МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

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

«Допущений до захисту»
Завідувач кафедри
_____ проф. Приходько С.Б.
«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти «магістр»

на тему: Удосконалення математичної моделі для оцінювання розміру застосунків мовою TypeScript за рахунок додаткової метрики та програма для її реалізації

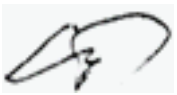
Виконав: студент групи 6151мз
Дімов В. С.
(підпис, ПІБ)

Керівник роботи:
доктор філософії
(посада, науковий ступень вчене звання)
Трухов А. С.
(підпис, ПІБ)

Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова

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



«ЗАТВЕРДЖУЮ»
Гарант освітньої програми
_____проф. С.Б.Приходько
(підпис)
« 27 » жовтня 2025 р.

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

Студенту Дімову Віталію Сергійовичу
(Прізвище, ім'я, по батькові)

1. Тема роботи: Удосконалення математичної моделі для оцінювання розміру застосунків мовою TypeScript за рахунок додаткової метрики та програма для її реалізації

Керівник роботи доктор філософії Трухов А.С.

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 1. Тема. 2. Актуальність теми. 3. Мета та завдання дослідження. 4. Об'єкт та предмет дослідження. 5. Наукова новизна та практичне значення одержаних результатів. 6. Особистий внесок здобувача 7. Апробація результатів досліджень та публікації 8. Огляд загальновідомих рішень оцінювання та аналіз їх обмежень. 8. Огляд спеціалізованих моделей для TypeScript. 9. Результати збору емпіричних даних. 10. Логарифмічна нормалізація даних. 11. Виявлення та вилучення викидів. 12. Побудова нелінійної регресійної моделі та порівняння результатів. 13. Діаграма варіантів використання веб-застосунку. 14. Програмна реалізація: інтерфейс користувача для прогнозування SLOC. 15. Програмна реалізація: аналітичний модуль адміністратора та звітність по моделі. 15. 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 _____ Дімов В.С.
(підпис) (ПІБ)

Керівник роботи _____ Трухов А.С.
(підпис) (ПІБ)

РЕФЕРАТ

Дімов Віталій Сергійович

«Удосконалення математичної моделі для оцінювання розміру застосунків мовою TypeScript за рахунок додаткової метрики 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: 134 стор., 19 табл., 24 рис., 27 використаних джерел, 5 додатків.

Актуальність теми роботи: необхідність підвищення точності прогнозування обсягу вихідного коду застосунків мовою TypeScript на ранніх етапах проєктування для ефективного планування ресурсів та бюджету розробки.

Мета та завдання дослідження. Метою є підвищення достовірності оцінювання розміру програмного забезпечення мовою TypeScript шляхом врахування структурної складності системи. Завдання дослідження - проаналізувати існуючі методи та моделі оцінювання розміру ПЗ; виконати збір статичних метрик коду на репрезентативній вибірці відкритих проєктів на TypeScript; розробити удосконалену нелінійну регресійну модель із врахуванням метрики TR; провести порівняльний аналіз точності розробленої моделі з існуючими рішеннями; розробити програму для автоматизації розрахунків.

Об’єктом дослідження є процес оцінювання розміру програмних застосунків, розроблених мовою TypeScript.

Предметом дослідження є математичні моделі оцінювання розміру програмного коду TypeScript-застосунків.

Методи дослідження. У вирішенні поставлених завдань використано методи системного аналізу, теорії ймовірностей, математичної статистики.

Наукова новизна одержаних результатів. Удосконалено нелінійну регресійну модель оцінювання розміру ПЗ мовою TypeScript шляхом інтеграції додаткової метрики загальної кількості зв’язків між класами TR. На основі логарифмічної нормалізації даних та вилучення багатовимірних викидів

побудовано нелінійну регресійну модель, яка, на відміну від існуючих лінійних та однофакторних рішень, враховує вплив структурної зв’язності на обсяг коду. Це дозволило підвищити коефіцієнт детермінації моделі до рівня 0,8995.

Практичне значення одержаних результатів. Розроблено програмне забезпечення у формі вебзастосунку на базі фреймворку Django, яке автоматизує процес прогнозування обсягу коду SLOC. Програма підтримує можливість завантаження нових даних для перенавчання моделі, що дозволяє актуалізувати прогнози без зміни вихідного коду системи.

Апробація результатів досліджень. Результати досліджень були оприлюднені на VI Міжнародній науково-практичній інтернет-конференції «Інформаційні технології: моделі, алгоритми, системи (ITMAS – 2025)» (м. Миколаїв).

Публікації. Результати роботи викладені у 1 публікації, а саме: тезах доповіді у матеріалах міжнародної науково-практичної конференції.

Ключові слова: оцінювання розміру ПЗ, TypeScript, нелінійна регресія, метрика TR, прогнозування, об’єктно-орієнтоване проєктування, SLOC.

ABSTRACT

Vitalii Dimov

«Improving the mathematical model for estimating the size of applications in TypeScript through an additional metric and a program for its implementation»

The qualification work in obtaining a master’s degree in specialty 121 - "Software Engineering". Admiral Makarov National University of Shipbuilding. Mykolayiv, 2025.

The qualification work is presented on the 134 pages of typewritten text, contains 19 tables, 24 figures, 5 appendices and 27 references.

Relevance of the topic of the work: the need to increase the confidence of forecasting the source code size of applications in TypeScript at the early design stages for effective resource and development budget planning.

The goal and objectives of the study. The goal of the study is to increase the confidence of software size estimation in TypeScript by taking into account the structural complexity of the system. The objectives of the study: to analyze existing regression models to estimate the software size; to collect static code metrics from a representative sample of open-source TypeScript projects; to improve the non-linear regression model for estimating the size of TypeScript applications based on the TR metric; to perform a comparative analysis of the developed model’s accuracy; to develop the software for estimating the size of applications in TypeScript.

The object of the study is the process of estimating the size of software applications developed in the TypeScript language.

The subject of study is the mathematical models for estimating the source code size of TypeScript applications.

The methods of the study. The methods of system analysis, probability theory, mathematical statistics, and regression analysis are used in solving the problems.

The scientific novelty of obtained results. The non-linear regression model for software size estimation in TypeScript is improved by integrating an additional metric – the total number of relationships between classes TR. Based on the normalizing transformation in the decimal logarithm form and outlier removal, a multi-factor model was constructed, which, in comparison to existing linear and single-factor solutions,

accounts for the impact of structural connectivity on code size, allowing to increase the coefficient of determination to the level of 0.8995.

The practical significance of obtained results is that the software in the form of a web application based on the Django framework is developed, which automates the source code size forecasting process and allows for model retraining when new data is uploaded.

Approbation of study results. The study results were presented at the VI International scientific and practical Internet-conference "Information technologies: models, algorithms, systems (ITMAS – 2025)".

Publications. The study results were published in one publication: the materials of the international scientific and practical Internet-conference.

Keywords: software size estimation, TypeScript, non-linear regression, TR metric, Django, forecasting, object-oriented design, SLOC.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 10

ВСТУП 11

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА МОДЕЛЕЙ ОЦІНЮВАННЯ РОЗМІРУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОВОЮ TYPESCRIPT 15

1.1 Особливості розробки програмних застосунків мовою TypeScript та їх вплив на метрики коду 15

1.2 Аналіз методів оцінювання розміру та ефективності регресійних моделей 17

1.3. Обґрунтування доцільності удосконалення моделі шляхом введення додаткового фактора 19

1.4 Висновки 22

2 УДОСКОНАЛЕННЯ НЕЛІНІЙНОЇ РЕГРЕСІЙНОЇ МОДЕЛІ ДЛЯ ПРОГНОЗУВАННЯ РОЗМІРУ ЗАСТОСУНКІВ, МОВОЮ TYPESCRIPT 24

2.1 Виявлення мультиколінеарності та перевірка значущості нової метрики 24

2.2 Нормалізація даних, та очищення від викидів 29

2.3 Розробка нелінійної регресійної моделі для прогнозування розміру застосунків TypeScript 34

2.4 Порівняння розробленої моделі з попередніми рішеннями 40

2.5 Висновок 42

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНКИ РОЗМІРУ ЗАСТОСУНКІВ МОВОЮ TypeScript 43

3.1 Постановка задачі визначення функціональних та не функціональних вимог 43

3.2 Обґрунтування вибору платформи реалізації та архітектури 45

3.3 Розробка діаграми варіантів використання 48

3.4 Побудова діаграм діяльності для ключових сценаріїв використання 54

3.5 Розробка моделі перехоідв станів ПЗ оцінювання розміру застосунків мовою TypeScript 57

3.5 Обґрунтування вибору засобів розробки та технологічного стека проєкту 58

3.7 Реалізація моделей збереження даних засобами Django ORM..... 62

3.8 Випробування ПЗ для оцінювання розміру застосунків TypeScript 66

3.9 Висновки 77

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВІД РОЗРОБКИ ТА ВПРОВАДЖЕННЯ ВЕБ-ДОДАТКУ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ЗАСТОСУНКІВ МОВОЮ TYPESCRIPT 80

4.1 Вступ..... 80

4.2 Розрахунок витрат на створення та впровадження програмного забезпечення81

4.3 Розрахунок економічної ефективності розробки програмного забезпечення .. 84

4.4 Висновки 86

5 ОХОРОНА ПРАЦІ 87

5.1 Загальні визначення та актуальні питання охорони праці..... 87

5.2 Аналіз небезпечних і шкідливих факторів при роботі з персональним комп’ютером 90

5.4 Розробка заходів щодо зменшення впливу шкідливих та небезпечних факторів при роботі з персональним комп’ютером..... 97

6 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА 102

6.1 Вступ..... 103

6.2 Види забруднення навколишнього природного середовища 104

6.3 Природоохоронна діяльність підприємств..... 104

6.5 Висновки 108

ВИСНОВКИ..... 109

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 111

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ 114

ДОДАТОК Б МЕТОДИКА ВИПРОБУВАНЬ ПЗ 120

ДОДАТОК В ІНСТРУКЦІЯ КОРИСТУВАЧА 122

ДОДАТОК Г ТЕКСТ ПРОГРАМИ 124

ДОДАТОК Д ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 132

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AAC – Average Attributes per Class;

CBO – Coupling Between Objects;

COCOMO – Constructive Cost Model;

FPA – Function Point Analysis;

NOC – Number Of Classes;

OOP – Object-Oriented Programming;

PRED – Percentage of prediction;

SLOC – Source lines of code;

SMD – Squared Mahalanobis Distance;

SVG – Scalable Vector Graphics;

TR - Total Relationships;

TS – TypeScript (мова програмування);

UML – Unified Modeling Language.

ВСТУП

Актуальність теми. В умовах цифрової трансформації економіки роль програмного забезпечення як ключового бізнес-активу невпинно зростає. Це, у свою чергу, призводить до збільшення масштабів і складності програмних проєктів, що викликає сумнів щодо ефективності традиційних, інтуїтивних підходів до управління [1]. За таких умов, фундаментальною передумовою успішності будь-якого програмного проєкту, є здатність до точного прогнозування його ключових атрибутів.

Центральне місце серед цих прогностичних завдань посідає оцінювання розміру програмного продукту, оскільки ця характеристика слугує базовим вхідним параметром для більшості алгоритмічних моделей розрахунку вартості, зокрема для класичної моделі COCOMO [2]. Неточність на цьому початковому етапі провокує ефект «доміно»: виникають проблеми з розподілом ресурсів, відбувається неконтрольоване розширення обсягу робіт (scope creep), і, як наслідок, проєкт стикається із загрозою зриву термінів та виходу за межі бюджету. Таким чином, розробка та вдосконалення надійних, математично обґрунтованих методів оцінювання розміру ПЗ залишається одним з ключових та актуальних науково-практичних завдань сучасної інженерії програмного забезпечення.

Проте варто зазначити, що універсальні моделі, такі як COCOMO, демонструють обмежену точність при застосуванні до сучасних програмних проєктів, реалізованих із використанням високорівневих мов програмування. Це пов’язано з тим, що вони не враховують унікальні синтаксичні та парадигматичні особливості кожної мови, де один і той самий рядок коду може мати кардинально різну функціональну складність. Ця проблема зумовила методологічний зсув у бік розробки моделей, спеціалізованих під конкретні мови програмування. Яскравим прикладом є створення удосконалених нелінійних регресійних моделей для Java-застосунків, що було продемонстровано у роботах Приходько С.Б. [3].

Розробка подібних спеціалізованих моделей є не менш актуальною і для мови TypeScript. Це зумовлено її статусом не просто популярної, а й стратегічно важливої технології у сучасній веб-розробці. Будучи строго типізованою надбудовою над JavaScript, вона ефективно вирішує фундаментальні недоліки свого попередника, дозволяючи створювати більш надійний, масштабований та легкий у підтримці код, що є необхідною умовою для систем корпоративного рівня. Емпіричним підтвердженням її домінуючого положення слугують останні дані: згідно з провідними опитуваннями, такими як Stack Overflow Survey [4], TypeScript стабільно посідає друге місце за поширеністю, поступаючись лише JavaScript, який він, по суті, розширює та доповнює.

Суттєвим недоліком класичних підходів є те, що вони базуються на припущенні про нормальний закон розподілу вхідних даних. Однак емпіричні метрики програмного забезпечення зазвичай мають не гаусівський розподіл із вираженою асиметрією. Застосування лінійних методів до таких даних без попередньої обробки призводить до систематичних похибок, що робить прогнози ненадійними.

Перші кроки у напрямку створення спеціалізованих моделей для TypeScript з урахуванням нелінійності даних вже були зроблені; зокрема, було розроблено та проаналізовано нелінійні регресійні моделі, що враховують такі метрики, як кількість класів та атрибутів. Проте, як показало дослідження [5], навіть ці прогресивні моделі мають спільне обмеження – вони ігнорують структурну складність програмної системи, що виражається у зв’язках між її компонентами. Це призводить до зниження точності прогнозування для складних, сильно зв’язаних систем.

Таким чином, актуальною науково-практичною задачею є удосконалення математичної моделі для оцінювання розміру застосунків мовою TypeScript за рахунок врахування додаткової метрики структурної складності Total Relationships, що й стало предметом даного дослідження.

Мета і завдання дослідження. Метою кваліфікаційної роботи є підвищення точності оцінювання розміру програмних застосунків, розроблених мовою

TypeScript, шляхом удосконалення існуючої математичної моделі за рахунок врахування метрики загальної кількості зв’язків між програмними сутностями.

Для досягнення поставленої мети було визначено наступні завдання:

- проаналізувати існуючі методи та моделі оцінювання розміру програмного забезпечення, виявити їх переваги та недоліки;
- виконати збір та аналіз статичних метрик коду (кількість класів, атрибутів, зв’язків та рядків коду) на репрезентативній вибірці відкритих проєктів, розроблених мовою TypeScript;
- розробити удосконалену регресійну модель для прогнозування розміру застосунків, що додатково враховує метрику TR;
- провести експериментальну перевірку розробленої моделі та виконати порівняльний аналіз її точності з лінійною регресійною моделлю;
- провести експериментальну перевірку розробленої моделі та виконати порівняльний аналіз її точності з існуючою моделлю;
- розробити програму для оцінювання розміру застосунків мовою TypeScript, використовуючи побудовану удосконалену регресійну модель.

Об’єкт дослідження – процес оцінювання розміру програмних застосунків, розроблених з використанням мови TypeScript.

Предмет дослідження – регресійна модель оцінювання розміру програмного коду TypeScript-застосунків.

Методи дослідження. Для розв’язання поставлених завдань було використано методи системного аналізу для огляду предметної області, методи математичної статистики та теорії ймовірностей для обробки емпіричних даних, кореляційно-регресійний аналіз для побудови математичної моделі, а також методи об’єктно-орієнтованого програмування для розробки програмного продукту.

Наукова новизна одержаних результатів полягає в удосконаленні математичної моделі оцінювання розміру застосунків мовою TypeScript, яка, на відміну від існуючих, використовує як один із предикторів метрику загальної кількості зв’язків між класами TR. Це дозволяє підвищити точність прогнозу за

рахунок більш повного врахування структурної складності програмної системи на ранніх етапах її життєвого циклу.

Практичне значення одержаних результатів полягає у розробці удосконаленої математичної моделі та програмного засобу, що її реалізує, які дозволяють виконувати більш точне прогнозування розміру TypeScript-застосунків на ранніх етапах життєвого циклу.

