9375	6,625
2D_maze_game_engine_Java	15	184	8	1,875	23
2dGameEngine2007	17	110	13	1,3077	8,4615
2D-Platformer-Engine	52	208	25	2,08	8,32
APS-Engine2D	648	1798	118	5,49157	15,2373
Andreol-Engine	31	131	18	1,72227	7,2778

Таблиця 2.2 – Описова статистика початкового набору даних (N=302)

Назва метрики	Мінімальне	Максимальне	Середнє	Середньоквадратичне відхилення
NCL	4	7874	540,348	1020,819
CBO/NCL	0,200	22,417	5,806	3,690
RFC/NCL	1,325	37,278	12,648	5,728

Після перевірки двомірного набору даних CBO/NCL та RFC/NCL на відповідність нормальному розподілу згідно з (2.1), (2.2) було встановлено, що ці дані не мають ознак багатовимірної нормальності.

Значення багатовимірної асиметрії $\beta_{1,k}$=3,58, а відповідна тестова статистика за (2.4) дорівнює 180,61, яка значно перевищує критичне значення χ^2-розподілу 14,86 при 4 ступенях свободи та рівні значущості 0,005. Це свідчить про суттєву асиметрію розподілу, що не дозволяє вважати його нормальним за критерієм багатовимірної асиметрії.

Значення багатовимірного ексцесу $\beta_{2,k}$=13,86, а тестова статистика згідно з (2.5) становить 12,75, що також перевищує критичне значення 2,57, що вказує на надмірну піковість або плоскість розподілу.

У зв'язку з цим для приведення даних до нормального вигляду було здійснено нормалізацію за допомогою багатовимірного нормалізуючого перетворення Бокса–Кокса (2.6), яке вимагає, щоб усі значення були строго додатними. Оптимальні параметри λ за для кожної змінної були оцінені за методом максимального правдоподібності (2.7): $\lambda = [0{,}276, 0{,}227]$.

Під час нормалізації даних здійснювався ітеративний пошук та видалення викидів за допомогою квадрату відстані Махаланобіса та порогового значення статистичного тесту на основі *F*-розподілу, відповідно до (2.8) та (2.9):

$$D^2 \leq \frac{k(N^2-1)}{N(N-k)} F_{k,N-k,a} \approx 10{,}86.$$

Було ітеративно видалено чотири викиди, при цьому після кожного видалення перераховувалися параметри багатовимірного нормалізуючого перетворення Бокса–Кокса та знову обчислювалися квадрати відстані Махаланобіса.

Описова статистика очищеного набору даних наведена у таблиці 2.3.

Таблиця 2.3 – Описова статистика очищеного набору даних (N=298)

Назва метрики	Мінімальне	Максимальне	Середнє	Середньоквадратичне відхилення
NCL	4	7874	547,161	1025,952
CBO/NCL	0,696	22,417	5,865	3,678
RFC/NCL	3	37,278	12,636	5,609

Після видалення всіх викидів були визначені оптимальні значення λ: $\lambda = [0{,}235, 0{,}109]$. Значення багатовимірних асиметрії та ексцесу за Мардіа були обчислені на очищених даних за формулами (2.1) та (2.2) і дорівнюють 0,135 та 8,162 відповідно.

Тестові статистики для $\beta_{1,k}$ та $\beta_{2,k}$ знаходяться в межах критичних порогів, що вказує на відсутність суттєвого відхилення від багатовимірної нормальності згідно з порівнянням за формулами (2.4) та (2.5): $6{,}75 \leq 14{,}86,\ 0{,}351 \leq 2{,}57$.

2.3 Побудова математичної моделі для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO

Еліпс прогнозування для нормалізованих даних визначається наступним вектором середніх значень та матрицею коваріацій:

$$\bar{Z} = [1{,}98,\quad 2{,}82];$$
$$S_Z = \begin{bmatrix} 0{,}82 & 0{,}32 \\ 0{,}32 & 0{,}31 \end{bmatrix},$$

та має наступний вид згідно з (2.11):

$$\frac{(Z_1 - 1{,}98)^2}{0{,}82} + \frac{(Z_2 - 2{,}82)^2}{0{,}31} - \frac{0{,}64(Z_1 - 1{,}98)(Z_2 - 2{,}82)}{0{,}32} = 6{,}54.$$

За допомогою цього рівняння побудовано два еліпси прогнозування для нормалізованих даних за метриками RFC/NCL та CBO/NCL. Еліпс із α = 0,05 – межа низької складності. Еліпс із α = 0,005 – межа високої складності, дані поза цим межами вважаються викидами. Тестові дані Отримані еліпси прогнозування для нормалізованих даних наведено на рисунку 2.1.

Класифікація складності проєкту базується на порівнянні його квадрату відстані Махаланобіса з порогами $T_{0,05}$ та $T_{0,005}$. У нормалізованому просторі, якщо квадрат відстані Махаланобіса $d_i^2 \leq T_{0,05}$, тобто $d_i^2 \leq 6{,}09$, проєкт відноситься до низької складності, або нормальної зони. Якщо $T_{0,05} \leq d_i^2 \leq T_{0,005}$, тобто $6{,}09 \leq d_i^2 \leq 10{,}86$, проєкт потрапляє у граничну зону високої складності. Відстань, більша за $T_{0,005}$=10,86, визначає проєкт як викид або аномальний, тобто його складність не можна визначити, використовуючи дану модель.

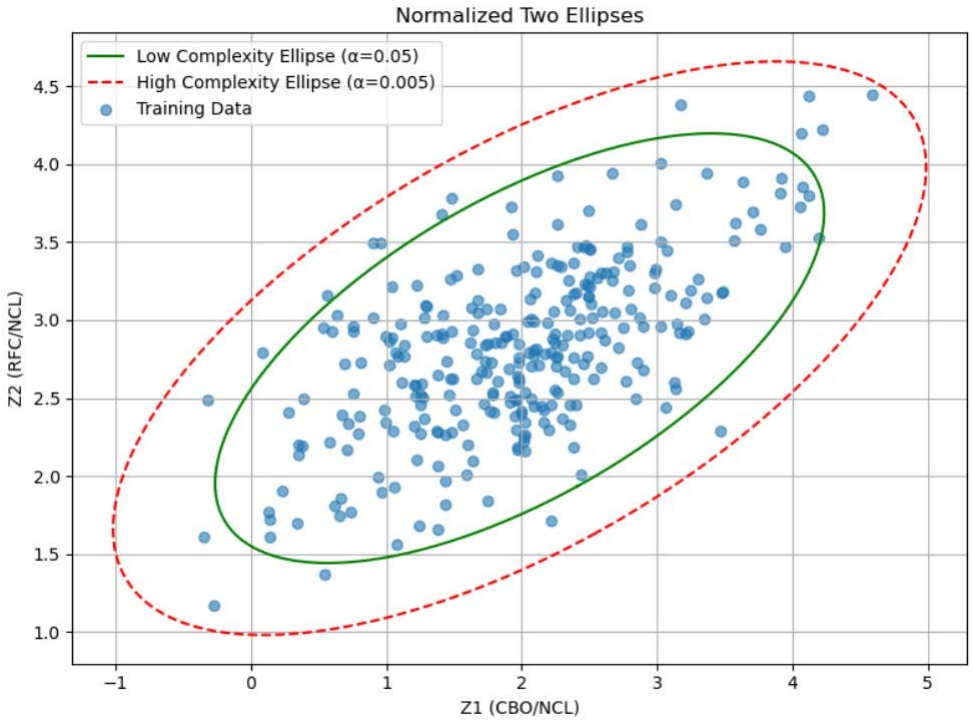

Рисунок 2.1 – Еліпси прогнозування для нормалізованих метрик RFC/NCL та CBO/NCL

Згідно з підходом, описаним у формулі (2.12), можна також побудувати трансформований еліпс прогнозування для початкових даних:

$$\frac{(\psi_1(X_1) - 1{,}98)^2}{0{,}82} + \frac{(\psi_2(X_2) - 2{,}82)^2}{0{,}31} - \frac{0{,}64(\psi_1(X_1) - 1{,}98)(\psi_2(X_2) - 2{,}82)}{0{,}32} = 6{,}54.$$

Отриманий трансформований еліпс прогнозування представлено на рисунку 2.2.

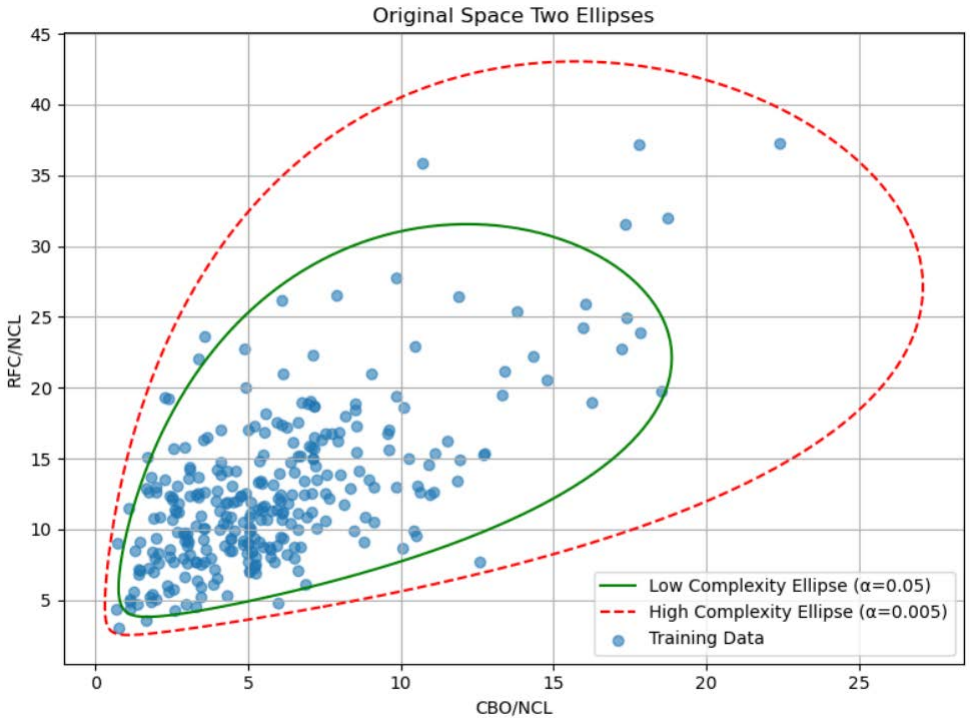

Рисунок 2.2 – Трансформовані еліпси прогнозування для метрик RFC/NCL та CBO/NCL

Використовуючи побудовану математичну модель для аналізу метрик CBO/NCL та RFC/NCL для 26 проєктів, які використовувалися як тестові, було проведено нормалізацію даних методом багатовимірного перетворення Бокса-Кокса та побудовано два еліпси для класифікації складності проєктів. Зібрані тестові дані із оцінюванням складності наведено в таблиці 2.4.

Таблиця 2.4 – Тестові емпіричні дані із оцінюванням складності

Назва	CBO	RFC	NCL	CBO/NCL	RFC/NCL	Conclusion
bookstore	92	210	26	3,538462	8,076923	Low
calculator-with-android-studio-Java	1	9	4	0,25	2,25	Outlier
cel-java	4947	12514	774	6,391473	16,16796	Low
Chat-App	84	331	52	1,615385	6,365385	Low
chess-ai	168	412	27	6,222222	15,25926	Low
glide-transformations	25	175	25	1	7	Low
glygen.cfde.generator	209	918	80	2,6125	11,475	Low
java-tor	75	250	31	2,419355	8,064516	Low
JavaAsciiGenerator	1	15	2	0,5	7,5	High
JavaSudokuSolver	2	32	3	0,666667	10,66667	High
kroxylicious	3262	9936	1006	3,242545	9,87674	Low
logger	48	149	14	3,428571	10,64286	Low
logstash	2894	7391	737	3,92673	10,02849	Low
lombok	8438	15491	3604	2,341287	4,29828	Low
mcav	1017	2385	305	3,334426	7,819672	Low
misle-java	611	1622	94	6,5	17,25532	Low
mockito	5988	12122	1825	3,281096	6,642192	Low
plantuml	30971	56076	3475	8,912518	16,13698	Low
presto	5182	16228	1186	4,369309	13,68297	Low
r2cloud	2319	7452	453	5,119205	16,45033	Low
recipe-app-java	30	51	14	2,142857	3,642857	High
Synapse	23	75	16	1,4375	4,6875	Low
TaskTracker	39	118	16	2,4375	7,375	Low
TextAnalyzr	1	19	2	0,5	9,5	Outlier
webmagic	1130	2614	309	3,656958	8,459547	Low
yp-online-store-showcase	270	359	79	3,417722	4,544304	Low

Аналіз результатів показав, що більшість проєктів (переважно від 1 до 17 за нормалізованими значеннями Z_1 та Z_2) класифікуються як Low, що свідчить про низьку складність об’єктно-орієнтованого проєктування цих застосунків. Проєкти, що потрапили до категорій High та Outlier, зазвичай мають відносно високі значення метрик або суттєво відрізняються від середнього профілю, що може вказувати на підвищену складність архітектури або наявність атипових класів.

Отримані результати наведено на рисунках 2.3 та 2.4.

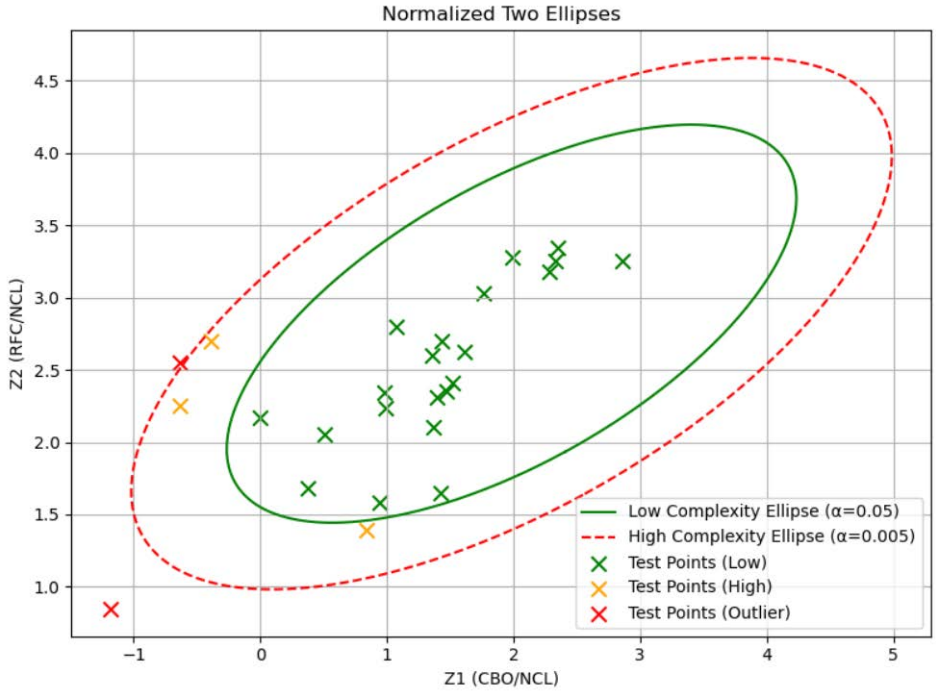

Рисунок 2.3 – Еліпси прогнозування із нормалізованими значеннями тестових даних

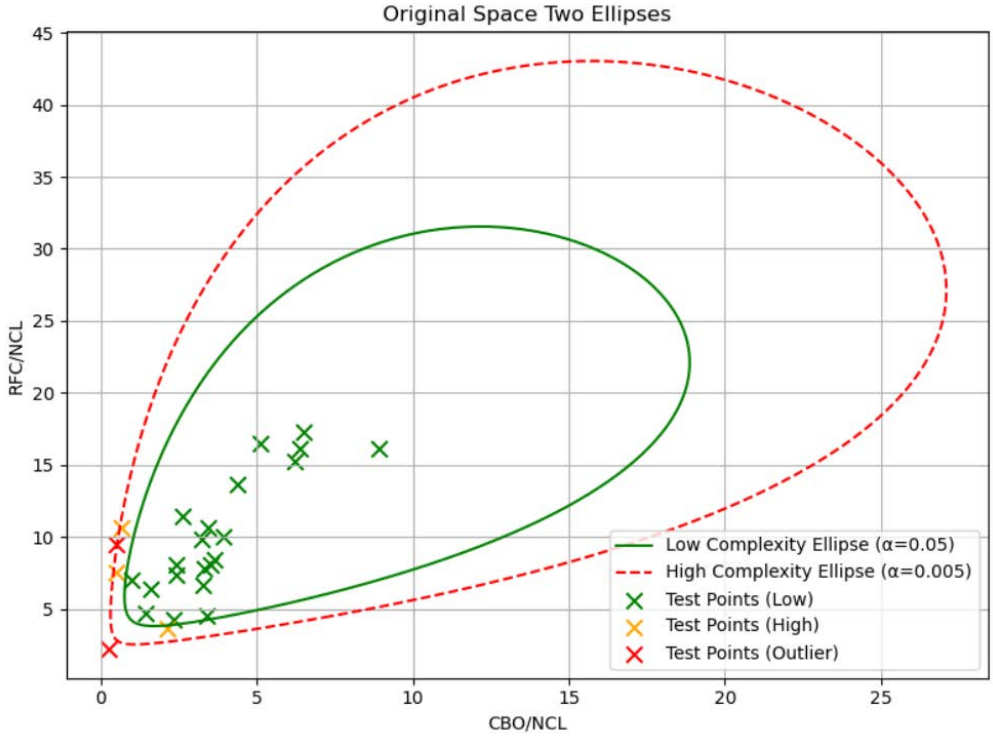

Рисунок 2.4 – Трансформовані еліпси прогнозування із значеннями тестових даних

Таким чином, побудована математична модель для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO, дозволяє не тільки виділити викиди та потенційно аномальні проєкти, а й оцінити загальний рівень складності для більшості застосунків, виявляючи тенденції та аномалії у метриках.