Результати роботи призначені для використання керівниками проєктів, системними аналітиками та розробниками для отримання обґрунтованих оцінок обсягу майбутніх програмних систем. Впровадження запропонованої моделі та програмного засобу у практику інженерії програмного забезпечення дозволить підвищити якість проєктного планування, оптимізувати розподіл ресурсів та знизити ризики, пов’язані з неточною оцінкою трудомісткості розробки.

Апробація результатів досліджень Основні положення та результати досліджень, викладені у кваліфікаційній роботі, пройшли апробацію на VI Міжнародній науково-практичній інтернет-конференції «Інформаційні технології: моделі, алгоритми, системи (ITMAS – 2025)» (м. Миколаїв, 16-17.11.2025 р.)

Публікації На основі результатів, отриманих у рамках кваліфікаційної роботи, було опубліковано тези доповіді на конференції [5].

Ключові слова: оцінювання розміру ПЗ; TypeScript; нелінійна регресійна модель; метрики програмного забезпечення; коваріаційна матриця.

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА МОДЕЛЕЙ ОЦІНЮВАННЯ РОЗМІРУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОВОЮ TYPESCRIPT

1.1 Особливості розробки програмних застосунків мовою TypeScript та їх вплив на метрики коду

Сучасний стан індустрії розробки програмного забезпечення характеризується стрімким зростанням складності веб-застосунків та переходом до кросплатформних рішень. В умовах жорсткої ринкової конкуренції точність оцінювання трудомісткості та розміру програмного продукту стає критичною передумовою успіху проєкту. Як зазначає Соммервіль, помилки на етапі початкового планування можуть призвести до перевищення бюджету на 30-40% або навіть до повного провалу проєкту [6].

Серед сучасних інструментальних засобів розробки особливе місце посідає мова програмування TypeScript. Згідно з офіційною документацією Microsoft, TypeScript позиціонується як типізована надбудова (superset) над JavaScript, що компілюється в чистий JavaScript. Головною особливістю мови, яка суттєво впливає на процес розробки та, відповідно, на метрики коду, є наявність суворої статичної типізації та повноцінна підтримка об’єктно-орієнтованого підходу ООР – Object-Oriented Programming.

Окрім технічних аспектів транспіляції, варто враховувати масштаб проєктів. Завдяки потужній екосистемі та статичному аналізу коду, TypeScript став стандартом де-факто для розробки великих корпоративних систем Enterprise-level, де над одним продуктом працюють десятки розробників. Саме в таких проєктах, ціна помилки при оцінюванні трудомісткості є найвищою, а потреба в точних метриках – критичною. Стабільне зростання попиту на дану технологію, підтверджується глобальними дослідженнями ринку праці та опитуваннями розробників, що наочно демонструє статистика використання мов програмування.

Актуальність вибору саме представленої мови для дослідження підтверджується статистичними даними. За результатами щорічного опитування розробників Stack Overflow Developer Survey 2025 (рис 1.1) [1] TypeScript стабільно входить в топ найпопулярніших технологій, випереджаючи класичні мови, такі як Java або C#, і свідчить про те, що значна частина сучасного комерційного коду пишеться саме цією мовою, а отже, потреба в точних моделях для її оцінювання є високою.

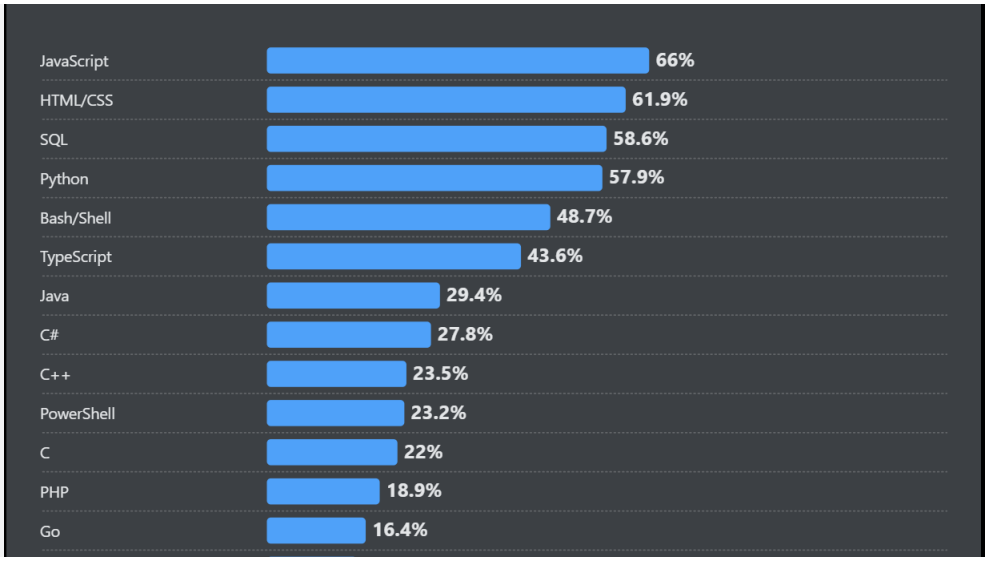


Рисунок 1.1 – Рейтинг популярності мов програмування 2025 р. [1]

Специфіка оцінювання розміру застосунків мовою TypeScript визначається її проміжним положенням між динамічними скриптовими мовами (JavaScript, Python) та класичними статичними мовами (Java, C#). З одного боку, на відміну від Python або чистого JavaScript, TypeScript вимагає від розробника створення додаткових синтаксичних конструкцій – інтерфейсів, аліасів типів (Type Aliases), дженериків та декораторів. Названі елементи, є необхідними для забезпечення архітектурної цілісності та надійності системи, що збільшує час проєктування.

З іншого боку, TypeScript характеризується значно меншою кількістю шаблонного коду (так званого «boilerplate code») порівняно з Java або C#. Завдяки потужним механізмам виведення типів (Type Inference) та лаконічному синтаксису,

розробнику не потрібно явно вказувати типи там, де компілятор може визначити їх автоматично [6].

Подібний баланс між структурною строгістю та лаконічністю, створює специфічну проблему для прогнозування. Пряме використання моделей, розроблених для інших мов, призводить до систематичних похибок, тому що моделі для Java схильні завищувати оцінку розміру оскільки один клас в TypeScript зазвичай потребує менше коду, а моделі для динамічних мов – занижувати її, ігноруючи витрати часу на типізацію [7].

Для TypeScript доцільно використовувати підхід, що базується на об’єктно-орієнтованих метриках (кількість класів, атрибутів, методів), оскільки ця мова має повноцінну підтримку класової ієрархії, але потребує побудови власної, адаптованої регресійної моделі.

1.2 Аналіз методів оцінювання розміру та ефективності регресійних моделей

Оцінювання трудомісткості розробки програмного забезпечення є критичним етапом планування, від якого залежить бюджет та терміни реалізації проєкту. Класичним стандартом в даній галузі, є конструктивна модель вартості COCOMO, запропонована Баррі Боемом [9]. Представлена модель, дозволяє розрахувати витрати часу в людино-місяцях, базуючись на розмірі програмного продукту, вираженому в тисячах рядків коду KLOC. Однак, головною проблемою застосування COCOMO є те, що вона вимагає знати точне значення KLOC ще до початку написання коду. На практиці, для отримання значення показника, часто застосовують спрощені лінійні підходи, припускаючи, що розмір системи зростає пропорційно до кількості її елементів (наприклад, кількості класів). Тобто, використовується:

$$Y = a \cdot X + b, \tag{1.1}$$

де Y – залежна змінна розмір коду; X – незалежна змінна, наприклад, кількість класів; a, b – коефіцієнти моделі.

Проте, специфіка сучасних об’єктно-орієнтованих мов, зокрема TypeScript, спростовує ефективність лінійного підходу. Збільшення кількості класів в системі, не означає лінійного збільшення кількості коду, тому що зі зростанням кількості сутностей NOC, експоненційно зростає складність їх взаємодії, потреба в «інфраструктурному» коді, конфігураціях та описах типів.

Як зазначають дослідники метрик об’єктно-орієнтованого дизайну Чидамбер і Кемерер, реальна залежність між архітектурними метриками та розміром коду має складніший характер [10]. Лінійні моделі систематично дають похибку на великих проєктах, так як вони недооцінюють складність інтеграції компонентів, що призводить до заниження бюджету та ризиків зриву дедлайнів.

Сучасні дослідження фокусуються на розробці та вдосконаленні нелінійних регресійних моделей. Використання подібного підходу вже довело свою ефективність в роботах, присвячених оцінюванню застосунків мовою TypeScript, в роботі Чебакова Д.В. [5] було досліджено однофакторну модель, де предиктором виступала кількість класів. Автор довів, що застосування степеневої функції вигляду:

$$Y = a \cdot X^b, \tag{1.2}$$

в поєднанні з логарифмічною нормалізацією даних дозволяє значно точніше описати варіацію розміру коду порівняно з лінійною регресією.

Подальший розвиток даний підхід отримав в дослідженні Павлюк К.А., де модель було розширено до двох факторів, кількості класів (X_1) та середньої кількості атрибутів на клас (X_2) [6]. Загальний вигляд такої мультиплікативної моделі описується рівнянням:

$$Y = a \cdot X_1^{b_1} \cdot X_2^{b_2}. \tag{1.3}$$

Нововведення дозволило підвищити коефіцієнт детермінації R^2 та врахувати внутрішню насиченість класів. Результати даних досліджень підтверджують, що для TypeScript, який поєднує строгу типізацію з лаконічністю синтаксису, найбільш адекватним є використання нелінійних багатофакторних моделей. Вони здатні врахувати ефект «насичення», коли додавання нових функціональних блоків призводить до непропорційного зростання кодової бази через необхідність підтримки цілісності типів та архітектурних зв'язків.

1.3. Обґрунтування доцільності удосконалення моделі шляхом введення додаткового фактора

Аналіз робіт попередників [5, 6] та сучасних досліджень в галузі метричного аналізу дозволяє зробити висновок, що існуючі регресійні моделі досягли певної межі точності, яка обмежується набором вхідних факторів. Моделі, що базуються лише на кількості класів (X_1) та їх внутрішній насиченості (X_2), ефективно описують «статичний» обсяг коду, проте ігнорують витрати на інтеграцію компонентів системи.

Теоретично, для покращення моделі та подальшого зменшення похибки прогнозування проведено детальний аналіз потенційних додаткових метрик, які могли б увійти до складу моделі.

Функціональні точки FPA. Мовно-незалежна метрика, що оцінює розмір системи на основі функціональності, яку вона надає користувачеві зображено на рисунку 1.2.

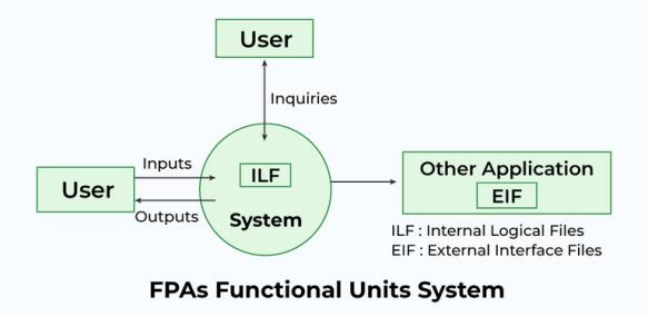


Рисунок 1.2 - Function Point Analysis [11]

FPA є чудовим інструментом для оцінки на етапі вимог, але важко піддається автоматизованому видобутку з існуючого коду. Для її розрахунку потрібна детальна технічна документація та суб'єктивна експертна оцінка, що робить неможливим збір великого репрезентативного датасету з відкритих репозиторіїв TypeScript для навчання регресійної моделі.

Метрики складності коду, такі як, цикломатична складність МакКейба, метрики Холстеда оцінюють внутрішню (алгоритмічну) складність окремого методу чи функції, базуючись на аналізі шляхів потоку управління (рис.1.3) або кількості операторів/операндів в коді.

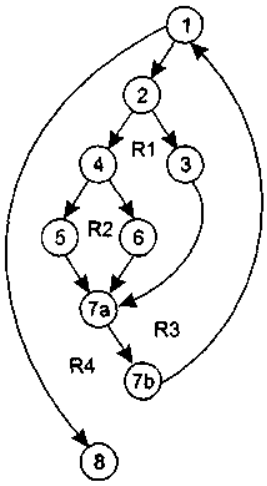


Рисунок 1.3 - Потоковий граф управління [12]

Представлені метрики вимагають наявності вже написаного коду тіла методів або розроблених схем алгоритмів, що робить їх недоступними на ранніх етапах проєктування системи, де оцінка розміру є найбільш затребуваною.

Оскільки жодна з традиційних метрик не задовольняє вимогам, щодо доступності великої кількості даних для побудови моделі, або для отримання яких, треба детальна архітектура кожного алгоритму, робить неможливим їх використання на ранніх етапах розробки програмного забезпечення. Натомість, було обрано метрику, що належить до набору об'єктно-орієнтованих метрик Чидамбера та Кемеріера [10]. Кількість зв'язків між класами CBO визначена як найбільш оптимальний третій фактор (X_3).

В контексті мови TypeScript, яка широко використовується для побудови модульних архітектур, критичним фактором трудомісткості стає структурна складність, яка виражається через метрику зв’язності CBO.

Висока зв’язність між класами призводить до нелінійного зростання обсягу коду, що не фіксується простим підрахунком методів. Даний фактор, добре ілюструється на прикладі структурних патернів проєктування, які часто використовуються в TypeScript-проєктах. Розглянемо, наприклад, патерн «Фасад» (Facade), який надає простий інтерфейс до складної системи класів (рис. 1.4).

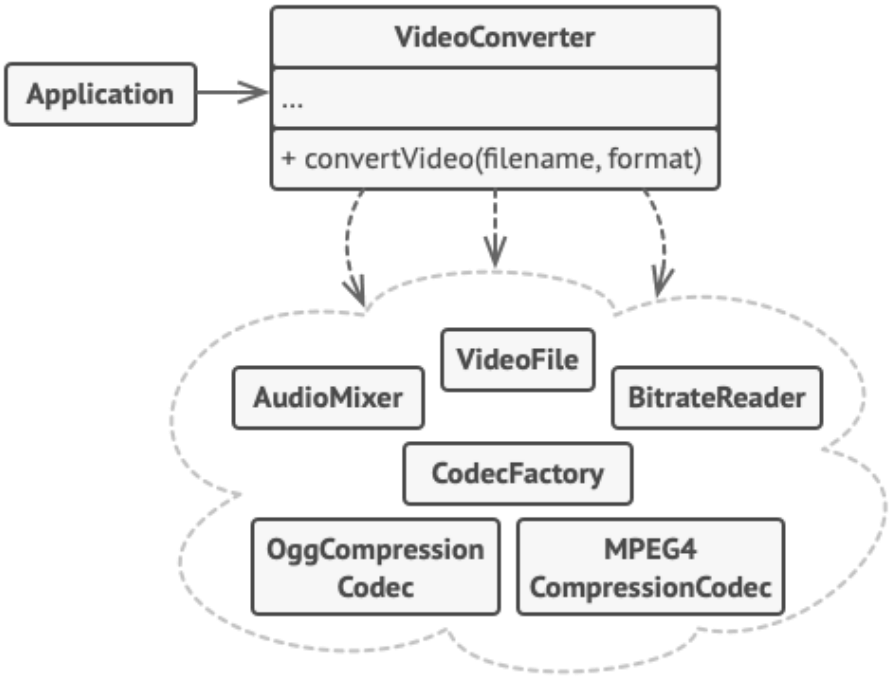


Рисунок 1.4 – Приклад високої зв’язності у структурному патерні проєктування «Фасад» [13]

Як видно з рисунка 1.4, центральний клас «Фасаду» має зв’язки з багатьма іншими компонентами системи. З точки зору метрики «Кількість класів» NOC (X_1) це лише одиниця. З точки зору «Середньої кількості атрибутів» AAC (X_2) – він може бути досить простим, лише делегуючи виклики. Проте, для реалізації даної структури в TypeScript розробнику необхідно прописати імпорти для кожного

залежного модуля (import ...). Реалізувати ін’єкцію залежностей через конструктор Dependency Injection, узгодити типи даних між різними підсистемами.

Ігнорування фактора «інфраструктурного коду» призводить до того, що модель систематично занижує оцінку розміру для складних систем. Тому, доцільно ввести в модель третій фактор – загальну кількість залежностей TR, позначивши його як X_3.

Представлену гіпотезу підтверджує аналіз досліджень, які були спрямовані на розробку моделей для прогнозування моделей інших об’єктно-орієнтованих мов програмування. Одні з найкращих моделей [6] показують, що перехід до трьохфакторної моделі дозволяє врахувати синергетичний ефект зростання коду у великих системах. Враховуючи, що TypeScript є повноцінною об’єктно-орієнтованою мовою, адаптація успішного досвіду є логічним та обґрунтованим кроком. Удосконалення математичного апарату полягатиме в розробці нелінійної регресійної моделі, яка, на відміну від попередніх рішень, враховуватиме метрику зв’язності, що дозволить підвищити точність прогнозування для архітектурно складних проєктів.

1.4 Висновки

В даному розділі було проведено комплексний аналіз сучасних методів та моделей оцінювання розміру програмного забезпечення з особливою увагою до застосунків, що розробляються мовою TypeScript, та обґрунтовано необхідність удосконалення існуючих регресійних моделей. Встановлено, що актуальність дослідження зумовлена стрімким зростанням складності великомасштабних корпоративних систем на TS, де точність початкової оцінки розміру є критичною.

Проведений аналіз підтвердив, що специфіка мови, яка поєднує сувору статичну типізацію з лаконічністю об’єктно-орієнтованого синтаксису, вимагає використання нелінійних багатофакторних моделей. Лінійні підходи та пряма адаптація моделей, розроблених для інших мов, призводять до систематичних

похибок, оскільки вони не здатні адекватно врахувати нелінійне зростання складності коду зі збільшенням кількості сутностей.

На основі вивчення робіт попередників [5, 6] підтверджено ефективність використання двох основних факторів – кількості класів X_1 та середньої кількості атрибутів на клас X_2. В ході дослідження можливостей покращення існуючих моделей було проведено критичний аналіз типових метрик, таких як FPA та цикломатична складність. Проте, зроблено висновок, що дані метрики є непридатними як додаткові фактори через їх недоступність на ранніх етапах проєктування, коли оцінка розміру є найбільш затребуваною, а також, через неможливість їх автоматизованого збору для формування великої вибірки даних.

Натомість, для подолання представленого обмеження та подальшого підвищення точності, обґрунтовано необхідність введення третього фактора X_3 – TR. Нова метрика, як частина набору об’єктно-орієнтованих метрик [10], була обрана тому, що вона безпосередньо вимірює структурну складність і є доступною на етапі архітектурного проєктування, що дає можливість застосовувати модель до початку написання коду та заздалегідь оцінити розмір застосунку.

Не менш важливим фактором при виборі метрики була можливість отримати значну кількість даних для побудови моделі. Показник TR можна отримати шляхом аналізу вихідного коду вже готових проєктів, а завдяки великій кількості проєктів з відкритим кодом, які дозволяють використання та аналіз вихідних текстів, процес збору даних значно спрощується.

В результаті виконання розділу, сформовано теоретичне підґрунтя для проведення дослідження в сфері розробки вдосконаленої нелінійної регресійної моделі для прогнозування розміру TypeScript-застосунків, шляхом врахування нової метрики X_3 кількість зв’язків між класами. Дана метрика повинна врахувати недоліки існуючих моделей, та врахувати складність зв’язків між класами.

2 УДОСКОНАЛЕННЯ НЕЛІНІЙНОЇ РЕГРЕСІЙНОЇ МОДЕЛІ ДЛЯ ПРОГНОЗУВАННЯ РОЗМІРУ ЗАСТОСУНКІВ, МОВОЮ TYPESCRIPT

В першому розділі було визначено, що класичні лінійні методи часто не здатні адекватно описати складні взаємозв'язки між структурними елементами програми та її підсумковим розміром. Тому, в наступному розділі увага зосереджена на побудові удосконаленої нелінійної регресійної моделі. В якості ключових предикторів (незалежних змінних), які формуються на етапі проєктування архітектури, обрано:

1. X_1 (NOC) – загальну кількість класів;
2. X_2 (AAC) – середню кількість атрибутів на один клас;
3. X_3 (TR) – кількість залежностей між класами.

Метою наступного розділу є розробка математичної моделі, яка дозволяє прогнозувати розмір проєкту Y SLOC враховуючи нелінійну природу даних.

2.1 Виявлення мультиколінеарності та перевірка значущості нової метрики

В ході збору даних, було розглянуто близько 80 проєктів. Проте в процесі детального аналізу були відхилені ті з них, що використовували фреймворки, які змінюють синтаксис, такі як React, оскільки подібні інструменти суттєво впливають на спосіб опису логіки застосунку, що призводить до спотворення метрик. Для дослідження були обрані лише ті проєкти, які реалізовані в парадигмі об'єктно-орієнтованого програмування з використанням класів. Подібний перебіг подій зумовлений тим, що TypeScript є надбудовою над JavaScript, яка підтримує як класичний ООП-підхід, так і функціональний стиль, який в представленому дослідженні є нерелевантним.

Для побудови моделі було залишено 53 проєкти з відкритим вихідним кодом, розміщені на платформі GitHub (https://github.com/).

Було проаналізовано декілька утиліт та плагінів, які автоматично аналізують код, написаний мовою TypeScript, та надають програмні метрики. Зокрема, одним з найпопулярніших інструментів є SonarQube. Проте, жоден з досліджених засобів не надає метрики TR, яка є критично важливою для даного дослідження. В зв'язку з цим, було прийнято рішення на користь використання плагіна TypeScript UML Diagrams для середовища VSCode [7], який дозволяє згенерувати діаграми класів на основі вихідного TypeScript-коду (рис. 2.1) та отримати необхідні структурні характеристики, зокрема $X_1(NOC)$, X_2 (AAC), X_3 (TR).

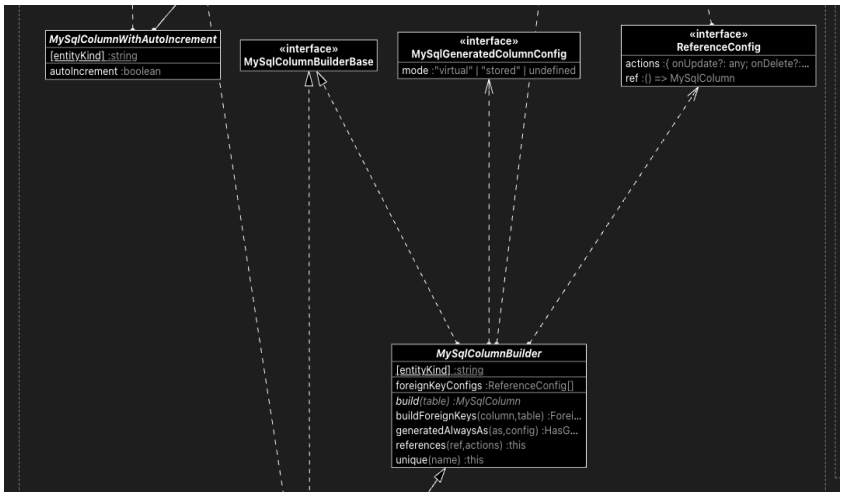


Рисунок 2.1 - Приклад згенерованої діаграми класів

Проте, зазначений плагін забезпечує лише візуалізацію, без надання числових метрик за замовчуванням. Тому, для отримання формалізованих даних було розроблено спеціальний скрипт, який на основі експортованої діаграми класів у форматі SVG автоматично здійснював підрахунок класів, атрибутів та зв'язків між ними. Представлена автоматизація значно спростила процес збору даних та забезпечила необхідну точність при формуванні вибірки для побудови регресійної моделі. Для збору значень залежної змінної Y $(SLOC)$ – кількості рядків вихідного коду було використано утиліту командного рядка cloc [6]. Зазначений інструмент формує детальний звіт для кожного з проєктів, з точним підрахунком кількості рядків коду для кожного типу файлів в проєкті. Отримані результати наведено в таблиці 2.1.

Таблиця 2.1 – Зібрані емпіричні дані з метриками та порахованим SMDx

№	*X_1(NOC)*	*X_2(AAC)*	*X_3(TR)*	*Y(SLOC)*	*SMDx*
1	642	8,02	4751	82548	6,32
2	196	6,95	832	14896	2,71
3	34	2,00	34	1422	0,80
4	144	1,11	574	7578	1,11
5	50	1,00	154	4078	1,35
6	242	1,83	858	8612	0,58
7	575	4,22	4994	71430	5,03
8	143	1,37	410	3873	0,97
9	37	3,43	142	6858	0,49
10	179	4,32	635	13322	0,48
11	1440	2,99	4261	120781	3,21
12	965	5,13	5612	122990	4,67
13	784	8,17	5309	239573	**9,98**
14	2807	2,43	3084	402434	**33,24**
15	43	5,09	296	1742	1,09
16	149	1,56	206	4476	0,92
17	11	2,09	66	511	0,77
18	105	0,99	150	6135	1,30
19	45	3,73	3	307972	7,54
20	91	4,93	424	9939	0,88
21	17	2,59	54	560	0,62
22	48	2,88	95	6330	0,51
23	65	1,54	170	5542	0,95
24	11	0,91	14	62714	1,53
25	69	1,09	178	3924	1,26
26	685	2,31	3350	40455	1,25
27	700	1,88	2406	39035	0,80
28	64	6,47	182	5578	2,49
29	20	1,15	76	1201	1,29
30	213	1,21	587	14025	0,97
31	622	1,66	3570	28617	2,23
32	180	1,48	670	10244	0,81
33	434	2,54	1148	52607	0,20
34	763	2,59	1214	107945	1,38
35	90	2,77	48	6927	0,56

Кінець табл. 2.1

1	2	3	4	5	6
36	1214	1,77	5199	89015	3,74
37	59	2,02	128	270559	5,97
38	39	2,97	103	1987	0,53
39	279	1,47	394	44530	0,80
40	47	1,00	134	2234	1,36
41	151	1,19	175	5340	1,16
42	171	1,67	912	7175	0,74
43	1092	1,93	5221	50279	3,96
44	27	9,96	46	7903	9,10
45	13	7,15	14	2326	3,53
46	39	4,92	130	5680	1,06
47	108	5,24	398	15503	1,09
48	83	6,64	402	13360	2,48
49	15	10,60	26	1858	**10,82**
50	354	2,19	1086	592931	**28,07**
51	1045	4,15	8176	56542	**14,09**
52	119	2,48	430	5923	0,45
53	2730	3,56	7964	52217	**18,77**

Окрім зібраних метрик в таблиці наданий квадрат відстані Махаланобіса (SMDx), обрахунок був зроблений за формулою [8]:

$$d_i^2 = (X_i - \bar{X})^T S_N^{-1} (X_i - \bar{X}), \tag{2.1}$$

де X_i – це вектор координат i-ї багатовимірної спостережуваної точки, $\overline{X}$ – вектор середніх значень, S_N – коваріаційна матриця, яка визначається за формулою [18]:

$$S_N = \frac{1}{N} \sum_{i=1}^{N} \left(X_i - \overline{X}\right)\left(X_i - \overline{X}\right)^T. \tag{2.2}$$

Аналіз емпіричних даних вказує на відхилення їх розподілу від нормального закону. Про це свідчить розрахунок квадрата відстані Махаланобіса: для застосунків № 13, 14, 49, 50, 51, 53 отримані значення перевищують критичний поріг 9,4877 (квантиль розподілу при рівні значущості 0,05).

На основі зібраних даних було побудовано діаграми в розрізі кожної з метрик (рис. 2.2).

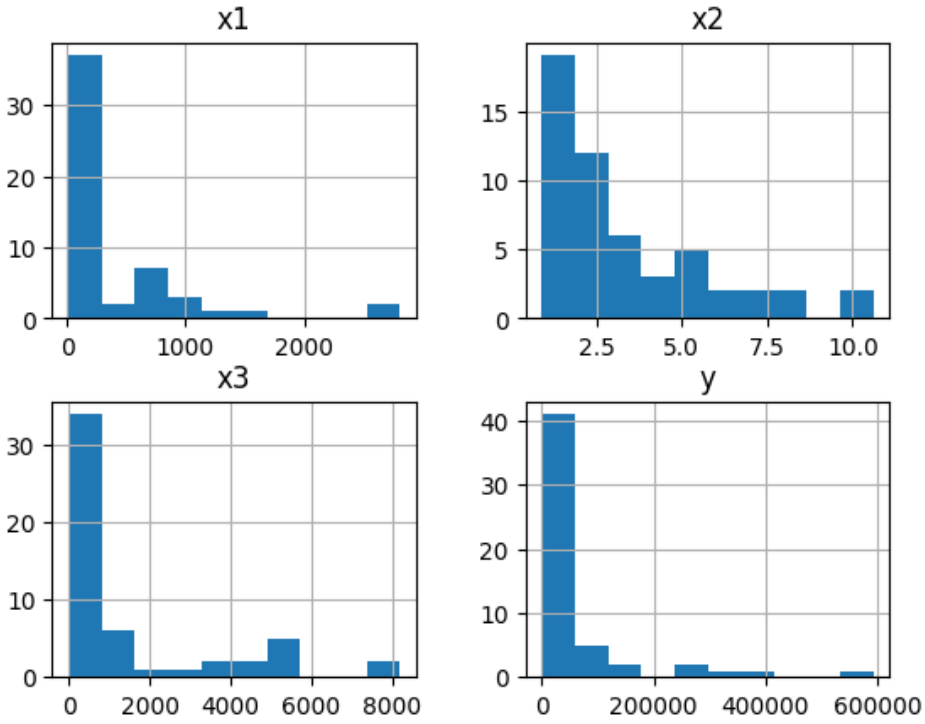


Рисунок 2.2 - Графік розподілення значень метрик

В даній роботі розробляється удосконалена модель з використанням нової метрики X_3 кількість залежностей між класами. Треба впевнитись, що дана метрика є значущою, тому необхідно переконатися у відсутності мультиколінеарності, яка призводить до необґрунтованого зростання стандартних помилок оцінок, що робить регресійну модель нестійкою та надмірно чутливою до найменших змін у вихідних даних, що робить неможливим коректне виокремлення індивідуального впливу кожної метрики на розмір вихідного коду. В результаті, отримана формула стає ненадійним інструментом для прогнозування реальних проєктів [14].

Для кількісної діагностики мультиколінеарності було використано розрахунок факторів інфляції дисперсії Variance Inflation Factor – VIF. Даний показник відображає, в скільки разів дисперсія оцінки коефіцієнта регресії збільшується через кореляцію між предикторами. Загальноприйнятим, в статистичному аналізі, є правило, згідно з яким значення $VIF < 5$ свідчать про

відсутність критичної мультиколінеарності, а значення $VIF > 10$ вказують на серйозну проблему, що вимагає виключення змінної з моделі.

Результати розрахунку VIF для обраних метрик X_1 – кількість класів, X_2 – середня кількість атрибутів, X_3 – кількість залежностей наведено в таблиці 2.2.

Таблиця 2.2 – Значення факторів інфляції дисперсії VIF

Змінна	Значення VIF
X1 (NOC)	2,6545532
X2 (AAC)	1,055238
X3 (TR)	2,6959781

Аналіз отриманих результатів демонструє, що значення факторів інфляції дисперсії VIF для досліджуваних метрик знаходяться в межах допустимого діапазону. Зокрема, для нової метрики X_3 показник становить 2,696, для X_1 – 2,655, а для X_2 – 1,055. Оскільки всі отримані значення є меншими за критичний поріг 5, це свідчить про відсутність мультиколінеарності.

2.2 Нормалізація даних, та очищення від викидів

Підтверджене відхилення емпіричних даних від нормального закону розподілу та наявність аномальних спостережень (викидів), виявлених за відстанню Махаланобіса, роблять неможливою побудову стійкої регресійної моделі на вихідній вибірці. Зібрані дані мають негаусівський розподіл, тому необхідно застосувати нормалізуюче перетворення. Було обрано перетворення на основі десяткового логарифма lg, що обчислюється за формулою (2.3):

$$\psi = \begin{pmatrix} \lg Y \\ \lg X_1 \\ \lg X_2 \\ \lg X_3 \end{pmatrix}. \tag{2.3}$$

Після застосування нормалізуючого перетворення lg розподіл даних наблизився до нормального розподілу (рис. 2.3).