3 ПРОЄКТ ПРОГРАМИ ДЛЯ ОЦІНЮВАННЯ СКЛАДНОСТІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОЕКТУВАННЯ JAVA-ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ МЕТРИК RFC ТА CBO

3.1 Постановка задачі на розробку програми для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків

З метою створення інструменту для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків визначено задачу розробки програми, здатної виявляти проєкти з підвищеною архітектурною складністю на основі математичних і статистичних методів аналізу метрик об'єктно-орієнтованого програмування. Програма повинна здійснювати аналіз багатовимірних даних, що характеризують структуру програмних систем, з урахуванням їх статистичних властивостей, а також забезпечувати класифікацію проєктів за рівнем складності. Для цього використовуються такі ключові метрики: CBO/NCL – коефіцієнт зв'язності класів; RFC/NCL – кількість можливих відповідей класу.

На основі аналізу предметної області та сучасних підходів до статистичного контролю якості програмних систем було сформульовано вимоги до програми, яка отримало назву «Ellipse Predictor».

Програмний засіб «Ellipse Predictor» орієнтований на автоматизацію процесів: очищення даних від аномальних значень; перевірки статистичних припущень; побудови математичної моделі для оцінювання складності; класифікації програмних проєктів за рівнем складності.

Функціональність програмного засобу охоплює два ключові завдання:

– Побудова математичної моделі для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків.

– Оцінювання складності 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 математичної моделі для оцінювання складності

Обробка даних:

– Завантаження набору даних у форматі Excel, що містить метрики CBO/NCL та RFC/NCL;

– Автоматична фільтрація некоректних записів;

– Багатовимірна Box–Cox нормалізація;

– Ітеративне виявлення та видалення аномалій на основі відстані Махаланобіса та F-критерію;

– Формування журналів ітерацій у форматі Excel.

Побудова моделі:

– Обчислення коваріаційної та оберненої коваріаційної матриць. Формування математичної моделі для оціннювання складності;

– Побудова двох еліпсів: еліпс низької складності ($\alpha = 0{,}05$); еліпс високої складності ($\alpha = 0{,}005$);

– Збереження параметрів моделі в оперативній пам’яті програми.

Сценарій 2: Оцінювання складності на основі побудованої моделі

Введення даних:

– Введення або завантаження значень метрик CBO/NCL та RFC/NCL для аналізованого застосунку;

– Нормалізація введених значень з використанням параметрів навчальної вибірки.

Аналіз та класифікація:

– Обчислення відстані Махаланобіса до центру моделі;

– Перевірка належності точки до одного з еліпсів;

– Класифікація застосунку як: застосунок з низькою складністю; застосунок з високою складністю; аномальний застосунок (викид).

Виведення результатів:

– Відображення результатів класифікації у текстовому вигляді;

– Візуалізація положення точки відносно еліпсів у нормалізованому та трансформованому просторі метрик;

– Формування файлу з результатами класифікації тестових даних.

Вимоги до організації вхідних та вихідних даних

Програма повинна забезпечувати наступну організацію вхідних та вихідних даних:

1. Вхідні дані

Для побудови нової моделі та використання вже побудованої моделі:

– Набір даних у форматі Excel (.xlsx), що містить метрики CBO/NCL та RFC/NCL.

Для оцінювання складності:

– Значення метрик CBO/NCL та RFC/NCL для конкретного програмного проєкту або тестової вибірки.

2. Вихідні дані

Результатами роботи програмного засобу є:

– Параметри моделі еліпсу (середні значення, коваріаційні матриці, порогові значення);

– Очищена вибірка без викидів;

– Журнали ітерацій аналізу у форматі Excel;

– Класифікація проєктів за рівнем складності;

– Графічні візуалізації моделі та результатів аналізу.

Наведені вимоги формують основу для проєктування програмного засобу «Ellipse Predictor», який дозволяє підвищити ефективність аналізу оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків та забезпечує обґрунтовану оцінку архітектурних ризиків на ранніх етапах життєвого циклу програми.

3.2 Архітектура програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Архітектура розробленої програми побудована на основі шарової моделі, яка забезпечує чітке розділення відповідальностей між компонентами системи та сприяє її масштабованості, підтримуваності й повторному використанню коду [34]. Такий підхід є доцільним для прикладних аналітичних систем, що поєднують складну математичну обробку даних, візуалізацію результатів та інтерактивний графічний інтерфейс користувача.

Шарова архітектура передбачає організацію системи у вигляді логічно відокремлених рівнів, кожен з яких виконує визначене коло завдань і взаємодіє лише з сусідніми шарами. У межах розробленої програми можна виділити такі основні рівні.

Рівень представлення (Presentation Layer) :

Рівень представлення відповідає за взаємодію з користувачем та реалізує графічний інтерфейс застосунку.

Основні функції рівня представлення:

– забезпечення введення користувачем початкових даних;

– ініціація запуску аналітичних процедур;

– відображення результатів аналізу у вигляді текстових повідомлень, таблиць та графіків;

– передача команд і даних до нижчих рівнів без реалізації бізнес-логіки.

Рівень представлення не виконує обчислень і не містить алгоритмів аналізу, що забезпечує його незалежність від змін у внутрішній логіці системи.

Рівень прикладної логіки (Application / Business Logic Layer) :

Рівень прикладної логіки є ядром системи та відповідає за реалізацію основних алгоритмів оцінювання складності застосунків.

Рівень прикладної логіки є повністю незалежним від інтерфейсу користувача та механізмів візуалізації, що забезпечує можливість повторного використання обчислювального ядра в інших застосунках або середовищах.

Рівень роботи з даними (Persistence and Database Layer):

Рівень роботи з даними забезпечує зчитування, збереження та експорт інформації. Цей рівень не містить бізнес-логіки та використовується виключно для зберігання і передачі даних.

Взаємодія між рівнями здійснюється зверху вниз: рівень представлення ініціює запити до рівня прикладної логіки, який за потреби звертається до рівня роботи з даними та передає результати для візуалізації. Такий підхід забезпечує слабке зв’язування компонентів і високу модульність системи.

Для документування архітектури та відображення структури програми були використані UML-діаграми [35], створені за допомогою інструменту PlantUML [36]. На рисунку 3.1 наведено загальну діаграму компонентів та взаємодії програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків.

Таким чином, використання шарової архітектури дозволило створити гнучку, масштабовану та добре структуровану програму, придатне для подальшого розвитку й адаптації до нових вимог.

На рисунку 3.1 показано ланцюжок взаємодії між компонентами програми, призначеного для оцінювання розміру застосунків із використанням 2D графічних рушіїв.

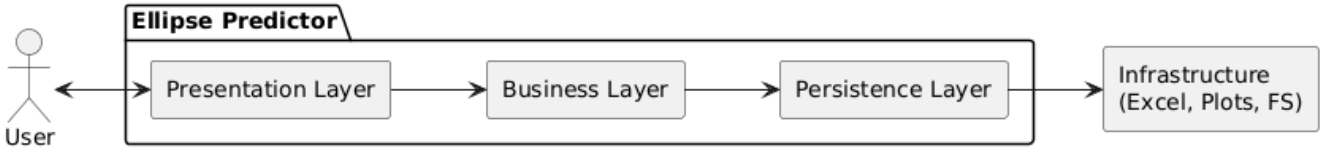

Рисунок 3.1 – Діаграма компонентів та взаємодії компонентів програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Ця схема демонструє архітектуру програми за моделлю Layered Architecture Pattern та її інтеграцію з локальною базою даних та файловою системою. Архітектура побудована у вигляді шарів з односпрямованими залежностями, що дозволяє чітко розділити відповідальності між компонентами системи.

Ця діаграма ілюструє архітектуру взаємодії компонентів у кілька рівнів:

1. Користувач (User) взаємодіє з UI Layer, який слугує інтерфейсом для введення даних і відображення результатів.

2. Presentation Layer передає введені дані Business Layer, який координує виконання сценаріїв аналізу та обробки даних.

3. Businees Layer передає Persistence Layer забезпечує роботу з постійними даними, такими як Excel-файли та файлову систем.

3.3 Розробка ескізного проекту програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

На основі аналізу вимог до програми був розроблений ескізний проєкт програми. Спочатку була побудована діаграма варіантів використання, яка

всебічно відображає основні функціональні сценарії та взаємодію користувачів з системним рішенням для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків [37].

У процесі побудови діаграми варіантів використання було визначено одного основного актора – Користувача, який взаємодіє із системою та виконує всі необхідні дії в межах програмного середовища. Саме він ініціює процеси, пов’язані з оцінюванням складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO.

Виявлені варіанти описані в таблиці 3.1., а візуальне представлення діаграми наведено на рисунку 3.2.

Таблиця 3.1 – Варіанти використання програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Код	Актори	Назва	Формулювання
1	2	3	4
P1	Користувач	Завантаження файлу із даними	Цей варіант використання дозволяє користувачу вибрати та завантажити Excel-файл із даними для подальшого аналізу.
P2	Користувач	Виконання аналіза файлу	Цей варіант використання дозволяє користувачу запустити аналіз даних, що включає нормалізацію та перевірку розподілу.
P3	Користувач	Перевірка нормальності розподілу даних	Цей варіант дозволяє користувачу перевірити, чи відповідають дані нормальному розподілу.
P4	Користувач	Нормалізація даних методом багатовимірного перетворення Бокса-Кокса	Цей варіант використання дозволяє користувачу застосувати багатовимірне перетворення Бокса-Кокса для нормалізації даних.
P5	Користувач	Перевірка та видалення викидів	Цей варіант використання дозволяє користувачу знайти та видалити викиди із вибірки.

Завершення таблиці 3.1

1	2	3	4
Р6	Користувач	Побудова моделей еліпсів	Цей варіант використання дозволяє користувачу побудувати моделі еліпсів на основі отриманих даних
Р7	Користувач	Відображення еліпсів у нормалізованому просторі	Цей варіант використання дозволяє користувачу візуалізувати моделі еліпсів для нормалізованих даних.
Р8	Користувач	Відображення трансформованих еліпсів	Цей варіант використання дозволяє користувачу візуалізувати моделі трансформованих еліпсів для даних після оберненої трансформації.
Р9	Користувач	Перевірка точки	Цей варіант використання дозволяє користувачу перевірити, чи належить введена точка до області еліпсів.
Р10	Користувач	Перевірка точки та відображення на просторі	Цей варіант використання дозволяє користувачу перевірити, чи належить введена точка до області еліпсів та відобразити її на просторі.
Р11	Користувач	Перевірка тестового набору	Цей варіант використання дозволяє користувачу завантажити тестові дані для оцінки моделі.
Р12	Користувач	Відображення тестових даних у нормалізованому просторі	Цей варіант використання дозволяє користувачу побачити розподіл тестових точок у нормалізованому просторі.
Р13	Користувач	Відображення тестових даних у трансформованому просторі	Цей варіант використання дозволяє користувачу переглянути, як тестові точки співвідносяться з оберненим еліпсом у трансформованому просторі.
Р14	Користувач	Експортування очищених даних	Цей варіант використання дозволяє користувачу зберегти оброблені та очищені дані у новий Excel-файл.

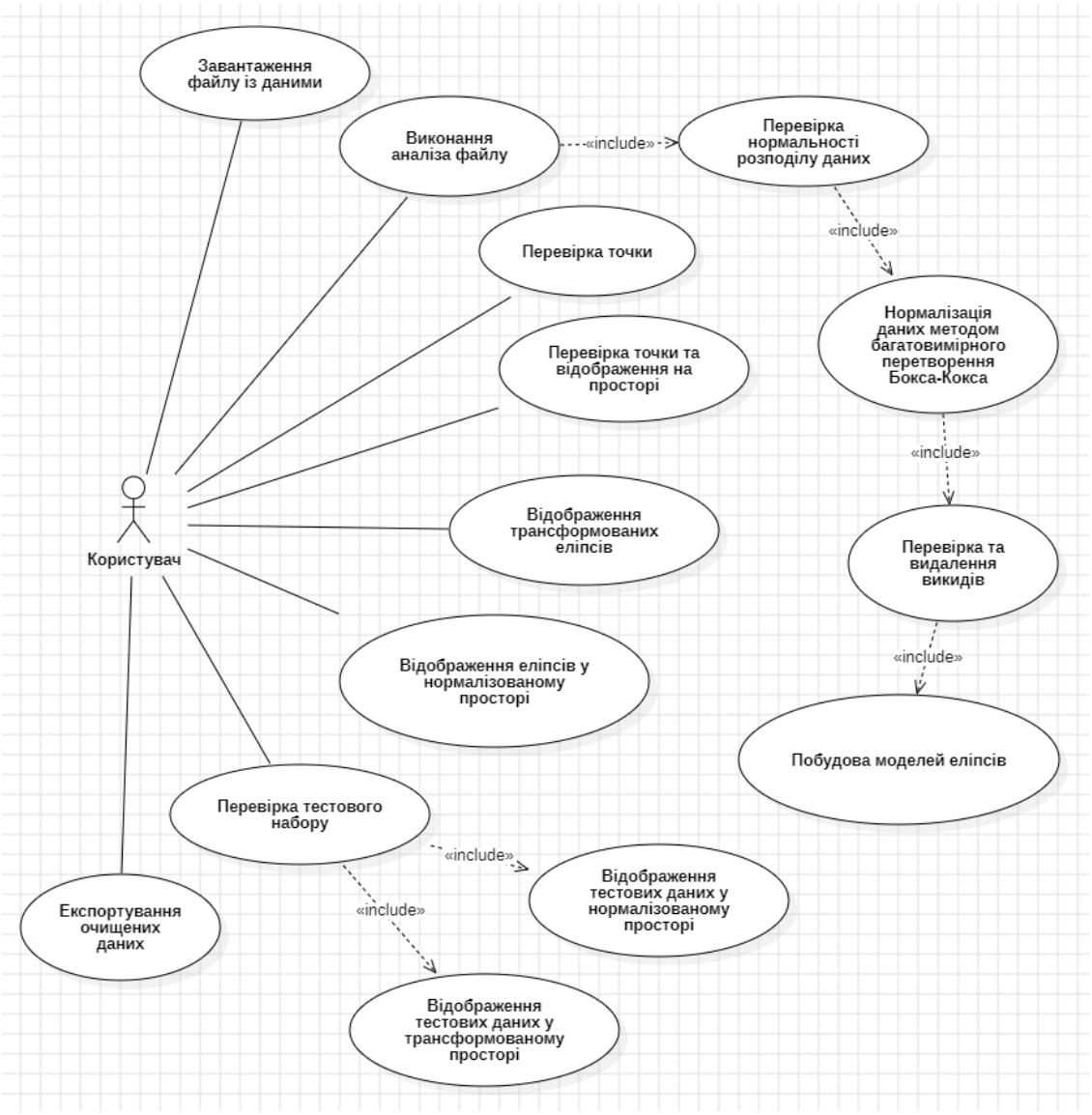

Рисунок 3.2 – Діаграма варіантів використання програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Створена діаграма варіантів використання є важливою основою для розробки системи. Наступним етапом є створення специфікації для виявлених варіантів використання.

Для кожного з раніше визначених варіантів використання слід створити окрему специфікацію. Кожна така специфікація включатиме: загальний опис, передумови, послідовність основних дій, можливі альтернативні сценарії та умови

завершення. Детальніші відомості про варіанти використання наведені в таблицях 3.2 – 3.14.

Таблиця 3.2 – Специфікація варіанту використання " Завантаження файлу із даними " (Р1)

Назва	Завантаження файлу із даними
Опис прецеденту	Цей варіант використання дозволяє користувачу вибрати та завантажити Excel-файл із даними для подальшого аналізу.
Дійові особи	Користувач
Зв'язок з іншими варіантами	–
Передумови	Користувач має доступ до програми та файл із даними.
Базовий потік	1. Користувач вибирає файл для завантаження. 2. Програма перевіряє формат файлу. 3. Програма підтверджує успішне завантаження файлу.
Альтернативні потоки	1. Якщо файл пошкоджений або має некоректний формат, програма видає повідомлення про помилку.
Постумови	Файл з даними успішно завантажений для аналізу.
Особливості	Підтримка різних форматів Excel-файлів.

Таблиця 3.3 – Специфікація варіанту використання "Виконання аналіза файлу" (Р2)

Назва	Виконання аналіз файлу
1	2
Опис прецеденту	Користувач може запустити аналіз даних, що включає нормалізацію та перевірку розподілу.
Дійові особи	Користувач
Зв'язок з іншими варіантами	Використовується після завантаження файлу.
Передумови	Файл із даними завантажено.
Базовий потік	1. Користувач запускає аналіз даних. 2. Програма перевіряє нормальність розподілу. 3. Виконується нормалізація даних методом Box-Cox. 4. Перевіряються та видаляються викиди.
Альтернативні потоки	1. Якщо дані некоректні або містять пропущені значення, програма видає повідомлення.

Завершення таблиці 3.3

1	2
Постумови	Дані проаналізовано та підготовлено для побудови еліпса.
Особливості	Можливість налаштування методів аналізу та нормалізації.

Таблиця 3.4 – Специфікація варіанту використання "Перевірка нормальності розподілу даних" (Р3)

Назва	Перевірка нормальності розподілу даних
Опис прецеденту	Перевірки нормальності розподілу даних
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується у складі аналізу файлу.
Передумови	Дані проєктів уже завантажені та підготовлені.
Базовий потік	1. Програма аналізує нормальність розподілу даних за тестом Мардіа. 2. Програма виводить результат
Альтернативні потоки	1. Якщо дані некоректні або містять пропущені значення, програма видає попередження.
Постумови	Користувач отримав інформацію про відповідність даних нормальному розподілу.