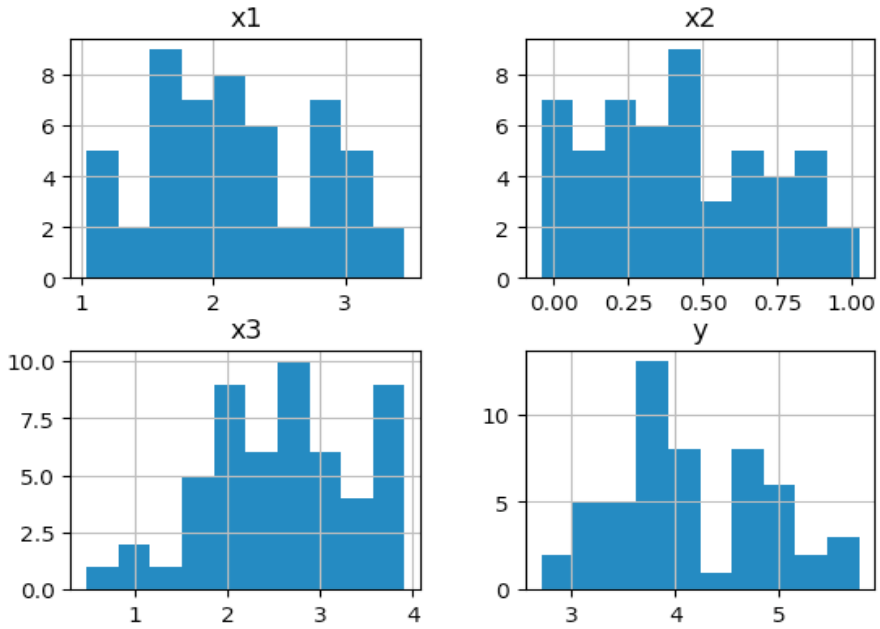


Рисунок 2.3 - Графік розподілення значень нормалізованих метрик

Проте, схожості на графіках не є достовірним доказом нормальності даних, є недостатнім критерієм, тому необхідно провести тест на нормальність даних. Оскільки дані є багатовимірними, застосування звичайних одновимірних тестів є некоректним у випадку векторних величин. Тому, для оцінки багатовимірної нормальності був використаний тест Мардія, 1970 [13], який є класичним та найбільш поширеним критерієм для подібних задач.

Тест Мардія вважається оптимальним для багатовимірної нормальності. Він одночасно оцінює багатовимірну асиметрію та багатовимірний ексцес, що дає повнішу картину відхилення від нормального закону. Формули тесту побудовані на статистиці відстаней між спостереженнями в просторі X, що робить його стійким до змін масштабів. Підходить як для малих, так і для середніх вибірок.

Багатовимірна асиметрія вимірює, наскільки розподіл багатовимірних даних симетричний навколо свого центру. В багатовимірному випадку ми не просто дивимося на окремі змінні, а на всі напрямки одночасно, тобто на форму хмари

точок у p-вимірному просторі. Розраховується багатовимірна асиметрія за формулою:

$$b_{1,p} = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left[(x_i - \bar{x})^T S^{-1}\left(x_j - \bar{x}\right)\right]^3, \quad (2.4)$$

де

n – кількість спостережень (рядків у матриці даних).

p – кількість змінних (стовпців).

x_i – вектор спостереження і-го об'єкта розмірності $p \times 1$.

$\overline{x}$ – вектор середніх значень по кожній змінній (розмірніст $p \times 1$).

S – вибіркова коваріаційна матриця $p \times p$ для всіх змінних.

Багатовимірний ексцес – це міра того, наскільки багатовимірний розподіл даних має важкі хвости або гостру вершину порівняно з нормальним розподілом розраховується за формулою:

$$b_{2,p} = \frac{1}{n}\sum_{i=1}^{n}[(x_i - \bar{x})^T S^{-1}(x_i - \bar{x})]^2 . \quad (2.5)$$

Для того щоб перевірити, чи відповідають дані нормальному закону розподілу, отримане значення $b_{2,p}$ порівнюють із теоретичним очікуваним значенням. Для багатовимірного нормального розподілу математичне сподівання ексцесу залежить від кількості змінних p і визначається виразом:

$$b_{2,p} \approx p(p+2). \quad (2.6)$$

Відповідно до тесту Мардія багатовимірний розподіл даних вибірки відхиляється від нормального закону, тому що значення тестової статистики розподілу вибірки для багатовимірної асиметрії склало 12,1042, що перевищує

значення розподілу x^2_{crit}, що ставновить 9,4877, де ступінь значущості α=0,05 та чотири ступені свободи.

Тестова статистика розподілу для багатовимірного ексцесу становить 57,1726 що перевищує квантиль розподілу Гауса, який становить 27,13 для математичного сподівання (2.6) 24 і диспресії 3,62 та рівня значущості 0,05.

На основі отриманих оцінок нульова гіпотеза про нормальність відхиляється. Можна констатувати, що навіть після виконання логарифмічної нормалізації, дані все ще не відповідають багатовимірному нормальному розподілу.

Одною з причин не нормального розподілу даних можуть бути викиди. Для вилучення аномальних значень було застосовано метод на основі відстані Махаланобіса, який дозволяє оцінити віддаленість точки від центру розподілу з урахуванням кореляцій між змінними. Спостереження класифікується як багатовимірний викид, якщо розраховане для нього значення D_2 перевищує критичне значення хі-квадрат розподілу x^2_{crit} для відповідного рівня значущості.

Критичне значення було обраховано і становить $x^2_{crit} = 9{,}4877$ де $\alpha = 0{,}05$ та кількість ступенів свободи становить 4.

Процес очищення даних відбувався ітеративно, оскільки вилучення екстремальних значень впливає на параметри розподілу всієї вибірки, що може виявити нові, раніше приховані викиди. Результати ітеративного пошуку викидів наведені в таблиці 2.3.

Таблиця 2.3 - Ітеративне вилучення викидів

Ітерація	*№*	*SMD*	*Різниця*
1	18	25,7476	16,2598
2	23	19,6472	10,1595
3	36	21,6397	12,1520
4	49	22,0227	12,5349
5	52	14,2951	4,8073
6	13	10,2274	0,7397
7	34	10,3352	0,8475

Вилучення викидів відбувалося ітеративно на нормалізованих даних. Задля вилучення всіх викидів знадобилося 8 ітерацій. Впродовж семи ітерацій на кожній з них було вилучено по одному значенню з найбільшим відхиленням, а восьму ітерацію проведено як контрольний підрахунок, під час якого, всі значення SMD виявилися меншими за критичне. Після видалення викидів з індексами $18 = 2{,}747; 23 = 19{,}6472; 36 = 21{,}639; 49 = 22{,}022; 52 = 14{,}295; 13 = 10{,}227;$ та $34 = 10{,}3352;$ залишилося 46 значень.

Дані нормалізовані, та очищені від викидів. Не буде зайвим провести тест Мардія на очищених даних, щоб впевнитись в їх нормальному розподілу. Оскільки, подальша розробка моделі, можлива виключно на нормалізованих даних. Проте треба переконатись, що вони дійсно стали відповідати нормальному розподілу. Було повторно застосовано тест Мардія на нормалізованих даних та з вилученими викидами.

Відповідно до тесту Мардія розподіл даних не відхиляється від багатовимірного нормального розподілу, тому що значення тестової статистики розподілу вибірки для багатовимірної асиметрії склало 2,35, що менше значення розподілу Хі-квадрат, що становить 9,4877 де ступінь значущості α=0,05 та кількість ступенів свободи 4.

Тестова статистика розподілу для багатовимірного ексцесу становить 20,52, що не перевищує квантиль розподілу Гауса, яке становить 27,13 для математичного сподівання 24 і диспресії 3,62 та рівня значущості 0,05.

Результати повторного тесту Мардія підтверджують, що після фільтрації аномальних значень дані задовольняють умову багатовимірної нормальності. Відсутність статистично значущих викривлень структури даних робить вибірку валідною для використання параметричних методів аналізу, що дає обґрунтовані підстави перейти до безпосередньої побудови лінійної регресійної моделі на основі підготовлених нормалізованих метрик.

2.3 Розробка нелінійної регресійної моделі для прогнозування розміру застосунків TypeScript

Після застосування нормалізуючого перетворення та ітераційного вилучення аномальних значень було сформовано нормалізовану вибірку даних, наведена в таблиці 2.4. Успішне проходження тесту Мардія гарантує відповідність даної вибірки закону багатовимірного нормального розподілу, що є необхідною передумовою для отримання незміщених та ефективних статистичних оцінок.

Таблиця 2.4 - Фрагмент нормалізованих та очищених даних від викидів

№	Z_1	Z_2	Z_3	Z_Y
1	2,807535	0,904357	3,676785	4,916707
2	2,292256	0,841921	2,920123	4,173070
3	1,531479	0,301030	1,531479	3,152900
4	2,158362	0,045757	2,758912	3,879555
5	1,698970	0,000000	2,187521	3,610447
6	2,383815	0,262588	2,933487	3,935104
7	2,759668	0,625223	3,698449	4,853881
8	2,155336	0,136920	2,612784	3,588047
9	1,568202	0,535602	2,152288	3,836197
10	2,252853	0,635888	2,802774	4,124569
11	3,158362	0,476216	3,629512	5,081999
12	2,984527	0,710078	3,749118	5,089870
14	2,894316	0,912474	3,725013	5,379438
15	1,633468	0,706976	2,471292	3,241048
16	2,173186	0,192302	2,313867	3,650890
17	1,041393	0,320335	1,819544	2,708421
19	2,021189	-0,004156	2,176091	3,787815
20	1,959041	0,693205	2,627366	3,997343
21	1,230449	0,413004	1,732394	2,748188
22	1,681241	0,458638	1,977724	3,801404
23	1,812913	0,187087	2,230449	3,743667

Для математичного опису залежності між нормалізованими метриками було сформовано рівняння множинної лінійної регресії. Математичний вираз трьохфакторної моделі у нормалізованому просторі ознак має вигляд:

$$Z_Y = \widehat{Z_y} + \varepsilon = b_0 + b_1 Z_1 + b_2 Z_2 + b_3 Z_3, \quad (2.7)$$

де

Z_Y – нормалізоване прогнозоване значення,

Z_1, Z_2, Z_3 – нормалізовані значення незалежних змінних (факторів),

ε – випадкова похибка.

Коефіцієнти $b_0 b_1 b_2 b_3$ для моделі (2.4) знайдені за методом найменших квадратів[16] : $b_0 = 1{,}1620;\ b_1 = 1{,}283;\ b_2 = 0{,}595;\ b_3 = -0{,}226.$

Після зворотного нормалізуючого перетворення, нелінійна регресійна модель має вигляд:

$$Y = 10^{1{,}620} X_1^{1{,}283} X_2^{0{,}595} X_3^{-0{,}226}. \quad (2.8)$$

Оцінки якості нелінійної моделі [17]:

– $R2$ = 0,893, відповідає критичному значенню ($R2$ > 0,75)

– $MMRE$ = 0,371 перевищує критичне значення ($MMRE \leq 0{,}25$);

– $PRED(0{,}25)$ = 0,428 нижче критичного значення ($PRED(0{,}25) > 0{,}75$).

Окрім нелінійної, було також побудовано лінійну регресійну модель. Це дозволить порівняти результати та переконатися в перевагах нелінійної моделі. Загальний вигляд лінійної моделі представлено у виразі:

$$\mathrm{Y} = \hat{Y} + \varepsilon = b_0 + b_1 X_1 + b_2 X_2 + b_3 X_3. \quad (2.9)$$

Коефіцієнти $b_0 b_1 b_2 b_3$ для моделі (2.9) були визначені методом найменших квадратів. Для розрахунків було використано нормалізовану вибірку даних, очищену від аномальних спостережень. Отримані коефіцієнти мають такі значення:: $b_0 = 27125{,}891,\ b_2 = 104{,}074,\ b_2 = 1812{,}587,\ b_3 = -10{,}588.$

Оцінки якості лінійної моделі [17]:

– $R2$ = 0,650, нижче критичного значення ($R2$ > 0,75)

– $MMRE$ = 1,853 значно перевищує критичне значення ($MMRE \leq 0{,}25$);

– $PRED(0{,}25)$ = 0,260 нижче критичного значення ($PRED(0{,}25)$ > 0,75).

Після проведення оцінки якості лінійної регресійної моделі та отримання її коефіцієнтів, наступним важливим кроком є кількісне визначення надійності та точності цих оцінок. Однак, оперування виключно точковими прогнозами є недостатнім, оскільки вони не відображають міру невизначеності, притаманну процесу розробки програмного забезпечення. Тому, з метою врахування стохастичної природи метрик та можливих похибок вимірювання, було застосовано метод інтервального оцінювання. Зокрема, побудовано довірчі інтервали Confidence Intervals. Довірчий інтервал нелінійної регресії розраховується за формулою:

$$\psi_Y^{-1}!\left(\widehat{z_Y} \pm t_{\alpha/2,\nu}\, S_{z_Y} \left\{\frac{1}{N} + (z_X^+)^T [S_{ZZ}]^{-1} (z_X^+)\right\}^{1/2}\right). \qquad (2.10)$$

В наведеному виразі враховується варіація як самої цільової змінної, так і факторів моделі, що дає змогу оцінити розкид можливих значень показника навколо його точкової оцінки. Це дозволяє встановити діапазон, в якому з заданою надійністю знаходяться істинні значення параметрів.

Проте, для практичного планування часто необхідно знати не лише, де знаходиться середнє значення, а й в які межі потрапить конкретне майбутнє значення метрики. Окрім довірчих інтервалів, було побудовано інтервали передбачення Prediction Intervals. Інтервал передбачення нелінійної регресії розраховується наступним чином:

$$\psi_Y^{-1}\left(\widehat{z_Y} \pm t_{\alpha/2,\nu}\, S_{z_Y} \left\{1 + \frac{1}{N} + (z_X^+)^T [S_{ZZ}]^{-1} (z_X^+)\right\}^{1/2}\right). \qquad (2.11)$$

Результати розрахованих довірчих інтервалів для лінійної та нелінійної моделі наведено в таблиці 2.5.

Таблиця 2.5 – Розраховані довірчі інтервали

№	SLOC (Y)	Межі інтервалів передбачення				Ширина інтервалу	
		Лінійна регресія		Нелінійна регресія			
		LB	UB	LB	UB	Лінійна регресія	Нелінійна регресія
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	14896,00	19071,60	49597,97	19860,44	32145,19	30526,38	12284,75
2	1422,00	-14590,19	8586,32	1877,26	3658,36	23176,51	1781,10
3	7578,00	-8842,96	15713,61	4839,31	7973,22	24556,57	3133,91
4	4078,00	-19312,81	7547,33	1524,76	2680,08	26860,14	1155,32
5	8612,00	5526,91	25714,74	12552,94	17601,40	20187,82	5048,46
6	71430,00	37747,73	84692,83	36981,73	67080,85	46945,11	30099,12
7	3873,00	-7267,81	15813,66	6199,97	9140,12	23081,47	2940,15
8	6858,00	-6594,25	14169,25	2342,53	3627,94	20763,50	1285,41
9	13322,00	11235,17	30538,06	15182,14	21400,85	19302,89	6218,71
10	1742,00	278,55	23560,80	2746,60	5220,69	23282,25	2474,09
11	4476,00	-5964,56	16649,31	7553,11	13252,31	22613,86	5699,20
12	511,00	-16528,55	7393,69	354,56	834,97	23922,24	480,41
13	6135,00	-13759,99	11535,21	3884,78	7068,59	25295,20	3183,81
14	9939,00	4669,22	26524,14	7279,67	11043,27	21854,93	3763,60
15	560,00	-13006,04	9315,68	861,99	1482,34	22321,72	620,35
16	6330,00	-8213,08	12621,81	3282,65	4905,37	20834,89	1622,72
17	5542,00	-14178,21	9878,78	2937,96	4356,29	24057,00	1418,33
18	3924,00	-16721,06	9125,91	2436,33	3965,72	25846,97	1529,39
19	40455,00	47455,48	72678,50	38178,95	59628,41	25223,02	21449,46
20	39035,00	43034,61	73207,39	37162,63	58688,95	30172,78	21526,31
21	5578,00	5032,64	34412,46	6388,85	10320,60	29379,82	3931,75
22	1201,00	-21512,07	5606,80	559,34	1130,55	27118,87	571,21
23	14025,00	-1580,53	21427,02	8636,87	13326,21	23007,55	4689,34
24	28617,00	37328,40	66611,97	26415,66	44176,51	29283,57	17760,85
25	10244,00	-2754,62	19465,79	7789,91	11500,37	22220,41	3710,47
26	52607,00	24000,23	48043,42	29017,30	44228,28	24043,19	15210,98

Результати розрахованих границь інтервалів передбачення та їх довжин для лінійної і нелінійної моделей наведено в таблиці 2.6.

Таблиця 2.6 – Розраховані границі інтервалів передбачення

№	SLOC (Y)	Межі інтервалів передбачення				Ширина інтервалу	
		Лінійна регресія		Нелінійна регресія			
		LB	UB	LB	UB	Лінійна регресія	Нелінійна регресія
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	14896,00	-24527,98	93197,55	9583,68	66615,09	117725,54	57031,41
2	1422,00	-61020,46	55016,58	967,40	7099,11	116037,04	6131,71
3	7578,00	-54724,94	61595,60	2350,78	16413,62	116320,54	14062,84
4	4078,00	-64297,02	52531,53	758,32	5388,85	116828,55	4630,53
5	8612,00	-42117,79	73359,44	5724,86	38594,72	115477,23	32869,87
6	71430,00	-284,37	122724,93	18597,56	133392,01	123009,29	114794,45
7	3873,00	-53736,12	62281,97	2885,52	19638,91	116018,09	16753,39
8	6858,00	-54002,13	61577,13	1111,55	7645,69	115579,26	6534,15
9	13322,00	-36776,30	78549,52	6938,98	46824,00	115325,82	39885,03
10	1742,00	-46109,43	69948,78	1403,61	10215,95	116058,20	8812,34
11	4476,00	-52620,61	63305,36	3754,04	26663,57	115925,97	22909,53
12	511,00	-62661,57	53526,71	193,84	1527,28	116188,28	1333,44
13	6135,00	-59351,75	57126,96	1955,72	14040,81	116478,71	12085,08
14	9939,00	-42293,48	73486,84	3426,55	23461,34	115780,31	20034,80
15	560,00	-59779,85	56089,49	425,35	3004,04	115869,33	2578,69
16	6330,00	-55591,69	60000,42	1536,02	10483,35	115592,11	8947,33
17	5542,00	-60257,77	55958,33	1370,50	9338,64	116216,10	7968,14
18	3924,00	-62097,47	54502,31	1178,15	8200,79	116599,78	7022,64
19	40455,00	1835,46	118298,52	18175,05	125256,86	116463,06	107081,80
20	39035,00	-696,17	116938,17	17766,64	122760,19	117634,35	104993,55
21	5578,00	-38994,18	78439,28	3080,69	21403,23	117433,46	18322,54
22	1201,00	-66396,79	50491,52	291,72	2167,69	116888,31	1875,97
23	14025,00	-48078,46	67924,94	4092,33	28125,00	116003,41	24032,68
24	28617,00	-6734,52	110674,89	12902,88	90441,18	117409,41	77538,31
25	10244,00	-49569,34	66280,52	3627,55	24696,27	115849,86	21068,72
26	52607,00	-22084,79	94128,45	13683,79	93788,72	116213,24	80104,93

Детальний аналіз результатів моделювання виявив низку критичних недоліків лінійного підходу, які ставлять під сумнів доцільність його використання для прогнозування розміру програмного забезпечення. Найбільш суттєво це проявилося у фізично неможливих прогнозах лінійної моделі: межі довірчих інтервалів значень розміру застосунку набували від'ємних значень, що суперечить природі досліджуваної величини $SLOC > 0$. Натомість нелінійна модель завдяки своїй математичній структурі забезпечує невід'ємність прогнозів.

Вагомою перевагою нелінійного підходу є також надійність інтервальних оцінок. У лінійній моделі було зафіксовано, що 2,8% прогнозованих значень вийшли за межі довірчих інтервалів. У нелінійній моделі всі емпіричні значення (100%) залишилися в межах розрахункових границь. Також ширина довірчих інтервалів нелінійної моделі виявилася в середньому на 51,1% вужчою, ніж у лінійної, що свідчить про значно меншу невизначеність прогнозу.

Кількісне зіставлення метрик якості наведено в таблиці 2.7.

Таблиця 2.7 - Оцінки якості моделі

Оцінки якості моделі	Лінійна	Нелінійна
R^2 (Коефіцієнт детермінації)	0,650	0,8995
$MMRE$(Середня відносна похибка)	1,853	0,3720
$PRED(0{,}25)$ (Точність прогнозу)	0,260	0,4286

Аналіз даних таблиці 2.7 демонструє повну непридатність лінійної моделі. Її коефіцієнт детермінації (R^2=0,650) не досягає мінімально необхідного порогу 0,75, тобто модель пояснює лише 65% варіації даних. Середня відносна похибка $MMRE$ сягає катастрофічного рівня 1,853 (185%), що робить прогноз фактично випадковим.

Нелінійна модель демонструє значно кращі показники:

– R^2 = 0,899 перевищує критичне значення, що підтверджує наявність

сильного зв’язку між предикторами та залежною змінною саме за нелінійним законом.

– $MMRE$=0,371 скоротилася більш ніж в 4 рази порівняно з лінійною моделлю. Хоча вона все ще перевищує ідеальний критерій (0,25), це є суттєвим прогресом.

– PRED=0,428, що також значно краще за лінійну модель.

2.4 Порівняння розробленої моделі з попередніми рішеннями

Важливим етапом верифікації отриманих результатів є зіставлення розробленої моделі, з іншими моделями які також були розроблені з урахуванням специфіки мови програмування TypeScript. Для порівняння, обрано нелінійну регресійну модель, представлену в кваліфікаційній роботі Чебакова Д. В. (2020). «Удосконалення регресійної моделі для оцінювання розміру програмних застосунків на TypeScript та розробка програми для її реалізації»[5]. Оскільки обидва дослідження базуються на ідентичному технологічному стеку та предметній області, подібне порівняння є найбільш репрезентативним.

Результати порівняльного аналізу метрик якості розробленої удосконаленої моделі та моделі Чебакова Д.В. наведено в таблиці 2.8.

Таблиця 2.8 – Порівняння показників якості моделей

Оцінки якості моделі	Модель Чебакова Д. В.	Розроблена модель
R^2 (Коефіцієнт детермінації)	0,6824	0,8995
$MMRE$ (Середня відносна похибка)	0,3982	0,3720
$PRED(0{,}25)$ (Точність прогнозу)	0,5	0,4286

Аналіз даних показує, що розроблена модель забезпечує значно вищий коефіцієнт детермінації 0,8995 проти 0,6824, що свідчить про кращу здатність описувати загальні тенденції в даних. Водночас спостерігається певне зниження показника $PRED(0{,}25)$, і пояснюється тим, що вибірка даних, зібрана в межах поточної роботи, характеризується значно вищою дисперсією (розкидом значень) та різнорідністю проєктів, ніж у попередньому дослідженні.

Оскільки модель, з якою порівнюємо поточну, є лише однофакторною та враховує лише X_1(NOC). Можливо перевірити, які результати покаже однофакторна нелінійна модель з коефіцієнтами $b_0 = 2{,}8887;\ b_1 = 0{,}6278$ [5] на даних з зібраних в рамках поточної роботи.

Порівняння метрик якості, моделей отриманих на даних з поточної роботи наведені в таблиці 2.9.

Таблиця 2.9 – Порівняння показників моделей отриманих на поточних даних

Оцінки якості моделі	Модель Чебакова Д. В.	Розроблена модель
R^2	0,4646	0,8995
$MMRE$	1,3591	0,3720
$PRED(0{,}25)$	0,1957	0,4286

На основі поточних даних розроблена модель продемонструвала значно кращі результати порівняно з моделлю попередника.

Отримані результати підтверджують суттєве покращення R^2 становить 0,4646 проти 0,8995, тобто покращення в 1,9 рази. $MMRE$ 1,3591 проти 0,3720, хоча все ще за критичним значенням це покращення в 3,6 разів. $PRED(0{,}25)$ 0,1957 проти 0,4286 також значне зростання точності в 2,1 рази порівнянно з минулою моделлю.

На підставі проведеного експерименту можна зробити висновок, що розроблена нелінійна модель має очевидні переваги, продемонструвавши

покращення за всіма ключовими метриками якості R^2, MMRE, PRED, в порівнянні з базовою моделлю, що дозволяє вважати її успішним удосконаленням попередніх підходів до прогнозування розміру TypeScript-застосунків, зокрема завдяки врахуванню додаткової метрики X_3 (TR).

2.5 Висновок

В представленому розділі виконано підготовку даних, що включала нормалізацію та очищення викидів за критерієм Махаланобіса. Статистичні перевірки зокрема тест Мардія підтвердили нормальність розподілу очищеної вибірки та відсутність мультиколінеарності, що обґрунтувало коректність регресійного аналізу.

Порівняння методів продемонструвало перевагу нелінійної моделі над лінійною, яка генерувала фізично некоректні від'ємні прогнози. Нелінійний підхід забезпечив адекватність оцінок та звуження довірчих інтервалів на 51,1%.

Ключовим досягненням стало доведення ефективності введення нової метрики, кількості зв'язків між класами (X_3), що дозволило компенсувати високу дисперсію даних, підвищити коефіцієнт детермінації до $R^2 \approx 0{,}9$ та зменшити середню похибку приблизно в чотири рази порівняно з однофакторною нелінійною моделлю. Розроблена модель є успішним удосконаленням методів прогнозування для TypeScript-застосунків та готова до програмної реалізації.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНКИ РОЗМІРУ ЗАСТОСУНКІВ МОВОЮ TypeScript

3.1 Постановка задачі визначення функціональних та не функціональних вимог

Після виконання аналізу існуючих методів оцінювання та побудови удосконаленої нелінійної регресійної моделі для прогнозування розміру TypeScript-застосунків, логічним кроком є перехід до практичної реалізації отриманих результатів. Основною метою даного етапу, є розробка програмного забезпечення, яке дозволить автоматизувати процес розрахунків відповідно до створеної математичної моделі. Система має оперувати трьома вхідними факторами – кількістю класів NOC, середньою кількістю атрибутів AAC та кількістю зв’язків між класами TR, – на основі яких обчислюватиметься результуюча змінна, прогнозований обсяг вихідного коду SLOC.

Перехід від теоретичного математичного моделювання до етапу безпосередньої програмної реалізації вимагає чіткої формалізації завдань, які має вирішувати майбутня система. Якість, надійність та відповідність програмного продукту очікуванням користувача безпосередньо залежать від повноти та точності вимог, сформульованих на початкових стадіях життєвого циклу розробки. Тому, для забезпечення коректної імплементації алгоритмів та гарантування відповідності розроблюваного рішення поставленим задачам, необхідно визначити та деталізувати функціональні та нефункціональні вимоги системи.

В межах програмної інженерії вимоги традиційно поділяють на дві основні категорії [17].

Функціональні вимоги – які визначають набір операцій та функцій, що їх повинна виконувати система для вирішення бізнес-задач користувача.

Нефункціональні вимоги – які описують атрибути якості системи, такі як продуктивність, надійність, зручність інтерфейсу та вимоги до оточення, в якому функціонуватиме програмний засіб.

Виходячи з мети роботи та особливостей розробленої математичної моделі, до програмного продукту висуваються наступні функціональні вимоги:

– введення вхідних даних – система повинна надавати інтерфейс для ручного введення значень трьох незалежних змінних, а саме, кількість класів NOC, середня кількість атрибутів AAC, загальна кількість зв'язків TR;

– валідація даних – система повинна виконувати перевірку коректності введених значень перед початком розрахунків.

Механізм валідації має включати:

– перевірку типів – впевнитися, що введені дані є числовими;

– перевірку діапазону – заборонити введення від'ємних чисел для всіх метрик;

– логічну перевірку – кількість класів NOC не може бути меншою за 1, оскільки проєкт не може існувати без сутностей. В разі виявлення некоректних даних, система повинна блокувати виконання розрахунку та виводити відповідне повідомлення користувачеві;

– виконання розрахунків – система повинна реалізувати обчислення прогнозованого розміру $SLOC$ шляхом підстановки введених значень в розроблене рівняння нелінійної регресії;

– відображення результатів – виведення розрахованого значення розміру застосунку у зручному для сприйняття форматі.

До нефункціональних вимог висуваються наступні критерії:

– щвидкодія – час виконання розрахунків та перемальовування графіків не повинен перевищувати прийнятних для користувача меж (1-2 секунди) навіть при обробці великих вибірок;

– сумісність – програмне забезпечення повинно функціонувати на сучасних операційних системах Windows, macOS, Linux без необхідності встановлення складного додаткового середовища;

– інформаційна безпека – оскільки розроблена модель прогнозування та результати її роботи становлять комерційну цінність та надають компанії конкурентні переваги, програмне забезпечення повинно гарантувати захист даних. Деталізований перелік функціональних та експлуатаційних вимог до розроблюваної системи, а також етапи її реалізації наведені у технічному завданні (Додаток А).

3.2 Обґрунтування вибору платформи реалізації та архітектури програмного засобу

На основі сформованих вимог, було проведено аналіз існуючих підходів до реалізації програмних продуктів – нативного, кросплатформного та веб-орієнтованого.

Загальноприйнятим стандартом в індустрії, вважається нативна розробка. До її беззаперечних переваг належить максимальна продуктивність та ефективність використання апаратних ресурсів, що є критичним для задач зі складними математичними обчисленнями або складною візуалізацією. Однак, суттєвим недоліком даного підходу є висока трудомісткість та вартість, тому що підтримка декількох операційних систем вимагає створення та супроводу окремих кодових баз для кожної платформи.

Оскільки, однією з ключових нефункціональних вимог є забезпечення сумісності з усіма сучасними операційними системами Windows, macOS, Linux, варіант нативної розробки було відхилено. Реалізація окремих версій застосунку призвела б до невиправданого збільшення часу розробки та ускладнення підтримки продукту.

Альтернативою, що дозволяє вирішити проблему мультиплатформенності, є використання кросплатформних фреймворків, подібно, Tauri, Xamarin або Electron. Вони дозволяють використовувати єдину кодову базу для створення застосунків під різні ОС (рис. 3.1).

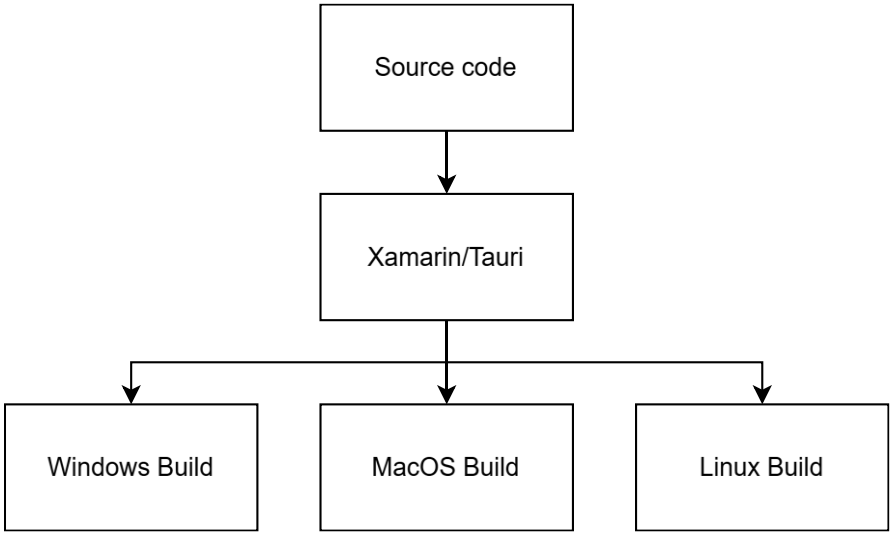


Рисунок 3.1 – Розробка застосунків кросплатформенними засобами

Використання засобів кросплатформної розробки дозволяє суттєво оптимізувати трудовитрати та задовольнити більшість функціональних вимог, зокрема, щодо сумісності з різними операційними системами. Однак, суттєвим недоліком представленого підходу, є складність забезпечення нефункціональної вимоги «Інформаційна безпека».