Таблиця 3.5 – Специфікація варіанту використання "Нормалізація даних методом багатовимірного перетворення Бокса-Кокса" (Р4)

Назва	Нормалізація даних методом багатовимірного перетворення Бокса-Кокса
1	2
Опис прецеденту	Користувач може застосувати багатовимірне перетворення Бокса-Кокса для нормалізації даних.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після перевірки нормальності розподілу.
Передумови	Дані завантажено та підготовлено до аналізу.
Базовий потік	1. Користувач запускає нормалізацію. 2. Програма обчислює перетворені значення. 3. Відображаються нормалізовані дані.

Завершення таблиці 3.5

1	2
Альтернативні потоки	1. Якщо дані містять некоректні значення, програма повідомляє про помилку.
Постумови	Дані успішно нормалізовані.

Таблиця 3.6 – Специфікація варіанту використання "Перевірка та видалення викидів" (Р5)

Назва	Перевірка та видалення викидів
Опис прецеденту	Користувач може знайти та видалити викиди із вибірки.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після нормалізації даних.
Передумови	Дані нормалізовані.
Базовий потік	1. Програма обчислює відстані Махаланобіса для точок. 2. Викиди виявляються та видаляються. 3. Зберігаються очищені дані.
Альтернативні потоки	1. Якщо викиди відсутні, користувачу повідомляється, що дані не містять аномалій.
Постумови	Дані очищені від викидів.

Таблиця 3.7 – Специфікація варіанту використання "Побудова моделей еліпсів" (Р6)

Назва	Побудова моделей еліпсів
1	2
Опис прецеденту	Користувач дозволяє користувачу побудувати моделей еліпсів на основі отриманих даних
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після перевірки та видалення викидів
Передумови	Дані очищені та нормалізовані.
Базовий потік	1. Програма будує моделі еліпсів прогнозування для отриманих даних. 2. Дані моделей зберігаются
Альтернативні потоки	1. Якщо дані містять некоректні значення, програма повідомляє про помилку.
Постумови	Побудована моделей еліпсів прогнозування.

Таблиця 3.8 – Специфікація варіанту використання "Відображення еліпсів у нормалізованому просторі" (Р7)

Назва	Відображення еліпсів у нормалізованому просторі
Опис прецеденту	Користувач візуалізує моделі еліпсів для нормалізованих даних, що допомагає оцінити розподіл і кореляції.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після нормалізації даних і видалення викидів (Р4, Р5).
Передумови	Дані нормалізовані та очищені від викидів.
Базовий потік	1. Користувач обирає відображення еліпсів. 2. Програма будує та відображає еліпси у нормалізованому просторі.
Альтернативні потоки	1. Якщо дані некоректні або порожні, програма видає повідомлення про помилку.
Постумови	Еліпси успішно відображені для аналізу нормалізованих даних.

Таблиця 3.9 – Специфікація варіанту використання "Відображення трансформованого еліпса" (Р8)

Назва	Відображення трансформованих еліпсів
Опис прецеденту	Користувач може переглянути еліпси у початковому просторі після оберненої трансформації нормалізованих даних.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після відображення еліпсів у нормалізованому просторі (Р7).
Передумови	Дані нормалізовані та очищені від викидів, еліпси у нормалізованому просторі побудований.
Базовий потік	1. Користувач обирає відображення трансформованих еліпсів. 2. Програма застосовує обернену трансформацію та відображає еліпси у початковому просторі.
Альтернативні потоки	1. Якщо дані некоректні або порожні, програма видає повідомлення про помилку.
Постумови	Трансформовані еліпси успішно відображено.

Таблиця 3.10 – Специфікація варіанту використання "Перевірка точки" (Р9)

Назва	Перевірка точки
Опис прецеденту	Користувач може визначити, чи належить введена точка до області еліпсів.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після побудови еліпсів у нормалізованому або початковому просторі (Р6, Р7).
Передумови	Еліпс побудований.
Базовий потік	1. Користувач вводить координати точки. 2. Програма перевіряє належність точки до еліпсів. 3. Відображається результат перевірки.
Альтернативні потоки	1. Якщо введені некоректні координати, програма видає попередження.
Постумови	Результат належності точки визначено.

Таблиця 3.11 – Специфікація варіанту використання "Перевірка точки та відображення на просторі" (Р10)

Назва	Перевірка точки та відображення на просторі
Опис прецеденту	Користувач може визначити, чи належить введена точка до області еліпсів та візуально відобразити її на просторі.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після побудови еліпсів у нормалізованому або початковому просторі (Р6, Р7).
Передумови	Еліпси побудовані.
Базовий потік	1. Користувач вводить координати точки. 2. Програма перевіряє належність точки до еліпса. 3. Відображається результат перевірки та точка на просторі.
Альтернативні потоки	1. Якщо введені некоректні координати, програма видає попередження.
Постумови	Результат належності точки визначено.

Таблиця 3.11 – Специфікація варіанту використання "Перевірка тестового набору" (Р11)

Назва	Перевірка тестового набору
Опис прецеденту	Користувач завантажує тестові дані для оцінки моделі та перевірки точності еліпсів.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Після побудови та відображення еліпса (Р6, Р7).
Передумови	Модель та еліпс побудовані, користувач має файл з тестовими даними.
Базовий потік	1. Користувач завантажує файл з тестовими даними. 2. Програма перевіряє формат і коректність даних. 3. Тестові дані зберігаються для подальшого аналізу.
Альтернативні потоки	1. Якщо файл некоректний, програма видає повідомлення про помилку.
Постумови	Тестовий набір підготовлено для аналізу.

Таблиця 3.12 – Специфікація варіанту використання "Відображення тестових даних у нормалізованому просторі" (Р12)

Назва	Відображення тестових даних у нормалізованому просторі
Опис прецеденту	Користувач може побачити розподіл тестових точок у нормалізованому просторі, щоб оцінити відповідність моделі.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після завантаження тестового набору (Р9) та побудови еліпса (Р6).
Передумови	Тестовий набір завантажено, еліпси у нормалізованому просторі побудовано.
Базовий потік	1. Користувач обирає відображення тестових даних. 2. Програма відображає точки на графіку із еліпсами.
Альтернативні потоки	1. Якщо дані некоректні, програма повідомляє користувача.
Постумови	Тестові дані успішно відображено у нормалізованому просторі.

Таблиця 3.13 – Специфікація варіанту використання "Відображення тестових даних у трансформованому просторі " (Р13)

Назва	Відображення тестових даних із трансформованим еліпсом
Опис прецеденту	Користувач може порівняти тестові точки із трансформованими еліпсами у початковому просторі.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після відображення трансформованих еліпсів (Р7) та завантаження тестових даних (Р9).
Передумови	Тестові дані завантажено; трансформовані еліпси побудовано.
Базовий потік	1. Користувач обирає відображення тестових даних із еліпсом. 2. Програма відображає точки разом із еліпсом у початковому просторі.
Альтернативні потоки	1. Якщо обернена трансформація даних неможлива, програма повідомляє про помилку.
Постумови	Тестові дані відображено та співвіднесено із еліпсами.

Таблиця 3.14 – Специфікація варіанту використання "Експортування очищених даних" (Р14)

Назва	Експортування очищених даних
Опис прецеденту	Користувач може зберегти оброблені та очищені дані у новий Excel-файл для подальшого аналізу.
Дійові особи	Розробник
Зв'язок з іншими варіантами	Використовується після видалення викидів (Р5) та нормалізації даних (Р4).
Передумови	Дані очищені від викидів та підготовлені для експорту.
Базовий потік	1. Користувач обирає команду експорту. 2. Програма формує Excel-файл із обробленими даними. 3. Користувач зберігає файл у вибраному місці.
Альтернативні потоки	1. Якщо збереження неможливе через помилки доступу, програма видає повідомлення.
Постумови	Очищені дані успішно збережено.

Створені специфікації використання є важливою деталізацією. Наступним етапом є побудова діаграми діяльності

Діаграми діяльності (Activity Diagrams) у мові UML застосовуються для візуалізації послідовності дій під час виконання певного процесу або операції. Вони мають спільні риси з діаграмами станів, проте відрізняються своєю логікою – тут основну увагу приділено моделюванню потоків робіт і взаємозв’язків між діями, а не фіксації конкретних станів системи [38].

Така діаграма показує, як система переходить від однієї діяльності до іншої, демонструючи залежності між діями, умови переходів та можливі паралельні гілки виконання. У більшості випадків стани на діаграмі відповідають певним видам діяльності, а переходи відбуваються після завершення поточної дії в початковому вузлі процесу. Діаграма діяльності варіантів використання програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків зображена на рис. 3.3.

Рисунок 3.3 – Діаграма діяльності варіантів використання програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Розроблена діаграма діяльності описує прцоесс загрузки файлів із даними, пеервірку на нормальність, нормалізацію за боксом коксом та ітеративне видалення викидів із повторною нормалізаціжю, після якоїх будуеться математична модель елпсів прогнозування.

Взаємодія між користувачем і програмою здійснюється за допомогою інтерфейсу користувача. Основне призначення цього інтерфейсу полягає у забезпеченні зручного та інтуїтивно зрозумілого зв'язку між користувачем і комп'ютером, мінімізації можливих помилок під час роботи з програмою та полегшенні сприйняття її основних можливостей.

Ефективність застосування функцій програми та комфорт користувача здебільшого визначаються якістю побудови інтерфейсу. Головними критеріями оцінювання користувацького інтерфейсу є його логічність, простота та доступність для розуміння. Ескіз форми представлено на рисунку 3.4.

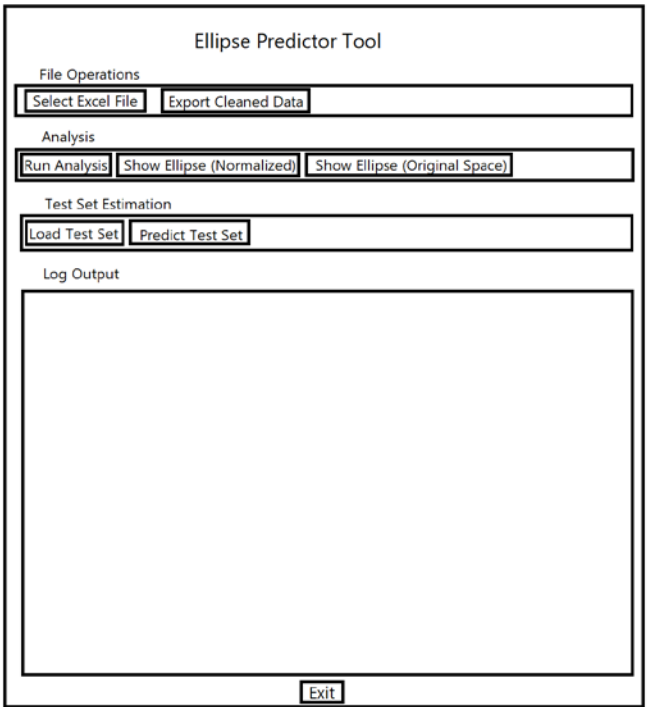

Рисунок 3.4 – Ескіз форми головного меню програми для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків

Розроблений прототип користувацького інтерфейсу повністю відповідає поставленим вимогам і забезпечує зручну та приємну взаємодію з програмою для користувачів.

3.4 Розробка техничного проекту програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Діаграма класів демонструє статичну структуру моделі, відображаючи основні декларативні елементи – класи, типи даних, їхній зміст та взаємозв’язки між ними. Вона забезпечує найбільш детальне уявлення про архітектуру програмного коду, взаємодію між компонентами, їхню функціональність та характеристики окремих класів [39].

Ця UML діаграма класів ілюструє систему, яка виконує нормалізацію даних, видалення викидів та побудову двох рівнів еліпсів для класифікації точок.

Опис класів

EllipseApp (UI): клас інтерфейсу користувача. Відповідає за завантаження даних, запуск аналізу, побудову еліпсів, класифікацію окремих точок та відображення результатів на графіку. Містить логування дій користувача та взаємодіє з усіма основними класами ядра.

OutlierDetector (Core): клас для виявлення та ітеративного видалення викидів. Використовує відстань Махаланобіса, коваріаційну матрицю та Box-Cox нормалізацію, а також логування результатів у Excel.

EllipseModel (Core): клас для побудови двох рівнів еліпсів (Low та High) та класифікації точок на основі цих еліпсів.

BoxCoxTransformer (Core): клас для нормалізації даних за методом Бокс-Кокса та зворотного перетворення, підтримує як унівaріантне, так і мультиваріантне застосування.

NormalityChecker (Core): клас для перевірки мультіваріантної нормальності даних за показниками асиметрії та куртозису (Mardia’s skewness & kurtosis).

ExcelLogger (Core): клас для логування дій в консолі або GUI, а також підсвічування викидів в Excel файлах.

plot_two_ellipses (Visualization): функція для візуалізації двох еліпсів на графіку, з можливістю відображення тренувальних та тестових точок у нормалізованих та оригінальних координатах.

Зв'язки між класами

Залежність: EllipseApp залежить від класів ядра (OutlierDetector, EllipseModel, BoxCoxTransformer, ExcelLogger) та модуля візуалізації (plot_two_ellipses), що означає їх використання для виконання основної логіки програми.

Агрегація: клас OutlierDetector використовує BoxCoxTransformer та ExcelLogger для нормалізації та логування ітерацій.

Асоціація: інші зв’язки відображають взаємодію між модулями, але не обов’язково означають володіння об’єктом.

Діаграма класів показує, як система забезпечує інтеграцію інтерфейсу користувача, ядра обробки даних та модулю візуалізації для побудови аналітичного процесу.

Процес роботи системи починається з вибору файлу користувачем через GUI. Далі дані нормалізуються та видаляються викиди за допомогою OutlierDetector. На очищених даних будується модель еліпсів через EllipseModel. Потім користувач може перевіряти окремі точки або тестовий набір, який відображається на графіках разом із класифікацією. Діаграма послідовностей ілюструє взаємодію користувача, GUI та ядра для виконання всіх етапів аналізу та візуалізації. Діаграма класів програми зображена на рисунку 3.5.

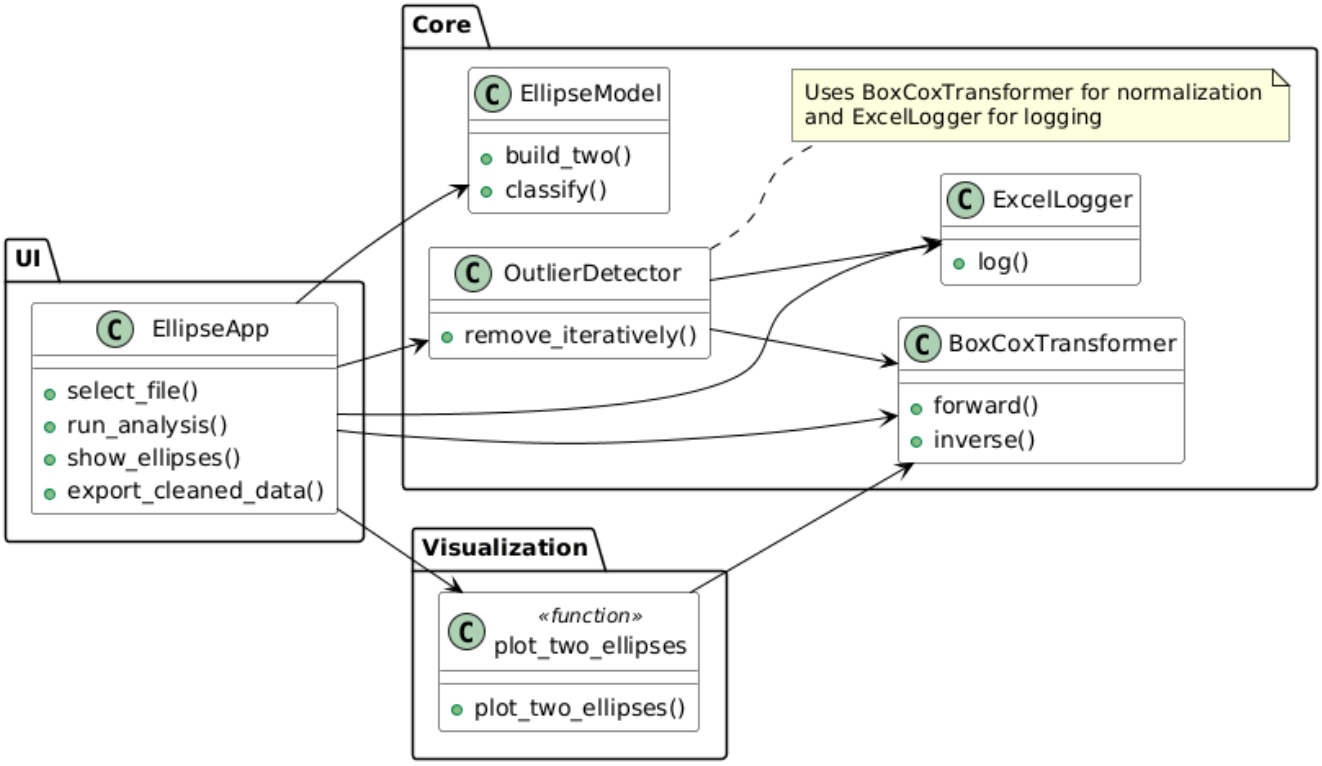

Рисунок 3.5 – Діаграма класів програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Розроблена модель описує базову архітектуру програми, визначаючи основні структурні елементи системи та принципи їхньої взаємодії

Діаграма послідовностей призначена для деталізації діаграм прецедентів та відображення поведінкових аспектів системи. Вона демонструє динамічну взаємодію об’єктів у часі, де обмін інформацією подається у вигляді послідовності повідомлень між ними в межах певного сценарію. Такий підхід дозволяє наочно простежити логіку виконання процесів, порядок викликів і взаємозалежності компонентів системи.

Приклад роботи програми зі створення моделі зображено на діаграмі послідовності на рисунку 3.6.

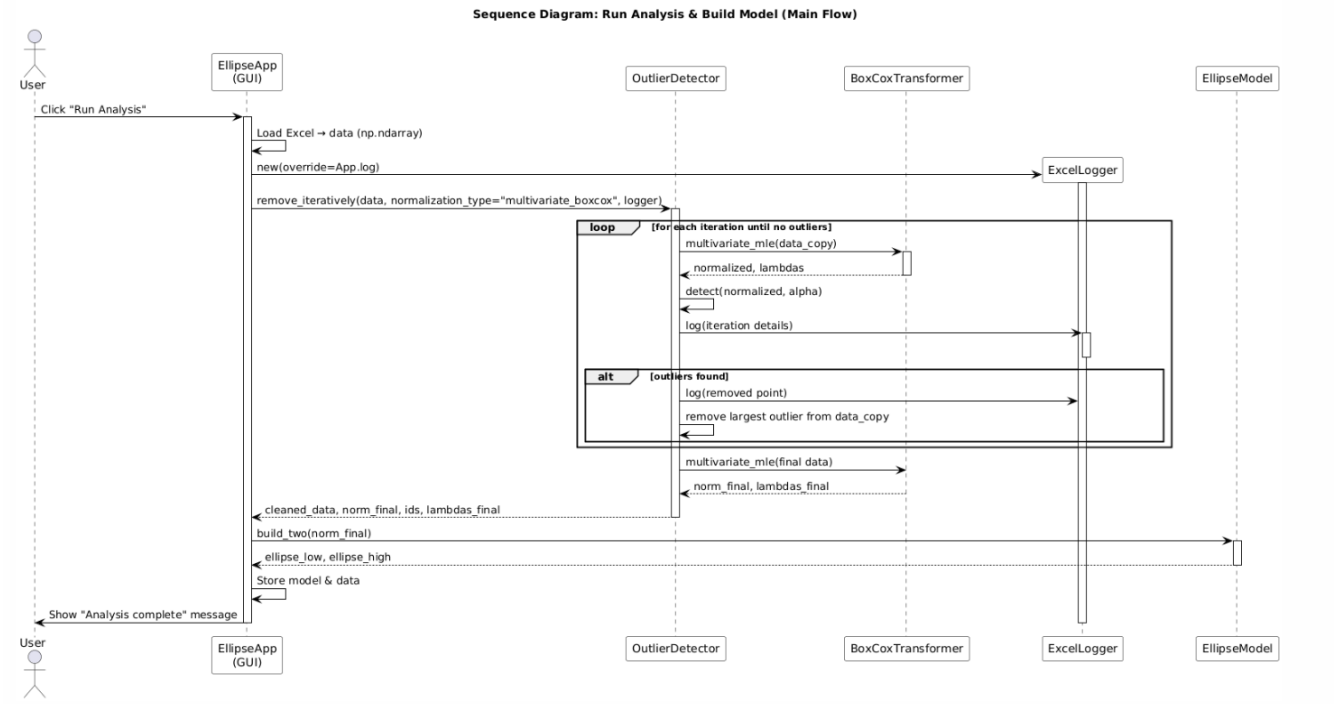

Рисунок 3.6 – Діаграма послідовності програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Діаграма послідовностей забезпечує глибше розуміння динаміки роботи програми та є важливим елементом його моделювання

3.5 Розробка робочого проекту програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Для розробки програми було використано мову програмування Python, яка є одним із найпопулярніших інструментів для наукових обчислень, обробки даних та побудови математичних моделей [40]. Python має широкий спектр вбудованих можливостей та зовнішніх бібліотек, що робить його універсальним середовищем для статистичного аналізу, регресійного моделювання та побудови графіків.

Для інтерактивної розробки, тестування алгоритмів та покрокового аналізу результатів було обрано інтегроване середовище розробки Spyder, яке входить до складу наукового дистрибутиву Anaconda [41, 42]. Spyder поєднує у собі

функціональність редактора коду, виконання команд у реальному часі та зручний перегляд змінних, що робить його ефективним інструментом для дослідницької роботи.

У процесі створення програми використовувалися бібліотеки NumPy (для високопродуктивних обчислень та роботи з масивами), SciPy (для реалізації методів оптимізації та нелінійної регресії), Matplotlib (для побудови графіків, зокрема відображення результатів моделювання та візуалізації нормальних еліпсів), Scikit-learn (для класифікації, кластеризації та роботи з відхиленнями (outliers)), Pandas (для структурованої роботи з даними) [43-47].

Python забезпечує високий рівень абстракції та дозволяє виконувати складні математичні розрахунки за допомогою лаконічного та зрозумілого коду. Використання наукових бібліотек значно пришвидшує реалізацію статистичних алгоритмів, зокрема нормалізації за Бокса–Кокса, обчислення довірчих еліпсів, аналізу відхилень та побудови моделей на основі двовимірних даних.

Окрім цього, Python надає можливість швидко будувати графічні інтерфейси, виконувати автоматизований аналіз даних та отримувати візуалізації, які легко інтегрувати у наукові роботи. Завдяки універсальності мови та стабільності наукової екосистеми Python є одним з найкращих виборів для реалізації складних математичних моделей та прикладних досліджень.

В результаті використання всіх зазначених технологій та інструментів, програми «Ellipse Predictor» готове до роботи. Воно надає користувачеві можливість завантажувати дані з метриками CBO/NCL та RFC/NCL, проводити нормалізацію даних, виявляти та видаляти викиди, а також будувати дворівневу еліптичну модель для класифікації точок (Low, High, Outlier) на основі відстані Махаланобіса.

Завдяки обраним технологіям, «Ellipse Predictor» є зручним, потужним та універсальним інструментом для аналізу даних та прогнозування рівня складності об’єктно-орієнтованих застосунків, що спрощує роботу розробників і забезпечує

високу продуктивність на всіх основних платформах. Головна сторінка застосунку представлена на рисунку 3.7.

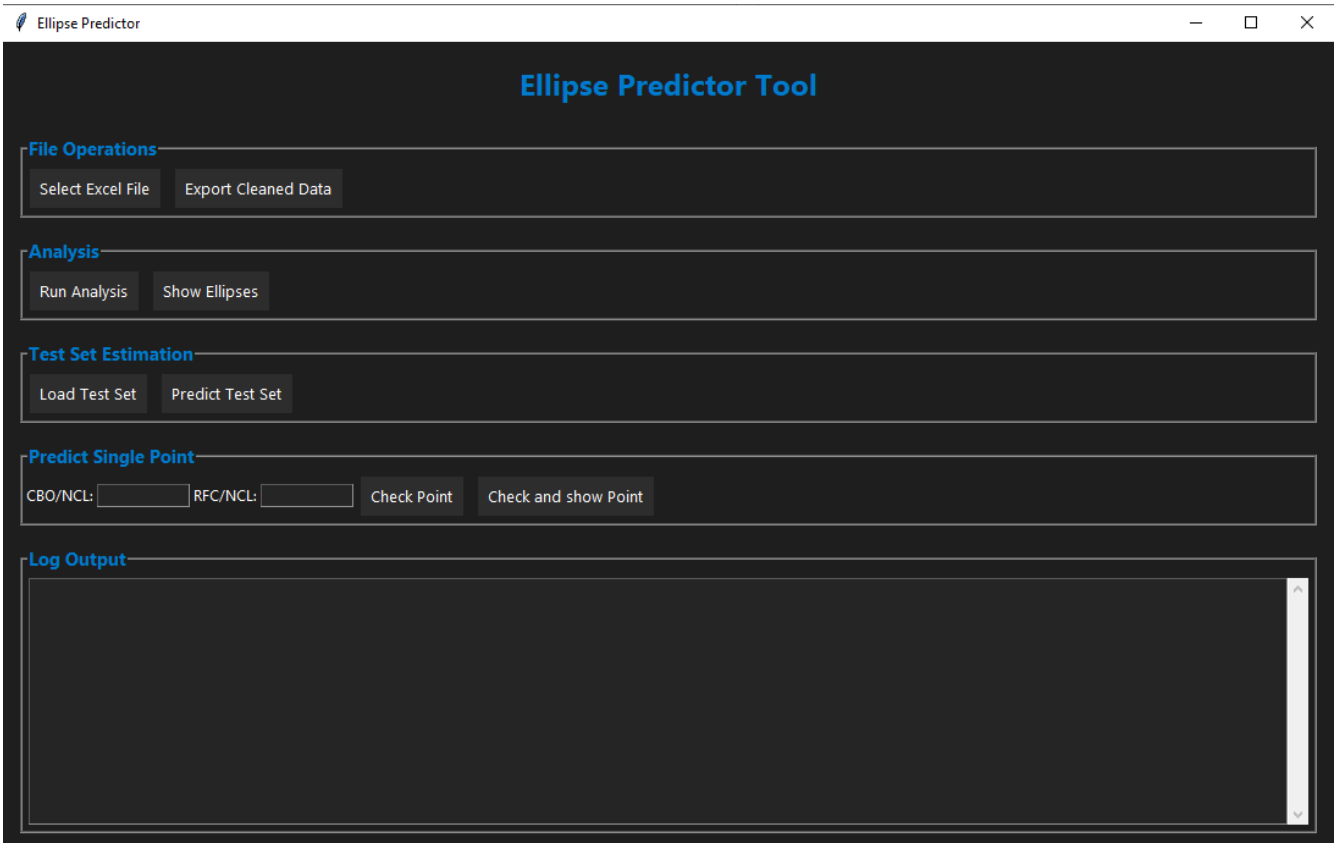

Рисунок 3.7 – Головна сторінка застосунку “Ellipse Predictor” для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

На наданому скріншоті зображено головну сторінку застосунку «Ellipse Predictor».

– Проекти та метрики – перегляд завантажених даних із метриками CBO/NCL та RFC/NCL.

– Аналіз – запуск обробки даних, включаючи нормалізацію та побудову еліпсів.

– Візуалізація еліпсів – відображення двох еліпсів, що відповідають категоріям Low та High, а також виділення точок-викидів.

– Тестовий набір / Прогнозування – завантаження тестового набору даних та класифікація точок за побудованою моделлю.

– Прогноз для однієї точки – введення значень метрик CBO/NCL та RFC/NCL для окремої точки з можливістю її класифікації та відображення на графіку еліпсів.

– Log Output – журнал обробки даних, де відображаються результати видалення викидів, нормалізації та класифікації точок.

Під час тестування програми «Ellipse Predictor» для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків, були проведені наступні тести для перевірки відповідності вимогам до функціональних характеристик:

– Завантаження даних з CSV файлу;

– Запуск аналізу даних;

– Побудова еліпсів;

– Перевірка тестових наборів;

– Перевірка одиничної точки.

Для оцінки якості програмного рішення проведено комплексне тестування його функціональних характеристик та продуктивності. Результати тестування представлено в таблиці 3.15.

Таблиця 3.15 – Результати тестування програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

№	Дія	Очікуваний результат	Результат
1	2	3	4
1	Завантаження даних з CSV файлу	Дані успішно завантажені.	Успішно
2	Запуск аналізу даних	Дані нормалізовані, викиди успішно виявляються та видаляються, результати відображаються у вкладці «Log Output» та вигружені до Excel.	Успішно

Завершення таблиці 3.15

1	2	3	4
3	Побудова еліпсів	Еліпси успішно побудовані, можуть бути відображені та результати вигружені до Excel.	Успішно
4	Перевірка тестових наборів	Прогнози успішно здійснюються, відображаються на графах та зберігаються у Excel.	Успішно
5	Перевірка одиничної точки	Прогноз успішно здійснюється, відображається на графах та зберігається у Excel.	Успішно

У ході тестування програми «Ellipse Predictor» всі функціональні вимоги були перевірені та підтверджені. Завантаження даних із CSV-файлу виконувалося коректно, виявлення та видалення викидів відбувалися без помилок, побудова моделі та прогнози були успішними.

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

Таким чином, програма «Ellipse Predictor» продемонструвало успішне проходження тестування та випробування і може бути ефективно використане для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВІД РОЗРОБКИ І ВПРОВАДЖЕННЯ ПРОГРАМИ ДЛЯ ОЦІНЮВАННЯ СКЛАДНОСТІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОЕКТУВАННЯ JAVA-ЗАСТОСУНКІВ З ВИКОРИСТАННЯМ МЕТРИК RFC ТА CBO

Дана кваліфікаційна робота спрямована на створення програмного засобу для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків.

Процес розробки програмного забезпечення передбачає одноразові витрати, пов'язані з його створенням і придбанням необхідних технічних засобів, а також поточні витрати, що виникають під час його експлуатації. Економічний ефект від використання програмного продукту визначається з урахуванням витрат на його функціонування, а співвідношення отриманої економії до витрат на розробку слугує показником економічної ефективності вкладених коштів. Розрахунок економічних показників здійснюється на основі оптових цін, тарифів і ставок оплати праці, чинних на момент проведення розрахунків.

У цьому розділі наведено розрахунок собівартості розробки програмного забезпечення, а також визначено показники економічної ефективності та строк його окупності [48].

4.1 Розрахунок витрат на створення й експлуатацію програми для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків

Витрати на розробку програмного забезпечення включають витрати на оплату праці розробника, експлуатацію та амортизацію комп'ютерної техніки, витрати на електроенергію, а також витрати на програмні й організаційні допоміжні ресурси, матеріали та комплектуючі.

Розробка програмного забезпечення здійснюється одним програмістом, місячна заробітна плата якого становить 30 000 грн. Балансова вартість персонального комп'ютера, що використовується у процесі розробки, складає 40 000 грн. Витрати на допоміжні матеріали наведені в таблиці 4.1.

Таблиця 4.1 – Витрати на допоміжні матеріали

№	Пункти витрат	Сума, грн.
1	Доступ до мережі Internet	300
2	Канцелярські матеріали	250
3	Обслуговування друкувального обладнання	320
4	Непередбачувані витрати	430
	Разом	1300

Вартість розробки програмного забезпечення розраховується за формулою:

$$К_{пр} = (В_{зп} + В_{есв} + В_{зг} + В_{е}) * Т + В_{дм}, \quad (4.1)$$

де: $В_{зп}$ – витрати на оплату праці, встановлені зарплатнею працівника, грн.,

$В_{есв}$ – єдиний соціальний внесок, що складає 38% від витрат на заробітну плату, грн.,

$В_{зг}$ – загальногосподарські витрати, що складають 10% від витрат на оплату праці без урахування податкового навантаження, грн.;

$В_{е}$ – витрати за електроенергію, грн.;

Т – тривалість розробки, міс.;

$В_{дм}$ – витрати на допоміжні матеріали, грн.

Розрахунок витрат на електроенергію виконується за умови вартості 1 кВт·год електроенергії 4,32 грн, середньої споживаної потужності комп'ютера 0,6 кВт та тривалості роботи 176 годин на місяць (22 робочі дні по 8 годин), розрахуємо значення B_e:

$$B_e = 176 \cdot 0{,}6 \cdot 4{,}32 = 456{,}19 \text{ грн.}$$

Витрати на розробку програмного забезпечення наведені в таблиці 4.2.

Таблиця 4.2 – Витрати на розробку програмного забезпечення

№	Найменування витрат	Одиниця	Кількість
1	Тривалість розробки	міс.	1
2	Основна і додаткова заробітна плата	грн.	30000
3	Відрахування в соціальні фонди	грн.	11400
4	Загальногосподарські витрати	грн.	3000
5	Витрати на допоміжні матеріали	грн.	1300
6	Витрати на електроенергію	грн.	456,19

Згідно з експертною оцінкою на основі аналізу аналогічних програмних продуктів встановлено, що тривалість розробки становить 1 місяць, а кількість задіяних спеціалістів – 1 особа. Тоді вартість розробки програмного забезпечення за формулою (4.1) дорівнює:

$$К_{пр} = (30000 + 11400 + 3000 + 456{,}19) \cdot 1 + 1300 = 46156{,}19 \text{ грн.}$$

Амортизаційні відрахування на устаткування складають 60% балансової вартості в рік:

$$А_{уст} = 40000 \cdot 0,5 = 20000 \text{ грн.}$$

У масштабах підприємства річні витрати на основні і допоміжні матеріали визначаються у розмірі 5% вартості основного устаткування:

$$В_{м} = 40000 \cdot 0,05 = 2000 \text{ грн.}$$

Оскільки в середньому устаткування споживає 0,12 кВт електроенергії і навантажується близько 6 годин на день, а в календарному році маємо 256 робочих днів, можемо розрахувати річне споживання електроенергії устаткуванням:

$$В_{е} = 256 \cdot 0,12 \cdot 6 \cdot 4,32 = 796,26 \text{ грн.}$$

Експлуатаційні витрати на устаткування на рік складуть:

$$В_{уст} = 20000 + 2000 + 796,26 = 22796,26 \text{ грн.}$$

Отже, у перший рік витрати на створення й експлуатацію програмного продукту складуть:

$$В_{р} = 46156,19 + 22796,26 = 68952,45 \text{грн.}$$

4.2 Економічна ефективність розробки та впровадження програми для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків

Основним показником економічної ефективності програмного забезпечення є підвищення результативності управління інформаційними процесами, що проявляється у зменшенні витрат на виконання управлінських операцій за одночасного зростання швидкості та якості отримання результатів.