Оскільки десктопний застосунок (навіть кросплатформний) виконується на стороні клієнта, існує високий ризик несанкціонованого копіювання (піратства) або реверс-інжинірингу розробленої математичної моделі. Для протидії, довелося б застосовувати складні механізми захисту, такі як обфускація коду, апаратні ключі або суворі ліцензійні обмеження. Впровадження подібних заходів, не лише ускладнює розробку, але й погіршує ергономіку використання продукту для кінцевого користувача.

Ефективним вирішенням даної проблеми є архітектурний підхід, при якому математична модель фізично не передається на пристрій користувача, а розрахунки відбуваються в ізольованому, контрольованому середовищі. Саме таку можливість надає реалізація системи у вигляді веб-застосунку, архітектура якого передбачає чіткий поділ на клієнтську Front-end та серверну Back-end частини [19] (рис. 3.2).

Використання засобів крос-платформеної розробки значно зменшує трудовитрати та дає змогу закрити майже всі функціональні та не функціональні вимоги. Окремі від не функціонального критерію «Інформаційна безпека», оскільки отримавши застосунок, будь-хто може неправомірно привласнити собі його, та для перешкоджання цьому доведеться застосовувати складні методи такі, як впровадження ліцензійних ключів, або встановлення часових обмежень на використання програми. Дані методи хоч і забезпечать кращий захист розробленої моделі, проте зроблять її використання менш зручним.

Гарним рішенням взагалі не відправляти на клієнт-модель, і щоб розрахунки відбувались десь в іншому ізольованому місці до якого доступ мали лише розробники застосунку. Подібний підхід дозволяє реалізувати веб-застосунок (рис. 3.2) оскільки розділяє проєкт на дві частини клієнтську та серверну.

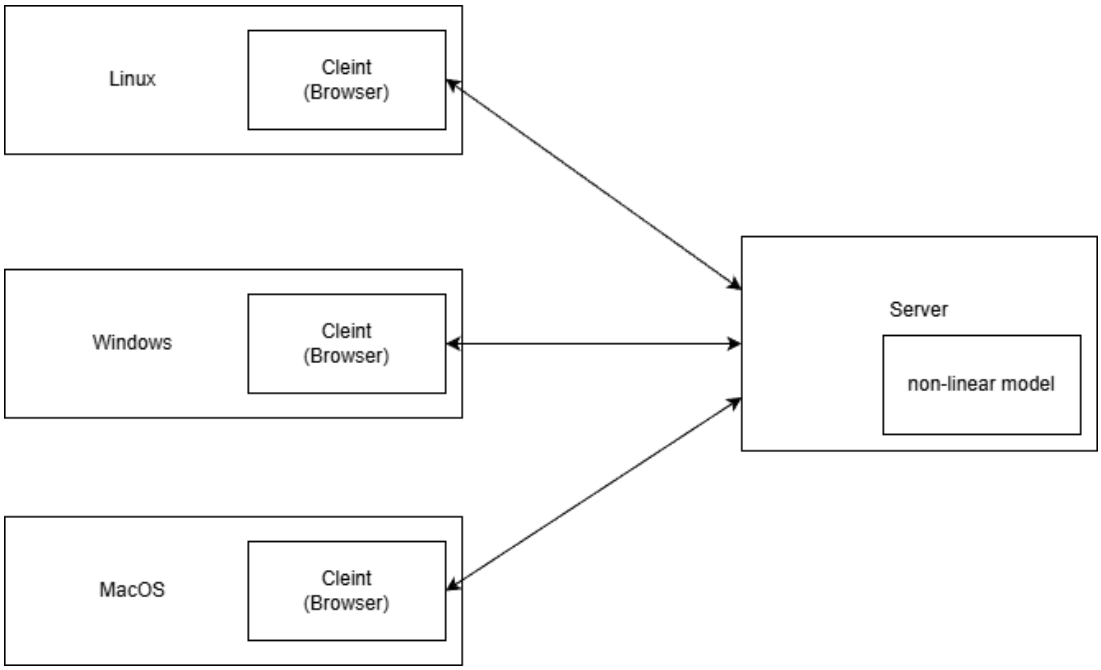


Рисунок 3.2 – Базова архітектура веб застосунку

В запропонованій архітектурі, на стороні клієнта виконується виключно логіка презентації та візуалізації даних. Натомість, застосування нелінійної регресійної моделі та обробка метрик відбуваються на сервері. Для клієнта, сервер

виступає у ролі «чорної скриньки», тобто він отримує вхідні параметри та повертає результат, що унеможливлює копіювання алгоритмів та коефіцієнтів моделі.

Додатковою перевагою веб-архітектури, є можливість налаштування мережевого доступу. Розроблений застосунок може бути розгорнутий виключно в межах захищеної корпоративної мережі. В даному випадку, навіть за умови витоку клієнтського коду, зловмисник не зможе скористатися функціоналом системи, оскільки API сервера буде недоступним ззовні (рис. 3.3).

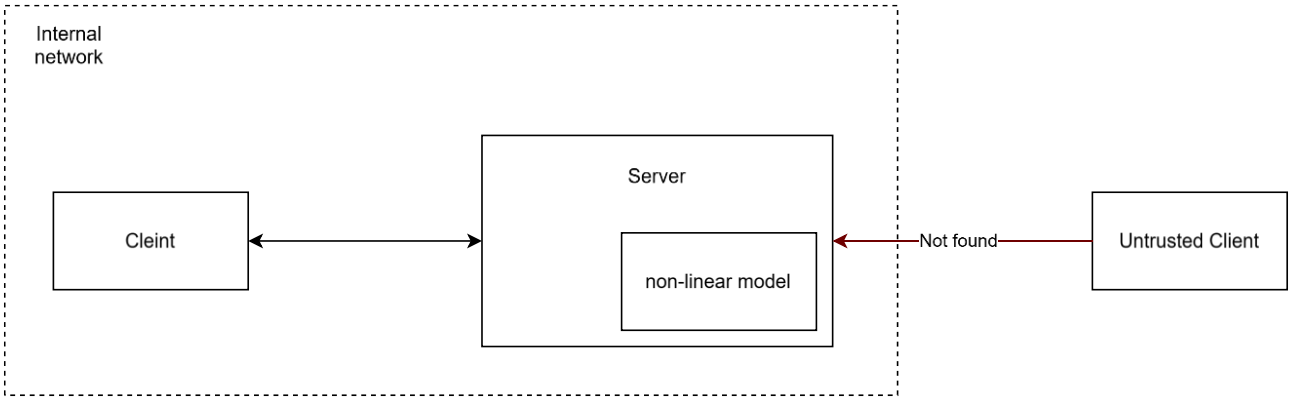


Рисунок 3.3 – Доступність сервера виключно з внутрішньої мережі

3.3 Розробка діаграми варіантів використання

Наступним етапом, після визначення переліку функціональних вимог є їх графічна формалізація та моделювання поведінки системи. Для цього, в сучасній інженерії програмного забезпечення загальноприйнятим стандартом є використання уніфікованої мови моделювання UML. Першим кроком в процесі об’єктно-орієнтованого проєктування, традиційно виступає побудова діаграми варіантів використання Use Case Diagram[20].

Діаграма варіантів використання – це діаграма поведінки, яка описує сукупність сценаріїв взаємодії між зовнішніми сутностями акторами та системою, що розробляється. Згідно з класичним визначенням [20], варіант використання use case специфікує набір послідовностей дій, які система виконує для отримання значущого результату для конкретного актора. Головною метою побудови цієї

діаграми в рамках даної роботи є візуалізація меж системи, чітке розмежування того, що відбувається всередині веб-застосунку, ідентифікація акторів, визначення ролей користувачів. Структурування функціоналу – відображення залежностей між функціями.

Розроблена діаграма варіантів використання, що відображає функціональну структуру системи та взаємодію з зовнішніми сутностями, наведена на рисунку 3.4. В процесі проєктування було ідентифіковано два типи акторів, які мають різні рівні доступу та цілі використання системи: «Користувач» та «Адміністратор».

Роль «Користувача» призначена для менеджерів проєктів або тімлідів, головною метою яких, є отримання швидкої та точної оцінки обсягу майбутнього програмного продукту. Для даного актора, реалізовано варіант використання «Прогнозування розміру проєкту». Даний сценарій передбачає введення архітектурних метрик та отримання результату у вигляді прогнозованого значення SLOC разом з довірчими інтервалами. Користувач не має доступу до зміни внутрішніх налаштувань моделі, що забезпечує цілісність даних.

Актор «Адміністратор» – роль адміністратора або аналітика даних є ключовою для підтримання актуальності математичного апарату системи. Його завдання полягає в періодичному оновленні навчальної вибірки новими даними та перерахунку коефіцієнтів регресії для підвищення точності прогнозів. Для даного актора, передбачено розширений набір функцій. «Завантаження даних» дозволяє імпортувати нові датасети з метриками реальних проєктів для розширення навчальної бази. «Розрахунок нелінійної моделі» та «Лінійної моделі», центральний процес адміністративної частини, який ініціює перенавчання системи. Важливою особливістю спроєктованої системи є наявність залежностей типу <<include>>, які демонструють, що виконання складних операцій неможливе без попередніх етапів обробки даних.

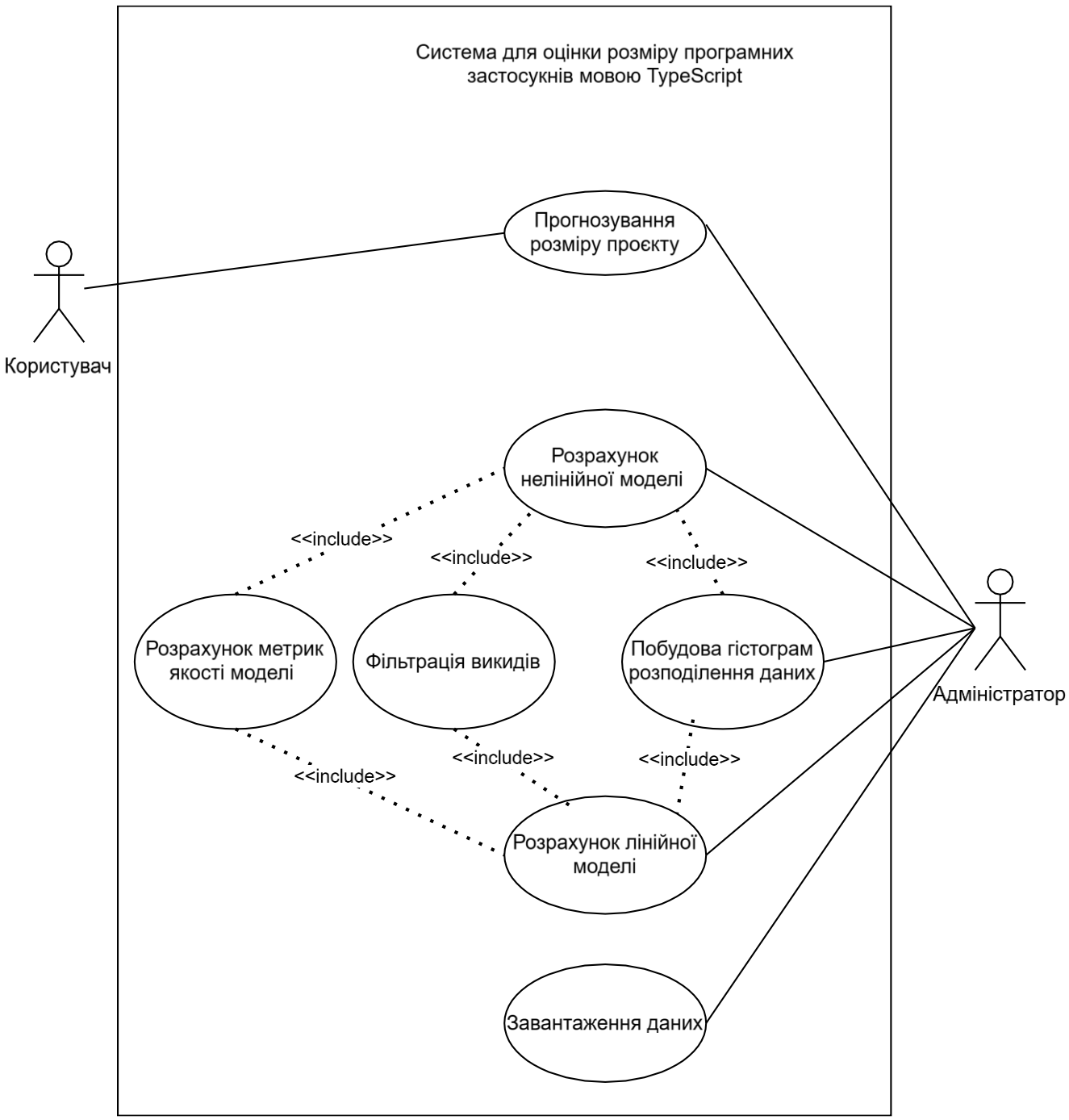


Рисунок 3.4 – Діаграма варіантів використання веб-застосунку

Після побудови діаграми використання, слід навести специфікації, кожної функції в таблицях 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8.

Таблиця 3.1 – Специфікація варіанту використання «Завантаження даних»

Призначення	Варіант використання призначений для завантаження нових даних в систему
Учасник	Адміністратор, система
Передумова	Дані про валідовані, в правильному форматі. Та фізично конкретних значень. Всі значення більше 0.
Короткий опис	Дозволяє завантажити дані до системи
Процедура	- адміністратор завантажує .csv файл - система робить валідацію даних - система інформує користувача
Альтернативна процедура	Не виділена.
Аварійні умови	Якщо файл з даними не вдалось зчитати. Система виводить помилку. Якщо дані в файлі, не є валідними, система виводить помилку відповідну помилку.
Пост умова	Після успішного завантаження даних, адміністратор бачить нові дані в таблиці.

Таблиця 3.2 – Специфікація варіанту використання «Прогнозування розміру проєкту»

Призначення	Варіант використання призначений для оцінки розміру програмного застосунку мовою TypeScript.
Учасник	Користувач/Адміністратор, система
Передумова	Введено валідні метрики NOC, AAC, TR.
Короткий опис	Дозволяє оцінити розмір застосунку мовою TypeScript.
Процедура	- користувач вводить метрики NOC, AAC, TR - натискає кнопку «Оцінити розмір застосунку» - система виводить розрахований розмір в $SLOC$
Альтернативна процедура	Не виділена.
Аварійні умови	Якщо введені метрики не валідні, система інформує про помилку користувача і блокує розрахунок.
Пост умова	Після успішного розрахунку, система виводить розмір застосунку в SLOC

Таблиця 3.3 – Специфікація варіанту використання «Розрахунок нелінійної регресійної моделі»

Призначення	Варіант використання призначений для розрахунку нелінійної регресійної моделі.
Учасник	Адміністратор, система
Передумова	Дані вже завантажені в системі.
Короткий опис	Даний варіант дозволяє розрахувати коефіцієнти нелінійної регресійної моделі.
Процедура	- адміністратор натискає кнопку «Розрахувати нелінійну регресійну модель» - система нормалізує завантажені данні - система фільтрує викиди - розраховує коефіцієнти нелінійної регресійної моделі - система робить «Розрахунок метрик якості моделі»
Альтернативна процедура	Не виділена.
Аварійні умови	Якщо, дані відсутні, система виводить помилку.
Пост умова	Після успішного розрахунку адміністратор отримує, гістограми розподілення даних, список викидів які було вилучено, коефіцієнти моделі та метрики якості моделі.

Таблиця 3.4 – Специфікація варіанту використання «Розрахунок лінійної регресійної моделі»

Призначення	Варіант використання призначений для розрахунку лінійної регресійної моделі.
Учасник	Адміністратор, система
Передумова	Дані вже завантажені в системі.
Короткий опис	Даний варіант дозволяє розрахувати коефіцієнти нелінійної регресійної моделі.
Процедура	- адміністратор натискає кнопку «Розрахувати лінійну регресійну модель» - розраховує коефіцієнти лінійної регресійної моделі - система виконує «Розрахунок метрик якості моделі»
Альтернативна процедура	Не виділена.
Аварійні умови	Якщо, дані відсутні, система виводить помилку.
Пост умова	Після успішного розрахунку адміністратор отримує, гістограми розподілення даних,, коефіцієнти моделі та метрики якості моделі.