Одним із ключових негативних чинників, усунення якого передбачається в результаті впровадження розроблюваного програмного забезпечення, є ручна обробка даних. Вона, окрім інших недоліків, призводить до значних економічних втрат, зокрема до витрат на зберігання паперової документації (шафи, архіви, папки), збільшення споживання канцелярських матеріалів, а також додаткових витрат, пов’язаних із пошуком, перевіркою та виправленням помилок, допущених під час обробки інформації.

До основних чинників зростання економічного ефекту від впровадження програмного забезпечення належать підвищення продуктивності праці та вивільнення робочого часу працівників. Водночас такі позитивні наслідки, як зростання оперативності управлінських рішень, підвищення якості результатів аналізу та покращення організації праці, не піддаються прямій грошовій оцінці, проте суттво впливають на загальну ефективність діяльності підприємства.

Необхідною умовою визначення економічної ефективності програмного забезпечення є забезпечення порівнянності всіх показників за часом, рівнем цін та нормативними параметрами, а також єдність підходів до їх розрахунку за складом і змістом витрат.

Визначення прямої економічної ефективності ґрунтується на експертній оцінці фахівців підприємства, відповідно до якої впровадження програмного забезпечення дозволяє вивільнити одного працівника.

Зарплата 1 працівника в рік складає:

(30000,00*12)*0,6 = 216000,00 грн.

Річний економічний ефект розраховується за наступною формулою:

$$\mathrm{E}_{\text{год}} = \Delta\mathrm{C}_n - E_n * k, \quad (4.2)$$

де $\Delta\mathrm{C}_n$– вивільнені кошти після впровадження програмного забезпечення (216000,00 грн) мінус експлуатаційні витрати (22796,26) = 193203,74 грн.;
E_n– коефіцієнт ефективності, який дорівнює коефіцієнту амортизації та має значення 0,6;
k – одноразові витрати на впровадження продукту (46156,19).

$$\mathrm{E}_{\text{год}} = 193203{,}74 - 0{,}5 \cdot 46156{,}19 = 193203{,}74 - 23078{,}10 = 170125{,}65 \text{ грн.}$$

Строк окупності програмного забезпечення розраховується по формулі:

$$\mathrm{T} = k/\Delta\mathrm{C}_n \text{ року.} \quad (4.3)$$

$$\mathrm{T} = 46156{,}19/193203{,}74 \approx 0{,}239 \text{ року.}$$

Отже, строк окупності розроблюваного програмного забезпечення становить приблизно 0,24 року, або близько 2,9 місяця, що доцільно округлити до 3 місяців.

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 [49]. 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 [50].

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 [51]. 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 [52]. 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 [53]. 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 [53]. 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 [54].

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 мову програмування Python, 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 [55].

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 [56]. 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 [57]. 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 [58]. 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 [59].

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було вирішено задачу побудови математичної моделі для оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO. Запропонована модель враховує взаємозалежність метрик програмного забезпечення та дозволяє виявляти архітектурні аномалії з більшою достовірністю.

У роботі здійснено аналіз сучасних способів оцінювання архітектури програм та визначено ключові метрики (RFC та CBO), що найбільш коректно відображають структурну складність Java-застосунків. Було досліджено відкриті програмні репозиторії та сформовано вибірки даних для подальшого моделювання.

На основі методів математичної статистики та теорії ймовірностей побудовано математичну модель у вигляді еліпсів прогнозування. Для підвищення достовірності оцінювання та усунення впливу нерівномірності даних використано нормалізуюче перетворення Бокса–Кокса. Видалення викидів виконано за допомогою квадрата відстані Махаланобіса, що забезпечило коректність форми та параметрів еліпсів прогнозування.

У межах дослідження розроблено програму, яка автоматизує побудову математичної моделі та дозволяє застосовувати її на практиці для оцінювання складності Java-застосунків. Тестування програмної реалізації підтвердило її коректність, придатність для реальних даних та практичну значущість.

У рамках виконання кваліфікаційної роботи були вирішені такі завдання:

– Досліджено сучасні методи оцінювання складності об’єктно-орієнтованого проєктування програмного забезпечення, проведено аналіз їхніх переваг і обмежень у контексті прогнозування архітектурних характеристик Java-застосунків;

– Визначено ключові метрики RFC та CBO, які найбільш повно характеризують структурну взаємодію класів і дозволяють оцінювати складність архітектурних рішень;

– Досліджено відкриті джерела Java-коду, сформовано вибірки метрик на основі даних з відкритих репозиторіїв, придатних для побудови математичної моделі;

– Побудовано математичну модель у вигляді еліпсів прогнозування, яка враховує багатофакторні залежності та дозволяє класифікувати складність програмних застосунків;

– Виконано тестування математичної моделі, що підтвердило її адекватність і здатність коректно виявляти аномальні значення метрик;

– Реалізовано програму для побудови математичної моделі, яка автоматизує процес оцінювання складності Java-застосунків на основі еліпсів прогнозування та метрик RFC і CBO;

– Проведено тестування розробленої програми, у результаті чого підтверджено її практичну придатність, достовірність прогнозування та можливість застосування на реальних проєктах.

Основні положення та результати роботи апробовані на міжнародних конференціях і опубліковані у вигляді наукових робіт. Розроблена модель та програмний інструмент можуть бути використані у подальших дослідженнях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. S.R. Chidamber, C.F. Kemerer. Towards a metrics suite for object oriented design. *ACM SIGPLAN Notices*, vol. 26, no. 11, 1991, p. 197–211. doi: 10.1145/118014.117970.

2. Y. Suresh, J. Pati, S. Ku Rath. Effectiveness of software metrics for object-oriented system. *Procedia Technology*, Volume 6, 2012, p. 420–427. doi: 10.1016/j.protcy.2012.10.050.

3. R.S. Pressman, B.R. Maxim. Software Engineering: A Practitioner's Approach, 8th edition. McGraw-Hill Education, 2014, 997 p. ISBN: 978-0078022128.

4. Sommerville I. Software Engineering. 10th edition. Pearson, 2015, 812 p.

5. Prykhodko A.S., Malakhov E.V. Determining object-oriented design complexity due to the identification of classes of open-source web applications created using PHP frameworks. *Radio Electronics, Computer Science, Control, no. 2, 2024.* doi:10.15588/1607-3274-2024-2-16

6. Prykhodko S., Makarova L., Latanska L., Bryzghalov M. Mathematical Model for Detecting Outliers in the Two-Dimensional Data of Software Metrics RFC and CBO from Applications in Java. *Proceedings of the 13th International Conference on Information Control Systems & Technologies (ICST)*, 2025. CEUR, Vol-4048, pp. 509-519. https://ceur-ws.org/Vol-4048/paper40.pdf

7. Приходько С., Макарова Л., Латанська Л., Бризгалов М. Виявлення викидів у двовимірних даних програмних метрик RFC та CBO для Java застосунків з використанням трансформованого еліпсу прогнозування. *Інформаційні управляючі системи і технології (ІУСТ-ОДЕСА-2025): матеріали XIII Міжнародної науково-практичної конференції (24–26 вересень 2025 р., Одеса)* / вип. ред. В.В. Вичужанін, 2025. С. 244-247.

8. Макарова Л. М., Бризгалов М. В. Оцінювання складності об’єктно-орієнтованого проєктування Java-застосунків з використанням метрик RFC та CBO. *Матеріали IX Міжнародної науково-технічної конференції «Комп’ютерне моделювання та оптимізація складних систем» (КМОСС-2025)*, (5–7 листопада 2025 р. м. Дніпро), С. 70.

9. Бризгалов М.В., Макарова Л.М. Розробка програми для оцінювання складності об’єктно-орієнтованого проєктування Java застосунків з використанням метрик RFC та CBO. *Інформаційні технології: моделі, алгоритми, системи* (ITMAS – 2025): *Матеріали VI Міжнародної науково-практичної інтернет конференції* (16-17 листопада 2025 р.). Миколаїв: НУК імені адмірала Макарова, 2025. С. 270-272.

10. DigitalOcean. Most Common Design Patterns in Java (with Examples). URL: https://www.digitalocean.com/community/tutorials/java-design-patterns-example-tutorial (дата звернення: 10.10.2025).

11. DigitalOcean. Abstract Class in Java. URL: https://www.digitalocean.com/community/tutorials/abstract-class-in-java (дата звернення: 10.10.2025).

12. Medium. Java Annotations: A Comprehensive Guide. URL: https://anandb3.medium.com/java-annotations-a-comprehensive-guide-1d01352a56d4 (дата звернення: 10.10.2025).

13. Oracle. Making the Most of Java's Metadata, Part 3. URL: https://www.oracle.com/technical-resources/articles/hunter-meta2.html (дата звернення: 12.10.2025).

14. A.-J. Molnar, A. Neamţu, S. Motogn. Evaluation of Software Product Quality Metrics, *CCIS*, vol. 1172, 2020, p. 163–187. doi: 10.1007/978-3-030-40223-5_8.

15. Lines of Code as a Software Metric, GetDX. URL: https://linearb.io/blog/lines-of-code (дата звернення: 12.10.2025).

16. Essential Reliability Metrics for Software Quality. URL: https://signoz.io/guides/reliability-metrics (дата звернення: 12.10.2025).

17. Mono-repo vs Multi-repo vs Hybrid: What’s the Right Approach? URL: https://medium.com/outbrain-engineering/mono-repo-vs-multi-repo-vs-hybrid-whats-the-right-approach-5436c575c6e0 (дата звернення: 12.10.2025).

18. Weighted Methods Per Class. URL: https://dcm.dev/docs/metrics/class/weighted-methods-per-class (дата звернення: 12.10.2025).

19. Understanding the LCOM Metric: Ensuring Single Responsibility in Your Classes. URL: https://medium.com/@suraif16/lack-of-cohesion-in-methods-265a9a26fd66 (дата звернення: 12.10.2025).

20. Coupling Between Object classes (CBO). URL: https://objectscriptquality.com/docs/metrics/coupling-between-object-classes-cbo (дата звернення: 12.10.2025).

21. Response for Class. URL: https://objectscriptquality.com/docs/metrics/response-for-class (дата звернення: 12.10.2025).

22. S. Prykhodko, L. Makarova, K. Prykhodko, A. Pukhalevych. Application of Transformed Prediction Ellipsoids for Outlier Detection in Multivariate Non-Gaussian Data. *Proceedings of the IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, 2020, p. 359–362, doi: 10.1109/TCSET49122.2020.235454.

23. S. Prykhodko, L. Makarova, A. Pukhalevych. Statistical Analysis of the Three-Dimensional Data of Software Metrics RFC, CBO, and WMC that are not Normally Distributed. *Proceedings of International Conference on Applied Innovation in IT,* vol. 13, issue 1, 2025, pp. 127–132. doi: 10.25673/119254.

24. S. Prykhodko, N. Prykhodko, T. Smykodub. A Joint Statistical Estimation of the RFC and CBO Metrics for Open-Source Applications Developed in Java.

Proceedings of the IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT), 2022, p. 160–166, doi: 10.1109/CSIT56902.2022.10000457.

25. K.V. Mardia. Measures of multivariate skewness and kurtosis with applications, *Biometrika*, volume 57, 1970, p. 519–530. doi: 10.1093/biomet/57.3.519.

26. Яка загальна формула коваріації? URL: https://chowk.lis.v.ua/maysternist/yaka-zagalna-formula-kovariacii.html (дата звернення: 15.10.2025).

27. Using Box-Cox to Achieve Multivariate Normality. URL: https://richard-mei97.medium.com/using-box-cox-to-achieve-multivariate-normality-3957f0164e8f (дата звернення: 15.10.2025).

28. Метод максимальної правдоподібності. URL: https://studfile.net/preview/7812027/page:9/ (дата звернення: 15.10.2025).

29. Як розрахувати відстань Махаланобіса? URL: https://avokado.ranok.cx.ua/ukraincyam/yak-rozrakhuvati-vidstan-makhalanobisa.html (дата звернення: 15.10.2025).

30. F-критерій Фішера (f-розподіл). Оцінка різниці між коефіцієнтами варіації. URL: https://studfile.net/preview/5063288/page:10/ (дата звернення: 15.10.2025).

31. Build and ship software on a single, collaborative platform, 2025. URL: https://github.com (дата звернення: 15.10.2025).

32. SourceMeter for Java, 2025. URL: https://sourcemeter.com (дата звернення: 15.10.2025).

33. O. Oriekhov, T. Farionova, L.S. Chernova, L. Chernova, M. Vorona, Nonlinear regression models for software size estimation of Data Science and Machine Learning Java-applications. *Proceedings of the 5th International Workshop IT Project Management (ITPM)*, 2024; CEUR Workshop Proceedings, Vol. 3709, 2024, p. 54–66. doi:10.23939/IW_itpm2024.054.

34. Understanding the Layered Architecture Pattern: A Comprehensive Guide. URL: https://www.drawio.com/blog/uml-overview (дата звернення: 16.10.2025).

35. UML overview - where and why each UML diagram is used. URL: https://www.drawio.com/blog/uml-overview (дата звернення: 16.10.2025).

36. PlantUML. URL: https://plantuml.com (дата звернення: 16.10.2025).

37. Варіанти використання та сценарії (Use Cases and Scenarios). URL: https://www.maxzosim.com/use-cases-and-scenarios/ (дата звернення: 16.10.2024).

38. Діаграма діяльності в UML: символ, компоненти та приклад. URL: https://www.guru99.com/uk/uml-activity-diagram.html (дата звернення: 16.10.2025).

39. Що таке діаграма класів UML і найкращий творець діаграм класів UML. URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/ (дата звернення: 16.10.2025).

40. Python. URL: https://www.python.org (дата звернення: 16.10.2025).

41. Spyder, The Python IDE. URL: https://www.spyder-ide.org (дата звернення: 16.10.2025).

42. Anaconda Distribution. URL: https://www.anaconda.com/download (дата звернення: 16.10.2025).

43. NumPy. URL: https://numpy.org (дата звернення: 16.10.2025).

44. SciPy. URL: https://scipy.org (дата звернення: 16.10.2025).

45. Matplotlib. URL: https://matplotlib.org (дата звернення: 16.10.2025).

46. Scikit-learn. URL: https://scikit-learn.org/stable (дата звернення: 18.10.2025).

47. Pandas. URL: https://pandas.pydata.org (дата звернення: 18.10.2025).

48. Як розрахувати економічну ефективність проектів та підвищити ефективність використання бюджетних коштів? URL: https://flexi-project.com/uk/як-розрахувати-економічну-ефективні/ (дата звернення: 18.10.2025).

49. Основні функції служби охорони праці. URL: https://dsp.gov.ua/faq/osnovni-funktsii-sluzhby-okhorony-pratsi/ (дата звернення: 18.10.2025).

50. Санітарні норми мікроклімату виробничих приміщень (ДСН 3.3.6.042-99). URL: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення: 18.10.2025).

51. Комп'ютер і до сліпоти може довести. URL: https://armyinform.com.ua/2020/08/30/kompyuter-i-do-slipoty-mozhe-dovesty/ (дата звернення: 18.10.2025).

52. Здоров'я користувача ПК: опорно-руховий апарат. URL: https://i.factor.ua/ukr/journals/bk/2009/february/issue-4/article-100268.html (дата звернення: 18.10.2025).

53. Перерви під час роботи за комп'ютером: чи входять в тривалість робочої зміни. URL: https://buhplatforma.com.ua/news/24275-perervi-pd-chas-roboti-za-kompyuterom-chi-vhodyat-v-trivalst-robocho-zmni (дата звернення: 18.10.2025).

54. Про охорону навколишнього природного середовища: Закон України від 15.11.2024 № 1264-XII. URL: https://zakon.rada.gov.ua/laws/show/1264-12#Text (дата звернення: 19.10.2024).

55. Як розробити ЕКОполітику з нуля? URL: https://ecolog-ua.com/news/yak-rozrobyty-ekopolityku-z-nulya (дата звернення: 19.10.2024).

56. Як відповідально утилізувати електротехніку? Кілька лайфхаків і важливих правил. URL: https://kosmotech.com.ua/yak-vidpovidalno-utylizuvaty-elektrotehniku-kilka-lajfhakiv-i-vazhlyvyh-pravyl/ (дата звернення: 19.10.2024).

57. ISO 14001:2015 - Environmental management systems. URL: https://www.iso.org/standard/60857.html (дата звернення: 19.10.2024).

58. Overview of the EPEAT Ecolabel. URL: https://epeat.net/about-epeat (дата звернення: 19.10.2024).

59. 10 КРОКІВ для протидії зміні клімату. URL: https://ecoaction.org.ua/10-krokiv.html (дата звернення: 19.10.2024).

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ СКЛАДНОСТІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОЄКТУВАННЯ JAVA-ЗАСТОСУНКІВ НА ОСНОВІ ЕЛІПСІВ ПРОГНОЗУВАННЯ З ВИКОРИСТАННЯМ МЕТРИК RFC ТА CBO

Вступ

Повна назва програмного продукту: Програмне забезпечення для оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків на основі еліпсів прогнозування з використанням метрик RFC та CBO.

Коротка назва: «Ellipse Predictor».

Сфера застосування: підприємства та команди, що займаються розробкою та аналізом програмного забезпечення мовою програмування Java, зокрема проєкти з відкритим вихідним кодом.

1. Підстави для розробки

Підставою для розробки «Ellipse Predictor» 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

Розроблюване програмне забезпечення призначене для автоматизації оцінювання складності об'єктно-орієнтованого проєктування Java-застосунків на основі аналізу програмних метрик RFC та CBO з використанням математичної моделі у вигляді еліпсів прогнозування. Програма може застосовуватися на ранніх етапах проєктування програмного забезпечення, а також для аналізу вже існуючих Java-проєктів.