Таблиця 3.5 – Специфікація варіанту використання «Розрахунок метрик якості моделі»

Призначення	Варіант використання призначений для розрахунку метрик якості регресійної моделі
Учасник	Адміністратор, система
Передумова	Порахована регресійна модель.
Короткий опис	Даний варіант, дає змогу адміністратору оцінити якість отриманої моделі
Процедура	- після розрахунку регресійної моделі - система виводить метрики якості регресійної моделі R^2, $MMRE$, $PRED(0.25)$
Альтернативна процедура	Не виділена.
Аварійні умови	Якщо, в процесі підрахунку метрик виникла помилка, система виводить її користувачу, за для інформування.
Пост умова	Виведення метрик регресійної моделі.

Таблиця 3.6 – Специфікація варіанту використання «Побудова гістограм розподілення даних»

Призначення	Варіант використання призначений для побудови гістограм розподілення даних
Учасник	Адміністратор, система
Передумова	Порахована регресійна модель. Завантажені дані в систему.
Короткий опис	Даний варіант, дає змогу адміністратору оцінити розподілення даних в завантаженому дата сеті. Гістограми будується для проміжних даних. Якщо, будується нелінійна регресійна модель будуються гістограми для даних до нормалізації, та після.
Процедура	- під час розрахунку регресійної моделі - будуються гістограми розподілення даних
Альтернативна процедура	Не виділена.
Аварійні умови	Якщо, в процесі підрахунку метрик виникла помилка, система виводить її користувачу, за для інформування.
Пост умова	Виведення гістограми розподілення даних

Таблиця 3.7 – Специфікація варіанту використання «Фільтрація викидів»

Призначення	Варіант використання призначений для фільтрації викидів
Учасник	Адміністратор, система
Передумова	Завантажені дані в систему.
Короткий опис	Даний варіант, дає змогу очистити дані від аномальних значень.
Процедура	- процес підготовки даних, для побудови моделі - система розраховує викиди - система вилучає викиди з вибірки даних
Альтернативна процедура	Не виділена.
Аварійні умови	Якщо, в процесі підрахунку метрик виникла помилка, система виводить її користувачу, за для інформування.
Пост умова	Система виводить список вилучених значень з вибірки

Побудована діаграма варіантів використання дозволила чітко розмежувати зони відповідальності між акторами «Користувач» та «Адміністратор», а також, визначити перелік критично важливих процесів, таких як валідація даних, фільтрація викидів та розрахунок прогнозу. Розроблені специфікації прецедентів однозначно описують очікувану поведінку системи, що мінімізує ризики виникнення логічних помилок на етапі імплементації програмного коду.

3.4 Побудова діаграм діяльності для ключових сценаріїв використання

Після визначення функціональних вимог наступним критично важливим етапом є моделювання динамічних аспектів її поведінки. Статичні моделі, хоч і відображають архітектуру програмних компонентів, не дають повного уявлення про те, як саме система обробляє дані у часі, як реагує на дії користувача та які алгоритмічні перетворення відбуваються.

Оскільки розроблюваний веб-застосунок базується на складному математичному апараті, який включає ітераційні процедури фільтрації даних, перевірку статистичних гіпотез та розрахунок регресійних коефіцієнтів, виникає

необхідність у детальній візуалізації цих процесів перед початком написання коду. Помилка в логіці алгоритму на цьому етапі може призвести до некоректних прогнозів, що неприпустимо для системи оцінювання. Для вирішення цього завдання в мові UML використовуються діаграми діяльності Activity Diagrams[20]. Вони дозволяють відобразити послідовність виконання операцій, розгалуження логіки, цикли та потоки даних між різними частинами системи. Нижче наведено діаграми діяльності для основних головних варіантів використання.

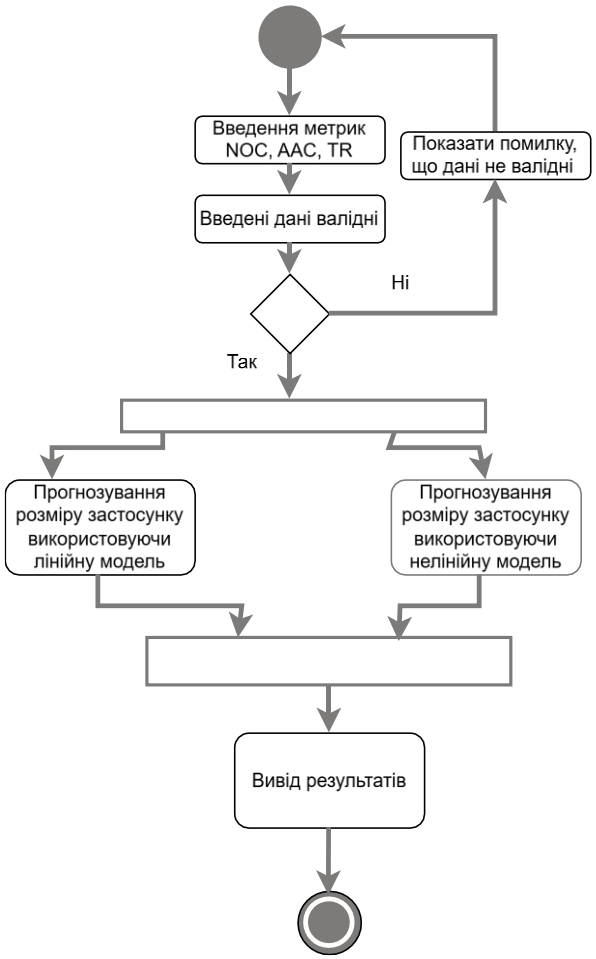


Рисунок 3.5 – Діаграма діяльнсоті прецеденту «Прогнозування розміру проєкту»

На рисунку 3.5 представлено детальну діаграму діяльності для прецеденту «Прогнозування розміру проєкту». Схема демонструє, що перед початком розрахунків вхідні дані проходять обов’язкову валідацію. У разі введення некоректних значень система виводить повідомлення про помилку та блокує подальші дії. Після успішного виконання обчислень користувачеві надаються

результати оцінювання, отримані за допомогою як лінійної, так і нелінійної моделей.

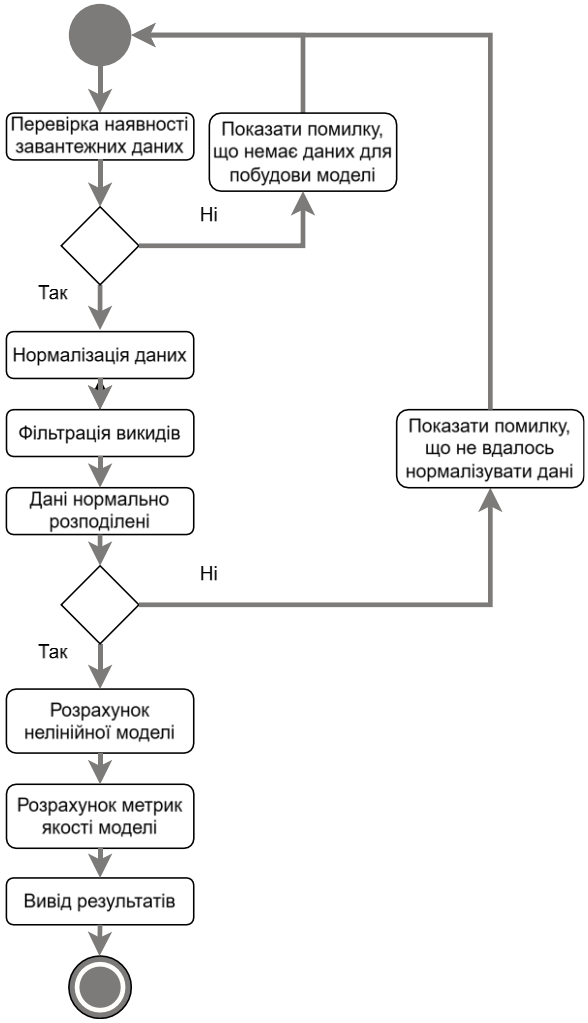


Рисунок 3.6 – Діаграма діяльності прецеденту «Розрахунок нелінійної моделі»

На рисунку 3.6 представлено детальну діаграму діяльності для прецеденту «Розрахунок нелінійної моделі». Схема демонструє, що перед початком розрахунків відбувається перевірка наявності даних для навчання моделі в системі. Якщо дані відсутні, виводиться повідомлення про помилку, і подальші розрахунки не виконуються. Також у процесі підготовки даних здійснюється перевірка розподілу: якщо не вдалося нормалізувати дані перед початком розрахунку, система генерує помилку. Після успішної побудови моделі проводиться

розрахунок метрик її якості, які користувач отримує у вигляді результату разом зі знайденими коефіцієнтами.

3.5 Розробка моделі переходів станів ПЗ оцінювання розміру застосунків мовою TypeScript

Після побудови діаграм діяльності для ключових сценаріїв використання, які відображають послідовність виконання дій у системі, доцільним є перехід до детальнішого аналізу її поведінки в часі.

Для цього використовується модель переходів станів [20], яка зосереджується не на діях, а на станах програмного забезпечення та умовах їх зміни. Такий підхід дозволяє доповнити результати попереднього розділу та отримати цілісне уявлення про функціонування системи. Діаграми станів дають змогу показати, у якому саме стані перебуває програмне забезпечення для оцінювання розміру застосунків у певний момент часу, а також які події або умови ініціюють перехід до іншого стану.

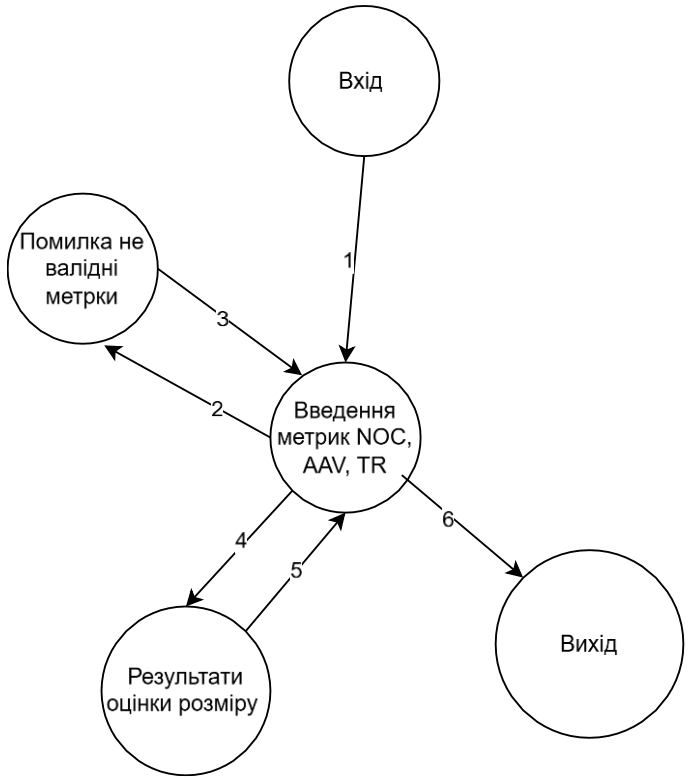


Рисунок 3.7 – Діаграма переходів станів програми для користувача

На рисунку 3.7 зображено діаграма переходів станів застосунку коли їм користується користувач. Можливі події на переходах описані в таблиці 3.8.

Таблиця 3.8 – Можливі події на переходах в інтерфейсі користувача

№	Події на переходах
1	Користувач відкрив веб застосунок сторінку
2	Введення не валідних метрик, наприклад -1
3	Повернення до стану введення метрик
4	Вибір дії - прогнозувати розмір застосунку
5	Повернення до стану введення метрик
6	Вибір дії – закрити сторінку

3.5 Обґрунтування вибору засобів розробки та технологічного стека проєкту

Завершивши етап формалізації функціональних та нефункціональних вимог, а також, розробку специфікацій основних компонентів системи, доцільно перейти до етапу безпосередньої програмної реалізації.

Першочерговим завданням є визначення технологічного стека – набору інструментальних засобів, які дозволять реалізувати спроєктовану архітектуру з мінімальними витратами ресурсів та максимальною ефективністю.

Аналіз вимог до інтерфейсу користувача показав, що клієнтська частина не передбачає складне маніпулювання станом додатка, або інтенсивної динамічної зміни контенту без перезавантаження сторінки, що є характерним для односторінкових застосунків. Основні задачі інтерфейсу зводяться до візуалізації статичного тексту, відображення таблиць, відправки форм з вхідними даними та завантаження файлів.

В зв’язку з цим, використання важковагових UI-фреймворків, таких як React, Vue.js або Angular, було визнано недоцільним. Впровадження даних інструментів, значно ускладнило б архітектуру проєкту, необхідність налаштування збирачів модулів, керування залежностями, без надання суттєвих переваг для даного типу задач. Тому, було прийнято рішення використовувати нативний стек веб-

технологій, HTML5 для розмітки та CSS3 для стилізації, що забезпечує максимальну швидкість завантаження та сумісність з усіма сучасними браузерами.

Хоча дана робота сфокусована на дослідженні проєктів мовою TypeScript, для розробки клієнтської частини власного інструментарію було обрано класичний JavaScript. Використання TypeScript є виправданим в масштабних проєктах зі складною бізнес-логікою на клієнті, де критичною є типізація та підтримка великої кодової бази. В даному випадку, де логіка на клієнті обмежується валідацією форм, відправкою асинхронних запитів AJAX та відображенням індикатора завантаження, налаштування трансляції TypeScript в JavaScript стало б надмірним ускладненням. Функціоналу стандарту ECMAScript 6+ [21] достатньо для реалізації поставлених завдань.

Серверна частина є ядром системи, оскільки саме тут реалізуються алгоритми фільтрації викидів, нормалізації даних та побудови регресійних моделей. Для вибору мови програмування було проведено порівняльний аналіз популярних рішень, таких як C# .NET, Java, TypeScript Node.js/Bun та Python.

Хоча мови C# та Java забезпечують високу продуктивність, вони вимагають написання значного обсягу супровідного коду. Екосистема Node.js, попри свою популярність, має обмежені можливості для складних наукових обчислень. В результаті аналізу, в якості мови серверної розробки було обрано Python. Даний вибір зумовлений домінуванням Python в сфері Data Science та наявністю потужних бібліотек для наукових обчислень. Pandas та NumPy для ефективної обробки табличних даних та матричних операцій, SciPy та Statsmodels для реалізації статистичних тестів та побудови регресійних моделей; Matplotlib/Seaborn для генерації графіків та візуалізації результатів аналізу зображено (рис. 3.8).

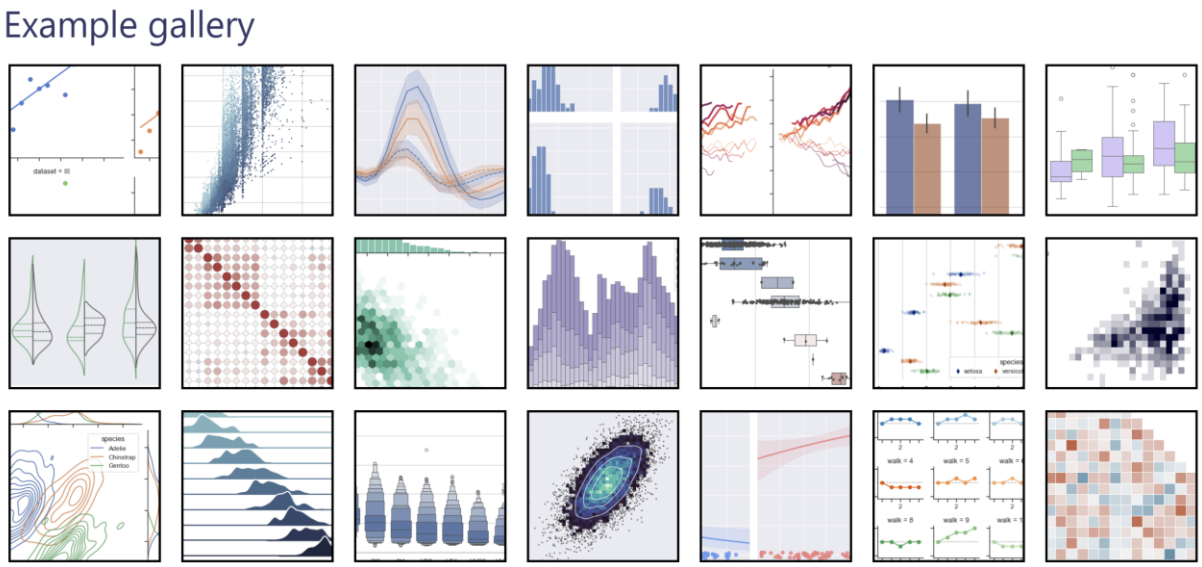


Рисунок 3.8 – Приклад візуалізації за допомогою бібліотеки Seaborn [22]

Для реалізації веб-сервера було обрано фреймворк Django. Згідно з актуальними дослідженнями та опитуваннями розробників, хоча Django вже не вважається найсучаснішим засобом для розробки, проте, все ще залишається одним з найнадійніших фреймворків для швидкої розробки складних веб-систем [23].