2.2 Функціональне призначення

Функціональним призначенням «Ellipse Predictor» є завантаження та перегляд метрик застосунків, що використовують мову програмування Java, 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

– зчитування та обробку вхідних даних метрик Java-застосунків;

– нормалізації метрик RFC та CBO шляхом приведення до відносних показників RFC/NCL та CBO/NCL;

– перевірки вибірок на відповідність багатовимірному нормальному розподілу за тестом Мардіа;

– виконання багатовимірного нормалізуючого перетворення Бокса–Кокса;

– виявлення та видалення викидів на основі квадрату відстані Махаланобіса;

– побудови внутрішнього та зовнішнього еліпсів прогнозування;

– побудови трансформованих еліпсів прогнозування для початкових даних;

– класифікації Java-застосунків за рівнем складності (Low, High, Outlier).

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 програми на Java: RFC/NCL, CBO/NCL;

– метрики RFC/NCL, CBO/NCL для визначення складності.

Вихідні дані:

– нормалізовані значення метрик;

– параметри перетворення Бокса–Кокса;

– матриці коваріацій та їх обернені;

– значення статистик тесту Мардіа;

– порогові значення еліпсів прогнозування;

– результат класифікації складності проєкту (Low, High, Outlier);

– графічні зображення еліпсів прогнозування.

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 запуску, 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 та Spyder. Вихідні коди програми повинні бути реалізовані на мові Python.

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	15.09.2025
2	Розробка архітектури ПЗ	25.09.2025
3	Розробка проекту ПЗ	15.10.2025
4	Реалізація та тестування ПЗ	1.11.2025
5	Випробування ПЗ	5.11.2025

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 ДЛЯ ОЦІНЮВАННЯ СКЛАДНОСТІ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОЄКТУВАННЯ JAVA-ЗАСТОСУНКІВ НА ОСНОВІ ЕЛІПСІВ ПРОГНОЗУВАННЯ З ВИКОРИСТАННЯМ МЕТРИК RFC ТА CBO

main.py

```
# main.py
from ui.app_ui import EllipseApp
import tkinter as tk

if __name__ == "__main__":
    root = tk.Tk()
    app = EllipseApp(root)
    root.mainloop()
```

app_ui.py

```
# ui/app_ui.py
"""
GUI for Ellipse Predictor – fully class-based core integration
(Two-level ellipse: Low / High / Outlier)
"""

import tkinter as tk
from tkinter import filedialog, messagebox, scrolledtext
import pandas as pd
import numpy as np
import os

# --- Core ---
from core.normalization import BoxCoxTransformer
from core.outlier_detection import OutlierDetector
from core.ellipse_model import EllipseModel
from core.utils_excel import ExcelLogger

# --- Visualization ---
from visualization.plot_ellipse import plot_two_ellipses

class EllipseApp:
    def __init__(self, master):
        self.master = master
        self.master.title("Ellipse Predictor")
```

```
        self.master.geometry("850x650")
        self.master.minsize(750, 550)
        self.master.configure(bg="#1e1e1e")

        # --- Style ---
        self.bg = "#1e1e1e"
        self.fg = "#ffffff"
        self.accent = "#007acc"
        self.button_bg = "#2d2d2d"
        self.button_hover = "#3c3c3c"
        self.textbox_bg = "#252526"

        self.default_font = ("Segoe UI", 10)
        self.bold_font = ("Segoe UI", 11, "bold")

        self.apply_style()

        # --- State ---
        self.file_path = None
        self.raw_df = None
        self.data = None
        self.cleaned_data = None
        self.normalized_data = None
        self.lambdas = None

        self.ellipse_low = None
        self.ellipse_high = None

        self.test_points = None
        self.test_df_raw = None

        # --- UI ---
        tk.Label(
            master,
            text="Ellipse Predictor Tool",
            font=("Segoe UI", 18, "bold"),
            bg=self.bg,
            fg=self.accent
        ).pack(pady=15)

        self.build_ui()

    def apply_style(self):
        self.btn_style = {
            "bg": self.button_bg,
            "fg": self.fg,
            "font": self.default_font,
            "activebackground": self.button_hover,
```

```
        "activeforeground": "#ffffff",
        "relief": tk.FLAT,
        "bd": 1,
        "padx": 5,
        "pady": 3
    }

def build_ui(self):
    # === FILE ===
    file_frame = tk.LabelFrame(self.master, text="File Operations",
                               fg=self.accent, bg=self.bg, font=self.bold_font)
    file_frame.pack(fill="x", padx=15, pady=8)
    self.add_button(file_frame, "Select Excel File", self.select_file, 0, 0)
    self.add_button(file_frame, "Export Cleaned Data", self.export_cleaned_data, 0, 1)

    # === ANALYSIS ===
    analysis_frame = tk.LabelFrame(self.master, text="Analysis",
                                   fg=self.accent, bg=self.bg, font=self.bold_font)
    analysis_frame.pack(fill="x", padx=15, pady=8)
    self.add_button(analysis_frame, "Run Analysis", self.run_analysis, 0, 0)
    self.add_button(analysis_frame, "Show Ellipses", self.show_ellipses, 0, 1)

    # === TEST ===
    test_frame = tk.LabelFrame(self.master, text="Test Set Estimation",
                               fg=self.accent, bg=self.bg, font=self.bold_font)
    test_frame.pack(fill="x", padx=15, pady=8)
    self.add_button(test_frame, "Load Test Set", self.load_test_set, 0, 0)
    self.add_button(test_frame, "Predict Test Set", self.predict_test_set, 0, 1)

    # === MANUAL ===
    input_frame = tk.LabelFrame(self.master, text="Predict Single Point",
                                fg=self.accent, bg=self.bg, font=self.bold_font)
    input_frame.pack(fill="x", padx=15, pady=8)

    tk.Label(input_frame, text="CBO/NCL:", bg=self.bg, fg=self.fg).grid(row=0, column=0)
    tk.Label(input_frame, text="RFC/NCL:", bg=self.bg, fg=self.fg).grid(row=0, column=2)

    self.x1_var = tk.StringVar()
    self.x2_var = tk.StringVar()

    tk.Entry(input_frame, textvariable=self.x1_var,
             bg=self.textbox_bg, fg=self.fg,
             insertbackground="white", width=12).grid(row=0, column=1)
    tk.Entry(input_frame, textvariable=self.x2_var,
             bg=self.textbox_bg, fg=self.fg,
             insertbackground="white", width=12).grid(row=0, column=3)

    self.add_button(input_frame, "Check Point", self.check_point, 0, 4)
```

```
        self.add_button(input_frame, "Check and show Point", self.check_and_plot_point, 0, 5)

        # === LOG ===
        log_frame = tk.LabelFrame(self.master, text="Log Output",
                                  fg=self.accent, bg=self.bg, font=self.bold_font)
        log_frame.pack(fill="both", expand=True, padx=15, pady=8)

        self.output_box = scrolledtext.ScrolledText(
            log_frame,
            wrap=tk.WORD,
            font=("Consolas", 10),
            bg=self.textbox_bg,
            fg="#d4d4d4",
            insertbackground="white"
        )
        self.output_box.pack(fill="both", expand=True, padx=5, pady=5)

        exit_style = self.btn_style.copy()
        exit_style["bg"] = "#a83232"
        tk.Button(self.master, text="Exit",
                  command=self.master.quit,
                  **exit_style).pack(pady=8)

    def add_button(self, parent, text, cmd, row, col):
        btn = tk.Button(parent, text=text, command=cmd, **self.btn_style)
        btn.grid(row=row, column=col, padx=6, pady=6)

    def log(self, text):
        self.output_box.insert(tk.END, text + "\n")
        self.output_box.see(tk.END)

    # ============================
    # CORE
    # ============================

    def select_file(self):
        path = filedialog.askopenfilename(filetypes=[("Excel Files", "*.xlsx")])
        if path:
            self.file_path = path
            self.log(f"Selected file: {path}")

    def run_analysis(self):
        if not self.file_path:
            messagebox.showwarning("No file", "Select an Excel file first.")
            return

        try:
            self.log("Loading data...")
```

```
            df = pd.read_excel(self.file_path)
            df = df[(df[['CBO/NCL', 'RFC/NCL']] != 0).all(axis=1)]
            data = df[['CBO/NCL', 'RFC/NCL']].values

            logger = ExcelLogger(self.log)

            self.log("Running iterative outlier removal...")
            cleaned, norm, ids, lambdas = OutlierDetector.remove_iteratively(
                data,
                normalization_type="multivariate_boxcox",
                folder="outlier_logs",
                logger=logger
            )

            self.cleaned_data = cleaned
            self.normalized_data = norm
            self.lambdas = lambdas

            self.log("Building two-level ellipse model...")
            self.ellipse_low, self.ellipse_high = EllipseModel.build_two(norm)

            self.log("Analysis complete.")
            messagebox.showinfo("Done", "Model successfully built.")

        except Exception as e:
            messagebox.showerror("Error", str(e))

    def show_ellipses(self):
        if self.ellipse_low is None:
            messagebox.showwarning("No model", "Run analysis first.")
            return

        plot_two_ellipses(
            self.normalized_data,
            self.ellipse_low,
            self.ellipse_high,
            self.lambdas,
            orig_data=self.cleaned_data,
            show_training=True,
            show_test=False
        )

    def export_cleaned_data(self):
        if self.cleaned_data is None:
            return
        folder = filedialog.askdirectory()
        if not folder:
            return
```

```
    out = os.path.join(folder, "Cleaned_Data.xlsx")
    pd.DataFrame(self.cleaned_data,
                 columns=["CBO/NCL", "RFC/NCL"]).to_excel(out, index=False)
    self.log(f"Saved cleaned data → {out}")

# ============================
# TEST SET
# ============================

def load_test_set(self):
    path = filedialog.askopenfilename(filetypes=[("Excel Files", "*.xlsx")])
    if not path:
        return
    df = pd.read_excel(path)
    df = df[(df[['CBO/NCL', 'RFC/NCL']] != 0).all(axis=1)]
    self.test_points = df[['CBO/NCL', 'RFC/NCL']].values
    self.test_df_raw = df
    self.log(f"Loaded test set ({len(self.test_points)} points)")

def predict_test_set(self):
    if self.test_points is None:
        messagebox.showwarning("No test set", "Load test set first.")
        return

    norm_test = BoxCoxTransformer.forward(self.test_points, self.lambdas)

    results, classes = [], []

    for i, z in enumerate(norm_test):
        label, dist = EllipseModel.classify(
            z, self.ellipse_low, self.ellipse_high
        )
        classes.append(label)
        results.append({
            "CBO/NCL": self.test_points[i][0],
            "RFC/NCL": self.test_points[i][1],
            "Z1": z[0],
            "Z2": z[1],
            "Mahalanobis²": dist,
            "Class": label
        })

    df = pd.DataFrame(results)
    df.to_excel("TestPoints_Predictions.xlsx", index=False)
    self.log("Predictions saved to TestPoints_Predictions.xlsx")

    plot_two_ellipses(
        self.normalized_data,
```

```
            self.ellipse_low,
            self.ellipse_high,
            self.lambdas,
            new_points=norm_test,
            classes=classes,
            orig_data=self.cleaned_data,
            show_training=False,
            show_test=True
        )

    def check_and_plot_point(self):
        if self.ellipse_low is None:
            messagebox.showwarning("No model", "Run analysis first.")
            return

        try:
            x = np.array([
                float(self.x1_var.get()),
                float(self.x2_var.get())
            ])

            # Normalize
            z = BoxCoxTransformer.forward(x, self.lambdas)

            # Classify
            label, dist = EllipseModel.classify(
                z, self.ellipse_low, self.ellipse_high
            )

            self.log(f"[PLOT] Point {x.tolist()} → {label} (Mahalanobis²={dist:.6f})")

            # Plot with this point as a temporary test set
            plot_two_ellipses(
                self.normalized_data,
                self.ellipse_low,
                self.ellipse_high,
                self.lambdas,
                new_points=np.array([z]),
                classes=[label],
                orig_data=self.cleaned_data,
                show_training=False,
                show_test=True,
                title="Single Point Classification"
            )

        except Exception as e:
            messagebox.showerror("Error", str(e))
```

```
    # ============================
    # SINGLE POINT
    # ============================

    def check_point(self):
        try:
            x = np.array([
                float(self.x1_var.get()),
                float(self.x2_var.get())
            ])
            z = BoxCoxTransformer.forward(x, self.lambdas)
            label, dist = EllipseModel.classify(
                z, self.ellipse_low, self.ellipse_high
            )
            self.log(f"Point {x.tolist()} → {label} (Mahalanobis²={dist:.6f})")
        except Exception as e:
            messagebox.showerror("Error", str(e))

if __name__ == "__main__":
    root = tk.Tk()
    app = EllipseApp(root)
    root.mainloop()
```

plot_ellipse.py

```
# visualization/plot_ellipse.py
import matplotlib.pyplot as plt
import numpy as np
from core.normalization import BoxCoxTransformer

def _ellipse(mean, cov, threshold):
    eigvals, eigvecs = np.linalg.eigh(cov)
    angles = np.linspace(0, 2*np.pi, 400)
    unit = np.array([np.cos(angles), np.sin(angles)])
    axes = np.sqrt(eigvals * threshold)
    return (eigvecs @ (unit * axes[:, None])) + mean[:, None]

def plot_two_ellipses(norm_data, ellipse_low, ellipse_high,
                      lambdas, new_points=None, classes=None,
                      orig_data=None,
                      show_training=True,
                      show_test=True,
                      title="Two-Ellipse Model"):

    e_low = _ellipse(*ellipse_low)
    e_high = _ellipse(*ellipse_high)

    # ---------- Normalized ----------
```

```
plt.figure(figsize=(8,6))
plt.plot(e_low[0], e_low[1], 'g', label="Low complexity (α=0.05)")
plt.plot(e_high[0], e_high[1], 'r--', label="High complexity (α=0.005)")

if show_training:
    plt.scatter(norm_data[:,0], norm_data[:,1], alpha=0.5, label="Training")

if show_test and new_points is not None:
    colors = {"Low":"green","High":"orange","Outlier":"red"}
    for cls in colors:
        idx = [i for i,c in enumerate(classes) if c==cls]
        if idx:
            pts = new_points[idx]
            plt.scatter(pts[:,0], pts[:,1],
                        marker='x', s=80,
                        color=colors[cls], label=f"Test ({cls})")

plt.xlabel("Z1")
plt.ylabel("Z2")
plt.legend()
plt.grid()
plt.title(title + " (Normalized)")
plt.show()

# ---------- Original ----------
e_low_o = BoxCoxTransformer.inverse(e_low.T, lambdas).T
e_high_o = BoxCoxTransformer.inverse(e_high.T, lambdas).T

plt.figure(figsize=(8,6))
plt.plot(e_low_o[0], e_low_o[1], 'g')
plt.plot(e_high_o[0], e_high_o[1], 'r--')

if show_training:
    plt.scatter(orig_data[:,0], orig_data[:,1], alpha=0.5)

if show_test:
    pts_o = BoxCoxTransformer.inverse(new_points, lambdas)
    for cls in colors:
        idx = [i for i,c in enumerate(classes) if c==cls]
        if idx:
            p = pts_o[idx]
            plt.scatter(p[:,0], p[:,1],
                        marker='x', s=80,
                        color=colors[cls])

plt.xlabel("CBO/NCL")
plt.ylabel("RFC/NCL")
plt.grid()
```

```
    plt.title(title + " (Original)")
    plt.show()
```

ellipse_model.py

```
# core/ellipse_model.py
import numpy as np
from scipy.stats import f
from .outlier_detection import OutlierDetector

class EllipseModel:
    """Two-level ellipse construction and classification."""

    @staticmethod
    def build_two(data: np.ndarray, alpha_low=0.05, alpha_high=0.005):
        mean = np.mean(data, axis=0)
        cov = OutlierDetector.calculate_covariance_matrix(data)
        N, k = data.shape

        def threshold(alpha):
            f_crit = f.ppf(1 - alpha, k, N - k)
            return (k * (N**2 - 1)) / (N * (N - k)) * f_crit

        return (
            (mean, cov, threshold(alpha_low)),
            (mean, cov, threshold(alpha_high))
        )

    @staticmethod
    def classify(point, ellipse_low, ellipse_high):
        def dist(p, m, c):
            d = p - m
            return np.dot(np.dot(d, np.linalg.pinv(c)), d.T)

        m, c, t_low = ellipse_low
        _, _, t_high = ellipse_high

        d = dist(point, m, c)

        if d <= t_low:
            return "Low", d
        elif d <= t_high:
            return "High", d
        else:
            return "Outlier", d
```

normality_check.py

```
# core/normality_check.py
import numpy as np
import pandas as pd
from scipy.stats import chi2, norm

class NormalityChecker:
    """Multivariate normality check (Mardia’s skewness & kurtosis)."""

    @staticmethod
    def _mean(data: pd.DataFrame) -> np.ndarray:
        return data.mean(axis=0).values

    @staticmethod
    def _covariance(data: pd.DataFrame) -> np.ndarray:
        return np.cov(data.T, bias=True)

    @staticmethod
    def _inverse_cov(cov: np.ndarray) -> np.ndarray:
        return np.linalg.inv(cov)

    @staticmethod
    def _skewness(data: pd.DataFrame, mean_vec: np.ndarray,
                  inv_cov: np.ndarray) -> tuple[float, float]:
        N = len(data)
        skew_sum = 0.0
        for i in range(N):
            xi = data.iloc[i].values - mean_vec
            for j in range(N):
                xj = data.iloc[j].values - mean_vec
                skew_sum += np.dot(np.dot(xi, inv_cov), xj) ** 3
        skewness = skew_sum / (N**2)
        Z_skew = (N * skewness) / 6
        return skewness, Z_skew

    @staticmethod
    def _kurtosis(data: pd.DataFrame, mean_vec: np.ndarray,
                  inv_cov: np.ndarray) -> tuple[float, float]:
        N = len(data)
        p = len(mean_vec)
        kurt_sum = 0.0
        for i in range(N):
            xi = data.iloc[i].values - mean_vec
            kurt_sum += np.dot(np.dot(xi, inv_cov), xi) ** 2
        kurtosis = kurt_sum / N
        expected = p * (p + 2)
        variance = (8 * p * (p + 2)) / N
```

```
    Z_kurt = (kurtosis - expected) / np.sqrt(variance)
    return kurtosis, Z_kurt

@staticmethod
def _chi2_critical(p: int, alpha: float = 0.005):
    df = p * (p + 1) * (p + 2) / 6
    return chi2.ppf(1 - alpha, df), df

@staticmethod
def _z_critical(alpha: float = 0.005):
    return norm.ppf(1 - alpha)

@classmethod
def analyze(cls, data: pd.DataFrame, alpha: float = 0.005) -> dict:
    mean_vec = cls._mean(data)
    cov = cls._covariance(data)
    inv_cov = cls._inverse_cov(cov)

    skewness, Z_skew = cls._skewness(data, mean_vec, inv_cov)
    kurtosis, Z_kurt = cls._kurtosis(data, mean_vec, inv_cov)
    chi_crit, chi_df = cls._chi2_critical(len(mean_vec), alpha)
    z_crit = cls._z_critical(alpha)

    skew_normal = Z_skew <= chi_crit
    kurt_normal = abs(Z_kurt) <= z_crit

    return {
        "mean_vector": mean_vec,
        "covariance_matrix": cov,
        "inverse_covariance_matrix": inv_cov,
        "skewness": skewness,
        "Z_skewness": Z_skew,
        "kurtosis": kurtosis,
        "Z_kurtosis": Z_kurt,
        "chi_squared_critical": chi_crit,
        "chi_df": chi_df,
        "Z_critical": z_crit,
        "verdict_skewness": (
            "Normal based on skewness"
            if skew_normal else
            "Not normal based on skewness (asymmetry detected)"
        ),
        "verdict_kurtosis": (
            "Normal based on kurtosis"
            if kurt_normal else
            "Not normal based on kurtosis (excess/peakedness detected)"
        ),
        "final_verdict": (
```

```
            "Data is multivariate normal"
            if (skew_normal and kurt_normal) else
            "Data is NOT multivariate normal"
        ),
    }
```

normalization.py

```
# core/normalization.py
import numpy as np
from scipy.stats import boxcox_normmax
from scipy.optimize import minimize
from numpy.linalg import slogdet

class BoxCoxTransformer:
    """Box-Cox (univariate & multivariate) + inverse."""

    @staticmethod
    def _apply_one(x: np.ndarray, lam: float) -> np.ndarray:
        return np.log(x) if lam == 0 else (np.power(x, lam) - 1) / lam

    @staticmethod
    def forward(data: np.ndarray, lambdas: list[float]) -> np.ndarray:
        arr = np.asarray(data, dtype=np.float64)
        if arr.ndim == 1:
            arr = arr.reshape(1, -1)

        n_samples, n_features = arr.shape
        if len(lambdas) != n_features:
            raise ValueError(f"lambdas length {len(lambdas)} ≠ features {n_features}")

        transformed = np.zeros_like(arr)
        for i, lam in enumerate(lambdas):
            transformed[:, i] = BoxCoxTransformer._apply_one(arr[:, i], lam)

        return transformed.flatten() if transformed.shape[0] == 1 else transformed

    @staticmethod
    def estimate_univariate_lambdas(data: np.ndarray) -> list[float]:
        return [boxcox_normmax(data[:, i], method='mle') for i in range(data.shape[1])]

    @staticmethod
    def multivariate_mle(data: np.ndarray):
        if np.any(data <= 0):
            raise ValueError("Box-Cox requires strictly positive data.")

        def transform(col, lam):
```

```
            return np.log(col) if lam == 0 else (col ** lam - 1) / lam

        def neg_log_likelihood(lambdas):
            transformed = np.column_stack([transform(data[:, i], lambdas[i])
                                           for i in range(data.shape[1])])
            cov = np.cov(transformed, rowvar=False)
            sign, logdet = slogdet(cov)
            if sign <= 0:
                return np.inf
            n = data.shape[0]
            log_jac = np.sum((lambdas - 1) * np.sum(np.log(data), axis=0))
            return 0.5 * n * logdet - log_jac

        init = BoxCoxTransformer.estimate_univariate_lambdas(data)
        res = minimize(neg_log_likelihood, init, method='L-BFGS-B')
        if not res.success:
            raise RuntimeError("Optimization failed: " + res.message)

        optimal = res.x
        transformed = np.column_stack([transform(data[:, i], optimal[i])
                                       for i in range(data.shape[1])])
        return transformed, optimal

    @staticmethod
    def inverse(transformed: np.ndarray, lambdas: list[float]) -> np.ndarray:
        arr = np.asarray(transformed)
        if arr.ndim != 2:
            raise ValueError("transformed must be 2-D")

        n_rows, n_cols = arr.shape
        if n_cols != len(lambdas) and n_rows == len(lambdas):
            arr = arr.T
            n_rows, n_cols = arr.shape

        original = np.zeros_like(arr, dtype=np.float64)
        for i, lam in enumerate(lambdas):
            z = arr[:, i]
            original[:, i] = np.exp(z) if lam == 0 else np.power(z * lam + 1, 1.0 / lam)
        return original
```

outlier_detection.py

```
# core/outlier_detection.py
import os
import numpy as np
import pandas as pd
from scipy.stats import f
```

```
from .normalization import BoxCoxTransformer
from .utils_excel import ExcelLogger

class OutlierDetector:
    """Mahalanobis distance, outlier detection & iterative removal."""

    @staticmethod
    def calculate_covariance_matrix(data: np.ndarray) -> np.ndarray:
        mean = np.mean(data, axis=0)
        centered = data - mean
        return np.cov(centered, rowvar=False)

    @staticmethod
    def mahalanobis_distances(data: np.ndarray, cov: np.ndarray) -> np.ndarray:
        mean = np.mean(data, axis=0)
        centered = data - mean
        inv_cov = np.linalg.pinv(cov)
        return np.array([np.dot(np.dot(x, inv_cov), x.T) for x in centered])

    @classmethod
    def detect(cls, data: np.ndarray, alpha: float = 0.005):
        N, k = data.shape
        cov = cls.calculate_covariance_matrix(data)
        distances = cls.mahalanobis_distances(data, cov)
        f_crit = f.ppf(1 - alpha, k, N - k)
        test_stat = (k * (N**2 - 1)) / (N * (N - k)) * f_crit
        outlier_idx = np.where(distances > test_stat)[0]
        return outlier_idx, distances, f_crit, test_stat

    @classmethod
    def remove_iteratively(cls,
                           data: np.ndarray,
                           alpha: float = 0.005,
                           folder: str = "logs",
                           normalization_type: str = "multivariate_boxcox",
                           column_names: list[str] | None = None,
                           logger: ExcelLogger | None = None):
        os.makedirs(folder, exist_ok=True)
        iteration = 1
        data_copy = data.copy()
        column_names = column_names or ["CBO/NCL", "RFC/NCL"]
        ids = np.arange(1, data.shape[0] + 1)
        current_ids = ids.copy()
        removed_ids = []

        while True:
            # ---- Normalization ----
            if normalization_type == "multivariate_boxcox":
```

```
    logger.log(f"Iteration {iteration}: Using multivariate Box-Cox normalization.")
    normalized, lambdas = BoxCoxTransformer.multivariate_mle(data_copy)
    logger.log(f"Iteration {iteration}: Lambdas = {np.round(lambdas, 6).tolist()}")
elif normalization_type == "boxcox":
    lambdas = BoxCoxTransformer.estimate_univariate_lambdas(data_copy)
    normalized = BoxCoxTransformer.forward(data_copy, lambdas)
    logger.log(f"Iteration {iteration}: Optimal Box-Cox lambdas = {lambdas}")
else:
    raise ValueError(f"Unknown normalization type: {normalization_type}")

# ---- Outlier detection ----
outliers, distances, f_crit, test_stat = cls.detect(normalized, alpha)

if len(outliers) == 0:
    logger.log(f"No outliers detected at iteration {iteration}. Stopping.")
    break

# ---- Log iteration ----
cov = cls.calculate_covariance_matrix(normalized)
df_iter = pd.DataFrame({
    "id": current_ids,
    column_names[0]: data_copy[:, 0],
    column_names[1]: data_copy[:, 1],
    "Mahalanobis Distance": distances,
    "F-Critical Value": [f_crit] * len(data_copy),
    "Test Statistic": [test_stat] * len(data_copy),
    "Conclusion": ["Outlier" if i in outliers else "Not Outlier"
                   for i in range(len(data_copy))]
})
cov_df = pd.DataFrame(cov, columns=column_names, index=column_names)
log_path = os.path.join(folder, f"iteration_{iteration}.xlsx")
with pd.ExcelWriter(log_path, engine="openpyxl") as writer:
    df_iter.to_excel(writer, index=False, sheet_name="Log")
    cov_df.to_excel(writer, sheet_name="Log",
                    startrow=len(df_iter) + 2)
logger.try_highlight(log_path)

# ---- Remove largest outlier ----
largest_idx = outliers[np.argmax(distances[outliers])]
removed_ids.append(current_ids[largest_idx])
logger.log(
    f"Iteration {iteration}: Removed ID [{current_ids[largest_idx]}] "
    f"NORMALIZED:({normalized[largest_idx].tolist()}), "
    f"ORIGINAL:({data_copy[largest_idx].tolist()}), "
    f"Mahalanobis² = {distances[largest_idx]:.6f}, "
    f"Test = {test_stat:.6f}"
)
```

```
            data_copy = np.delete(data_copy, largest_idx, axis=0)
            current_ids = np.delete(current_ids, largest_idx, axis=0)
            iteration += 1

        # ---- Final recompute ----
        logger.log("\nRecomputing final normalization and covariance...")
        norm_final, lambdas_final = BoxCoxTransformer.multivariate_mle(data_copy)
        cov_final = cls.calculate_covariance_matrix(norm_final)
        dist_final = cls.mahalanobis_distances(norm_final, cov_final)
        Nf, kf = norm_final.shape
        f_crit_f = f.ppf(1 - alpha, kf, Nf - kf)
        test_f = (kf * (Nf**2 - 1)) / (Nf * (Nf - kf)) * f_crit_f

        df_final = pd.DataFrame({
            "id": current_ids,
            column_names[0]: norm_final[:, 0],
            column_names[1]: norm_final[:, 1],
            "Mahalanobis Distance": dist_final,
            "F-Critical Value": [f_crit_f] * len(norm_final),
            "Test Statistic": [test_f] * len(norm_final),
            "Conclusion": ["Not Outlier"] * len(norm_final),
        })
        cov_f_df = pd.DataFrame(cov_final, columns=column_names, index=column_names)
        removed_df = pd.DataFrame({"Removed ID": removed_ids})

        final_path = os.path.join(folder, "iteration_final.xlsx")
        with pd.ExcelWriter(final_path, engine="openpyxl") as writer:
            df_final.to_excel(writer, index=False, sheet_name="Log")
            cov_f_df.to_excel(writer, sheet_name="Log",
                              startrow=len(df_final) + 2)
            removed_df.to_excel(writer, index=False, sheet_name="Removed IDs")
        logger.try_highlight(final_path)

        logger.log(f"Final cleaned data written to {final_path}")
        return data_copy, norm_final, current_ids.tolist(), lambdas_final
```

utils_excel.py

```
# core/utils_excel.py
from openpyxl import load_workbook
from openpyxl.styles import PatternFill

class ExcelLogger:
    """Console / GUI logger + Excel highlighting."""

    def __init__(self, override_print=None):
```

```
    self.override = override_print

def log(self, msg: str):
    if self.override:
        self.override(msg)
    else:
        print(msg)

@staticmethod
def _highlight(filepath: str):
    red = PatternFill(start_color="FF9999", end_color="FF9999", fill_type="solid")
    wb = load_workbook(filepath)
    ws = wb.active
    col = next((c for c in range(1, ws.max_column + 1)
                if ws.cell(row=1, column=c).value == "Conclusion"), None)
    if not col:
        wb.save(filepath)
        return
    for row in range(2, ws.max_row + 1):
        if ws.cell(row=row, column=col).value == "Outlier":
            for c in range(1, ws.max_column + 1):
                ws.cell(row=row, column=c).fill = red
    wb.save(filepath)

def try_highlight(self, filepath: str):
    try:
        self._highlight(filepath)
    except Exception:
        pass
```

ДОДАТОК В – ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ СКЛАДНОСТІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОЄКТУВАННЯ JAVA-ЗАСТОСУНКІВ НА ОСНОВІ ЕЛІПСІВ ПРОГНОЗУВАННЯ З ВИКОРИСТАННЯМ МЕТРИК RFC ТА CBO

Програмне забезпечення «Ellipse Predictor» розроблене для оцінювання складності об'єктно-орієнтованого проектування Java-застосунків з використанням метрик RFC та CBO. Інструмент призначений для очищення даних від викидів, перевірки багатовимірної нормальності, нормалізації даних та побудови двохрівневої еліптичної моделі, яка використовується для оцінювання та класифікації об'єктів за складністю.

Призначення та функціональні можливості:

1. Завантаження метрик.

Програма працює з файлами формату Excel (XLSX), що містять значення програмних метрик: CBO/NCL (Coupling Between Objects per Class), RFC/NCL (Response For a Class per Class). Користувач може завантажувати власні набори даних для подальшого аналізу. Перед обробкою виконується перевірка коректності даних (виключення нульових значень).

2. Нормалізація даних

Для забезпечення коректності статистичного аналізу реалізовано:

– одновимірну та багатовимірну Box–Cox нормалізацію;

– автоматичне оцінювання параметрів λ методом максимальної правдоподібності;

– можливість виконання зворотного перетворення для візуалізації результатів в оригінальному просторі метрик.

3. Перевірка багатовимірної нормальності

У програмі реалізовано перевірку багатовимірної нормальності розподілу даних на основі критеріїв Мардії:

– асиметрії (skewness);

– ексцесу (kurtosis).

Для кожного критерію обчислюються відповідні статистики та критичні значення, після чого формується підсумковий висновок щодо відповідності даних багатовимірному нормальному розподілу.

4. Виявлення та видалення викидів

Очищення даних виконується ітеративно з використанням:

– квадрата відстані Махаланобіса;

– F-критерію для багатовимірних вибірок.

На кожній ітерації:

– виконується нормалізація даних;

– обчислюється коваріаційна матриця;

– визначається тестова статистика та критичне значення;

– найбільш віддалене аномальне спостереження автоматично вилучається.

Результати кожної ітерації зберігаються у вигляді Excel-звітів із підсвічуванням викидів.

5. Побудова двохрівневої еліптичної моделі

На основі очищених та нормалізованих даних будується дві вкладені еліптичні області:

– еліпс низької складності ($\alpha = 0{,}05$),

– еліпс високої складності ($\alpha = 0{,}005$).

Модель базується на коваріаційній матриці та середньому векторі вибірки й дозволяє виконувати класифікацію об’єктів у три категорії:

– Low complexity,

– High complexity,

– Outlier.

6. Класифікація тестових даних і окремих точок

Програма підтримує:

– завантаження тестової вибірки;

– нормалізацію тестових даних із використанням параметрів навчальної вибірки;

– автоматичну класифікацію кожної точки;

– оцінювання належності до одного з класів складності.

Також передбачена можливість ручного введення окремої точки та її миттєвої перевірки.

7. Візуалізація результатів

Реалізовано побудову графіків:

– у нормалізованому просторі;

– в оригінальному просторі метрик після зворотного перетворення Box–Cox.

На графіках відображаються:

– навчальні дані,

– тестові точки,

– еліпси низької та високої складності з кольоровим маркуванням.

8. Збереження результатів

Програма підтримує:

– експорт очищених даних у форматі XLSX;

– автоматичне формування звітів з логами ітерацій;

– збереження результатів класифікації тестових наборів.

Мова розробки та технології:

Програмне забезпечення «Ellipse Predictor» реалізовано мовою Python з використанням таких бібліотек: NumPy — чисельні обчислення; Pandas — обробка табличних даних; SciPy — статистичні критерії та оптимізація; Matplotlib — візуалізація; Tkinter — графічний інтерфейс користувача; OpenPyXL — формування Excel-звітів.

Архітектура проєкту є модульною та включає окремі компоненти для нормалізації, аналізу нормальності, виявлення викидів, побудови еліптичної моделі та візуалізації.

Принципи роботи:

– Користувач завантажує Excel-файл із програмними метриками.
– Виконується нормалізація та перевірка багатовимірної нормальності.
– Дані очищуються від викидів ітеративним методом.
– Будується два еліпса із різною точністю.
– Проводиться класифікація тестових даних або окремих точок.
– Результати візуалізуються та зберігаються у файли.

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.

Ellipse Predictor є ефективним інструментом для розробників, який спрощує процес оцінювання складності об’єктно-орієнтованого проектування Java-застосунків з використанням метрик.

ДОДАТОК Г – ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ СКЛАДНОСТІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОЄКТУВАННЯ JAVA-ЗАСТОСУНКІВ НА ОСНОВІ ЕЛІПСІВ ПРОГНОЗУВАННЯ З ВИКОРИСТАННЯМ МЕТРИК RFC ТА CBO

1. Завантаження вхідних даних

Натисніть кнопку «Select Excel File» у блоці File Operations.

– У діалоговому вікні виберіть файл у форматі .xlsx, що містить дані про програмні проєкти.

Вхідний файл повинен містити такі стовпці:

– CBO/NCL

– RFC/NCL

Записи з нульовими значеннями автоматично відфільтровуються.

– Після вибору файлу шлях до нього відображається в журналі подій.

На рисунку Г.1 показано процес вибору та завантаження даних у програму.

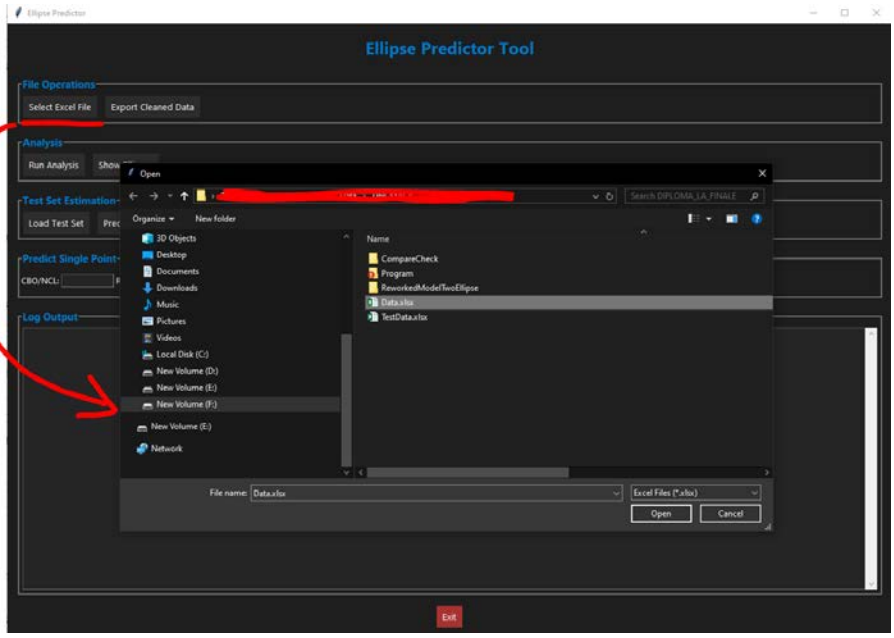

Рисунок Г.1 – Завантаження даних у програму «Ellipse Predictor»

2. Аналіз даних та побудова моделі

Натисніть кнопку «Run Analysis» у блоці Analysis.

У процесі аналізу програма автоматично виконує:

– багатовимірну Box–Cox нормалізацію даних;

– ітеративне виявлення та видалення викидів за відстанню Махаланобіса;

– обчислення коваріаційної та оберненої коваріаційної матриць;

– формування журналів ітерацій у форматі Excel;

– побудову двох еліпсів:

– еліпс низької складності (α = 0,05);

– еліпс високої складності (α = 0,005).

– Після завершення аналізу з’являється повідомлення про успішне створення моделі.

На рисунку Г.2 показано процес аналізу в програмі «Ellipse Predictor».

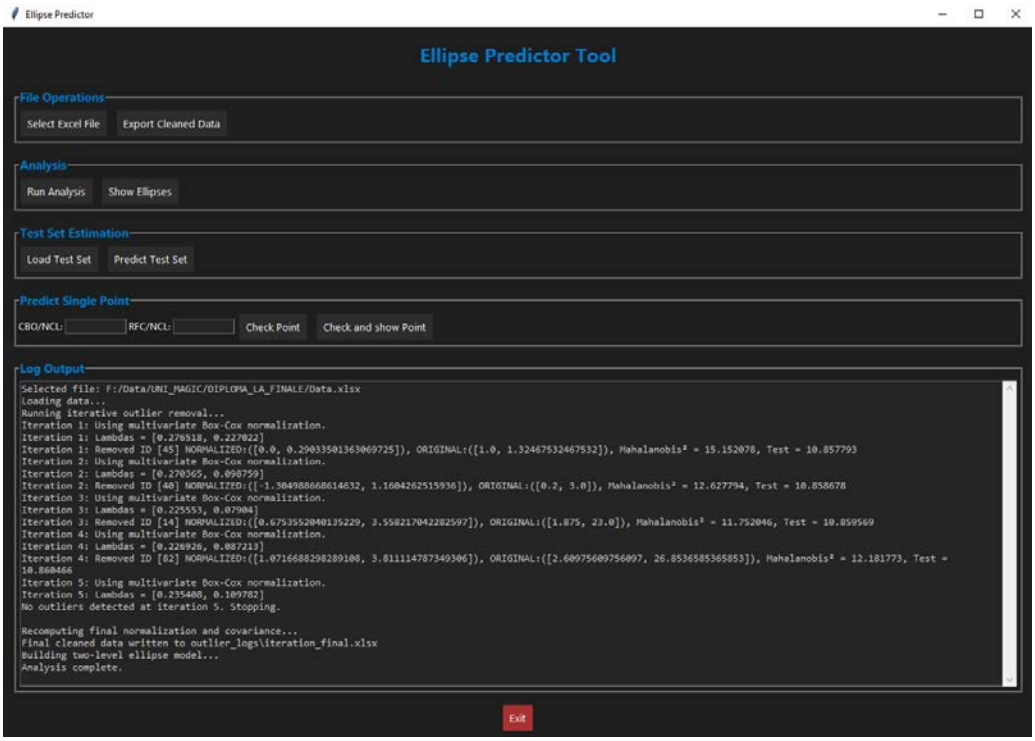

Рисунок Г.2 – Аналіз даних у програмі «Ellipse Predictor»

3. Візуалізація результатів

Для перегляду побудованої моделі натисніть кнопку «Show Ellipses».

Програма відображає два графіки:

– у нормалізованому просторі (Z1–Z2);

– в оригінальному просторі метрик (CBO/NCL – RFC/NCL).

На графіках відображаються:

– навчальні точки;

– еліпси низької та високої складності.

На рисунках Г.3 та Г.4 показано приклад візуалізації в програмі «Ellipse Predictor».

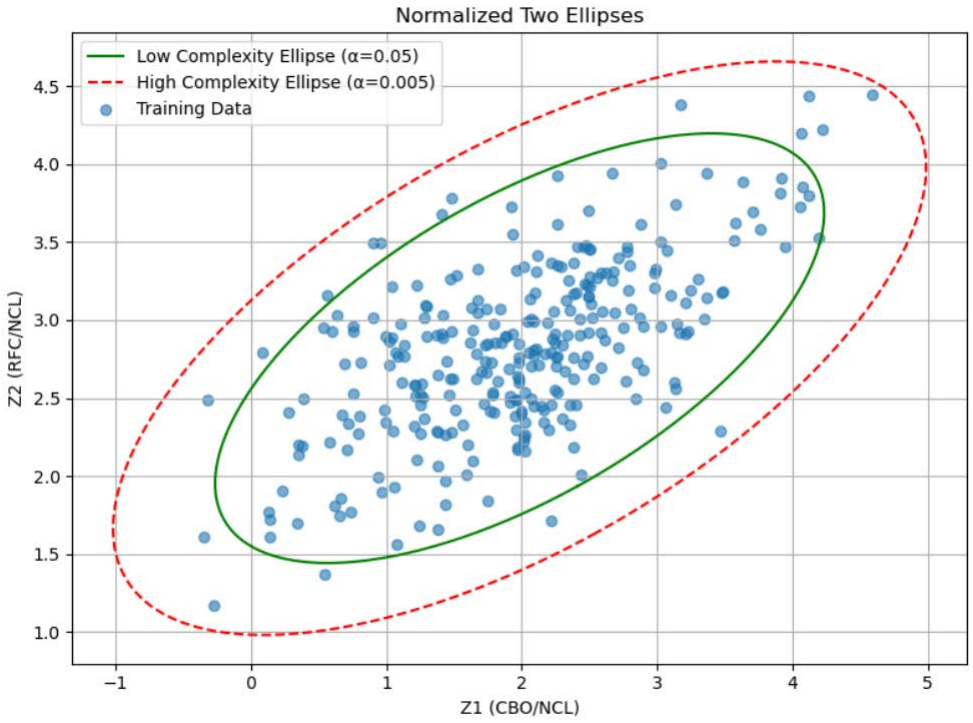

Рисунок Г.3 – Еліпси прогнозування для нормалізованих метрик RFC/NCL та CBO/NCL

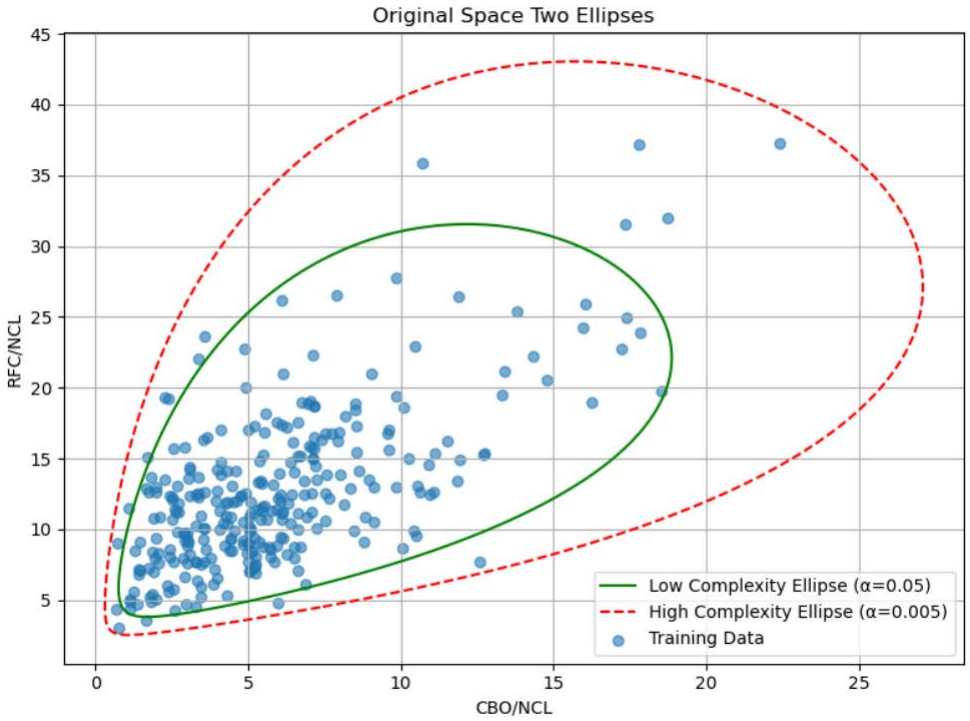

Рисунок Г.4 – Трансформовані еліпси прогнозування для метрик RFC/NCL та CBO/NCL

4. Аналіз тестової вибірки

Натисніть кнопку «Load Test Set» та виберіть Excel-файл з тестовими даними.

Файл повинен містити стовпці CBO/NCL та RFC/NCL.

Після завантаження натисніть кнопку «Predict Test Set».

Для кожної точки тестової вибірки виконується:

– нормалізація;

– обчислення відстані Махаланобіса;

– класифікація за двоеліпсною моделлю.

Результати автоматично зберігаються у файл TestPoints_Predictions.xlsx та відображаються на графіку. На рисунках Г.5 та Г.6 показано приклад візуалізації в програмі.

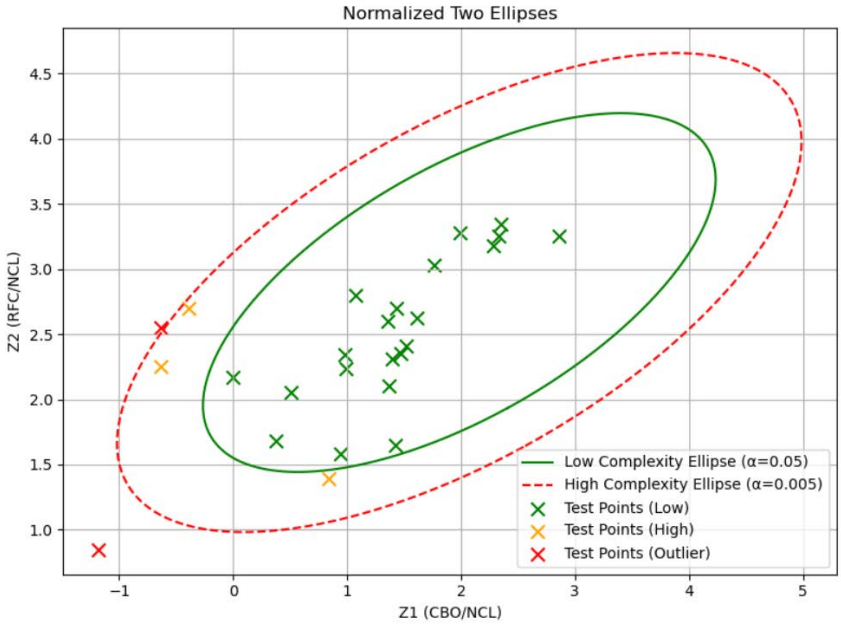

Рисунок Г.5 – Еліпси прогнозування із нормалізованими значеннями тестових даних

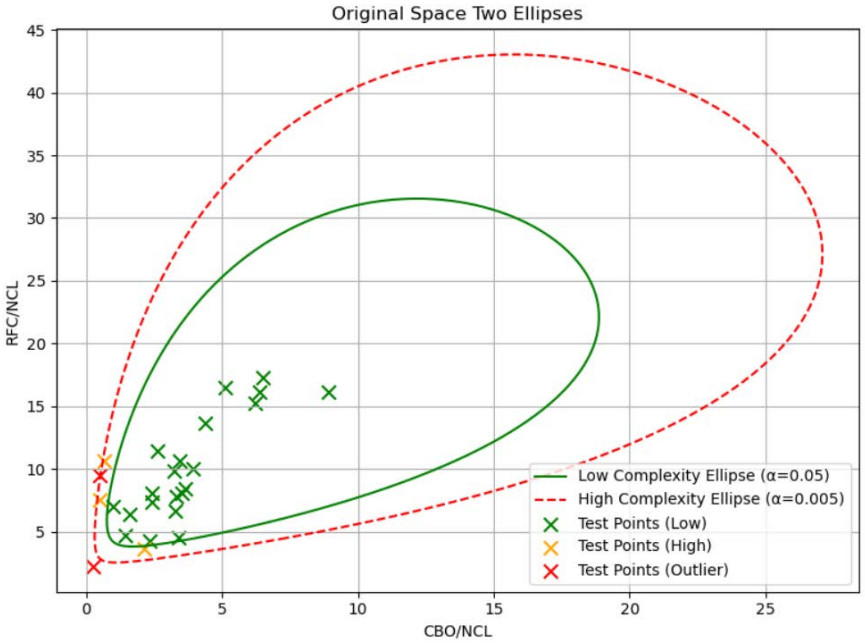

Рисунок Г.6 – Трансформовані еліпси прогнозування із значеннями тестових даних

5. Експорт очищених даних

Після завершення аналізу можна експортувати очищену вибірку.

– Натисніть кнопку «Export Cleaned Data».
– Виберіть директорію для збереження.
– Дані будуть збережені у файл Cleaned_Data.xlsx.

ДОДАТОК Д – ПРОГРАМА І МЕТОДИКА ВИПРОБУВАНЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ СКЛАДНОСТІ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОЄКТУВАННЯ JAVA-ЗАСТОСУНКІВ НА ОСНОВІ ЕЛІПСІВ ПРОГНОЗУВАННЯ З ВИКОРИСТАННЯМ МЕТРИК RFC ТА CBO

1. Об’єкт випробування

Об'єктом випробування є програмне забезпечення «Ellipse Predictor», призначене для оцінювання складності об’єктно-орієнтованого проектування Java-застосунків. Програмний засіб реалізує статистичну модель виявлення аномалій та класифікації складності з використанням метрик CBO/NCL та RFC/NCL.

2. Мета випробування

Метою випробування програмного забезпечення «Ellipse Predictor» 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 « Ellipse Predictor » 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 « Ellipse Predictor » 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 “Ellipse Predictor”, призначеного для оцінювання складності об’єктно-орієнтованого проектування Java-застосунків, необхідно забезпечити виконання таких етапів:

– Завантаження вхідних даних з Excel-файлу. Перевірка коректності завантаження метрик CBO/NCL та RFC/NCL з файлів формату .xlsx.

– Попередня обробка даних. Перевірка автоматичного видалення записів з нульовими значеннями.

– Нормалізація даних. Оцінка коректності застосування багатовимірної Box–Cox нормалізації.

– Виявлення та видалення викидів. Перевірка ідентифікації аномальних значень за допомогою відстані Махаланобіса та F-критерію з ітеративним очищенням вибірки.

– Побудова моделі та двух еліпсів. Перевірка коректності формування еліпсів низької та високої складності.

– Класифікація окремих точок. Оцінка правильності віднесення проєктів до класів Low, High або Outlier.

– Аналіз тестової вибірки. Перевірка коректності прогнозування та візуалізації результатів для тестових даних.

– Експорт результатів. Перевірка збереження результатів аналізу у форматі Excel з коректною 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 з Excel-файлу	Дані CBO/NCL та RFC/NCL успішно завантажено
2	Попередня фільтрація даних	Записи з нульовими значеннями видалено
3	Нормалізація даних	Дані коректно нормалізовано методом Box–Cox
4	Видалення викидів	Аномальні значення ідентифіковано та видалено
5	Побудова еліпсної моделі	Сформовано еліпси низької та високої складності
6	Класифікація тестових даних	Точки коректно класифіковано
7	Візуалізація результатів	Побудовано графіки у нормалізованому та оригінальному просторі
8	Експорт результатів	Дані успішно збережено у форматі Excel
9	Обробка некоректних даних	Виводиться відповідне повідомлення про помилку