Ключовим аргументом на користь Django є його архітектурний патерн MVT Model-View-Template, який забезпечує чітке розділення логіки обробки даних, взаємодії з базою даних та відображення інтерфейсу користувача. Структура взаємодії компонентів у цій архітектурі наведена на рисунку 3.9.

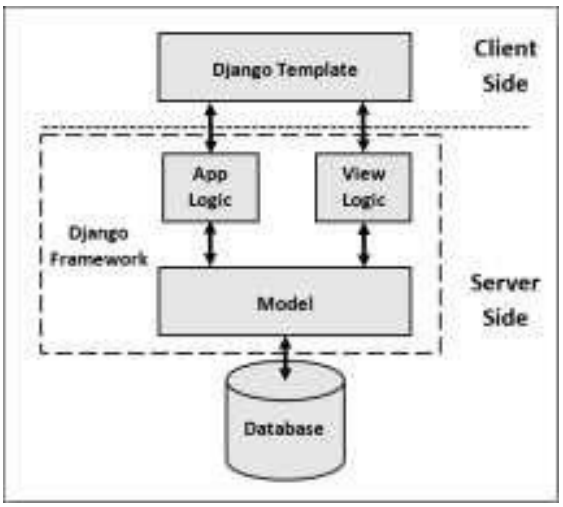


Рисунок 3.9 - Архітектура фреймворку Django MVT [24]

Як видно з рисунка, архітектура Django складається з трьох основних шарів, що функціонують на стороні сервера

– Model – відповідає за логічну структуру даних та взаємодію з реляційною базою даних ORM. В даному випадку, це дозволяє легко зберігати метрики проектів NOC, AAC, TR без написання складних SQL-запитів;

– View (Представлення/Логіка) – обробляє вхідні HTTP-запити, виконує валідацію даних, викликає математичні алгоритми та формує відповідь;

– Template – генерує HTML-сторінки, динамічно підставляючи результати обчислень в статичну розмітку, що відправляється на клієнтську частину.

Окрім архітектурних переваг, Django надає вбудовану адміністративну панель, що дозволяє задовольнити вимогу щодо надання окремого інтерфейсу для адміністратора без необхідності розробляти даний функціонал з нуля (рис. 3.10).

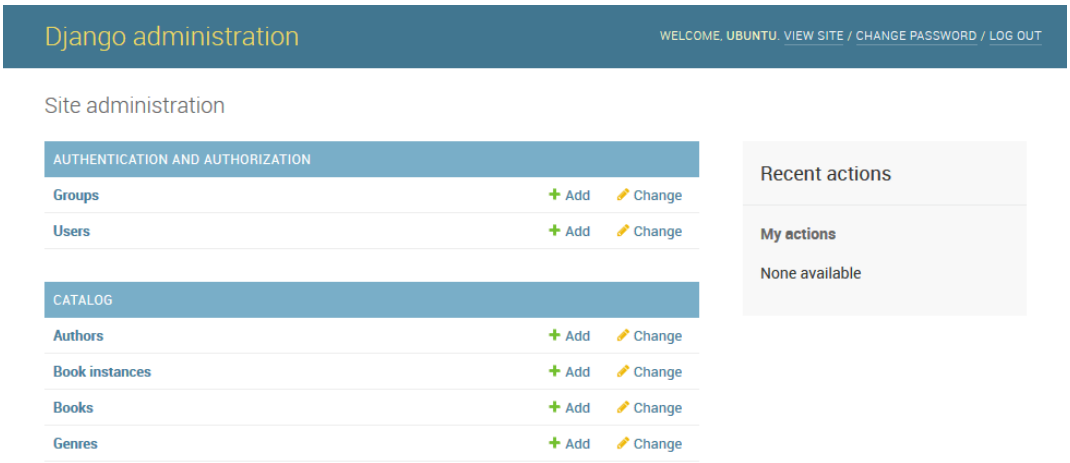


Рисунок 3.10 – Панель адміністратора Django

Використання вбудованої системи автентифікації та розмежування прав доступу, економить ресурси на розробку та підвищує загальний рівень інформаційної безпеки системи.

Обраний технологічний стек HTML/CSS/JS + Python/Django є оптимальним з точки зору балансу між швидкістю розробки, продуктивністю математичних

обчислень та безпекою даних. В якості середовища для розробки було обрано VS Code.

3.7 Реалізація моделей збереження даних засобами Django ORM

Після ініціалізації проєкту на базі фреймворку Django, першочерговим етапом програмної реалізації стала розробка специфікації даних. Фізична реалізація рівня збереження даних була значно спрощена завдяки використанню технології об’єктно-реляційного відображення ORM. В сучасній інженерії програмного забезпечення, ORM виступає як абстрактний прошарок, що вирішує проблему «невідповідності моделей» між об’єктно-орієнтованим кодом програми та реляційною структурою бази даних. Даний патерн дозволяє розробнику маніпулювати даними як звичайними об’єктами мови програмування, автоматично транслюючи ці операції у відповідні SQL-запити до бази даних (рис. 3.11) [25].

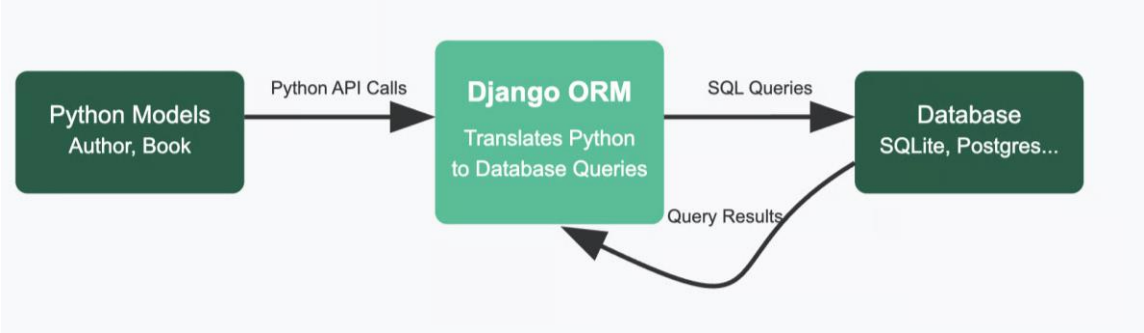


Рисунок 3.11 – Django ORM [25]

Використання Django ORM в даному проєкті, забезпечило ряд критичних переваг:

– абстрагування від SQL – опис моделей виконано виключно мовою Python. файл models.py, що робить код більш читабельним та легким для підтримки;

– безпека даних – вбудовані механізми ORM автоматично екранують вхідні дані, що забезпечує захист від поширених атак типу SQL-ін’єкцій без додаткових зусиль з боку розробника;

– переносимість – оскільки ORM генерує діалект SQL автоматично, система може бути легко перенесена з однієї СУБД на іншу наприклад, з SQLite на PostgreSQL без необхідності переписувати програмний код.

Як приклад практичного застосування підходу Code-First, наведено програмну реалізацію сутності NonLinearModel. Даний клас відповідає за збереження параметрів розрахованої нелінійної моделі та є ключовим елементом системи прогнозування. Програмний код класу, описаний у файлі models.py, наведено на рисунку 3.12.

```
class NonLinearModel(models.Model):
    b0 = models.FloatField()
    b1 = models.FloatField()
    b2 = models.FloatField()
    b3 = models.FloatField()

    r2 = models.FloatField(default=0.0)
    mmre = models.FloatField(default=0.0)
    pred_25 = models.FloatField(default=0.0)

    dataset = models.ForeignKey(Dataset, on_delete=models.CASCADE, related_name='nonlinear_models', null=True, blank=True)

    normalization_method = models.CharField(max_length=100, default='lg10')

    def __str__(self):
        return f"NonLinearModel(b0={self.b0}, b1={self.b1}, b2={self.b2}, b3={self.b3}), normalization_method={self.normalization_method})"
```

Рисунок 3.12 – Вихідний код NonLinearModel в файлі models.py

В наведеному коді, клас успадковується від базового класу models.Model, що надає йому функціонал для взаємодії з базою даних. Структура полів відповідає математичній моделі, обґрунтованій в другому розділі: Поля b_0, b_1, b_2, b_3 мають тип FloatField та призначені для зберігання дійсних значень коефіцієнтів регресії з плаваючою крапкою, що забезпечує необхідну точність обчислень. Поля $r2, mmre, pred\_25$ зберігають розраховані показники якості моделі, що дозволяє швидко відображати їх користувачеві без повторного перерахунку. Атрибут $dataset$, реалізує зв’язок «багато-до-одного» ForeignKey з сутністю $Dataset$. Параметр on_delete=models. CASCADE забезпечує цілісність даних? при видаленні набору даних автоматично будуть видалені всі моделі, побудовані на його основі. Поле $normalization\_method$ зберігає ідентифікатор методу нормалізації за замовчуванням – десятковий логарифм $lg10$, що забезпечує гнучкість архітектури

та можливість розширення системи новими методами обробки даних у майбутньому.

Після програмної реалізації всіх сутностей у файлі models.py, наступним важливим етапом стала синхронізація об’єктної моделі Python з реляційною структурою СУБД. У фреймворку Django, даний процес автоматизовано за допомогою механізму міграцій, який виконує роль системи контролю версій для схеми бази даних. На основі описаних класів, системою було автоматично згенеровано файл первинної міграції 0001_initial.py, фрагмент якого наведено на рисунку 3.13. Даний файл, містить інструкції для створення таблиць, визначення типів полів та накладання обмежень цілісності Primary Keys, Foreign Keys, трансльовані з Python-коду. Застосування міграції дозволило розгорнути фізичну структуру бази даних, без необхідності ручного написання SQL-запитів CREATE TABLE, що виключає ризик синтаксичних помилок та неузгодженості типів даних.

```
        migrations.CreateModel(
            name='LinearModel',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('b0', models.FloatField()),
                ('b1', models.FloatField()),
                ('b2', models.FloatField()),
                ('b3', models.FloatField()),
            ],
        ),
        migrations.CreateModel(
            name='NonLinearModel',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('b0', models.FloatField()),
                ('b1', models.FloatField()),
```

Рисунок 3.13 – Фрагмент коду файлу 0001_initial.py

Для верифікації спроєктованої структури та наочного відображення реляційних зв’язків між створеними таблицями було побудовано діаграму «Сутність-Зв’язок» ER-діаграма, зображену на рисунку 3.14.

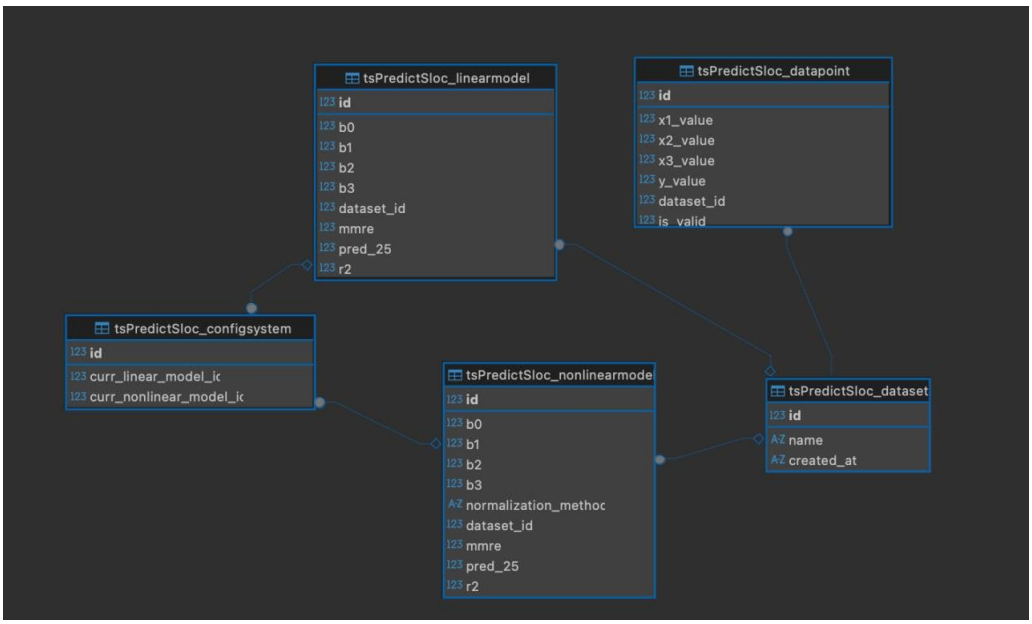


Рисунок 3.14 – ER-діаграми бази даних

Візуалізація підтверджує правильність реалізації закладених архітектурних рішень:

– централізація даних – сутність Dataset виступає кореневим елементом, до якого прив’язані як вхідні дані DataPoint, так і навчені моделі LinearModel, NonLinearModel. На діаграмі чітко простежується зв’язок «один-до-багатьох», один датасет може містити безліч точок даних та бути основою для декількох версій математичних моделей.

– каскадне видалення – реалізовані зв’язки забезпечують посилальну цілісність. Видалення запису з таблиці Dataset призведе до автоматичного видалення всіх пов’язаних з ним метрик та моделей, що запобігає накопиченню даних в системі.

– гнучкість конфігурації – таблиця ConfigSystem має зв’язки з таблицями моделей, але не жорстко прив’язана до конкретних записів, що дозволяє динамічно змінювати активну модель для розрахунків, просто оновлюючи посилання Foreign Key у конфігурації, без необхідності змінювати структуру бази. Сформована схема даних повністю відповідає вимогам предметної області, забезпечує надійне зберігання даних та результатів моделювання, а також, підтримує масштабованість системи при додаванні нових метрик у майбутньому.

Сформована схема даних повністю відповідає вимогам предметної області, забезпечує надійне зберігання історичних даних та результатів моделювання, а також, підтримує масштабованість системи при додаванні нових метрик в майбутньому.

3.8 Випробування ПЗ для оцінювання розміру застосунків TypeScript

Перевірка працездатності програмного засобу та верифікація результатів розрахунків проводилася згідно з розробленою програмою та методикою випробувань (Додаток Б).

Етап випробувань є невід’ємною складовою життєвого циклу розробки. В рамках даної роботи, об’єктом тестування виступає веб-застосунок. Специфіка системи вимагає перевірки не лише коректності роботи інтерфейсу, а й точності математичних обчислень, що виконуються на серверній стороні.

Методологічною основою проведення випробувань обрано стратегію тестування «чорної скриньки» Black Box Testing. Даний підхід дозволяє абстрагуватися від внутрішньої реалізації програмного коду та зосередитися на валідації бізнес-логіки. Згідно з Майерсом, суть методу полягає в подачі на вхід системи наборів тестових даних та порівнянні отриманих вихідних значень із заздалегідь відомими очікуваними результатами [26]. Для даної системи це означає перевірку того, чи коректно програма реагує на введення метрик та чи співпадають розраховані нею прогнози з результатами контрольних прикладів.

Враховуючи архітектуру системи, план тестування структуровано відповідно до розмежування прав доступу. Застосунок має два ізольовані інтерфейси для різних ролей:

– адміністратор – здійснює управління навчальною вибіркою, завантаження даних та ініціює перерахунок параметрів моделі;

– користувач – використовує налаштовану систему для отримання прогнозів розміру нових проектів.

Послідовність випробувань визначено логікою обробки даних в системі. Першочерговому тестуванню підлягає функціонал адміністративної панелі. Для документування процесу було обрано використання стандартного шаблону опису тест-кейсів [28]. Подібний формат, який включає передумови, кроки виконання та очікуваний результат, допомагає систематизувати перевірку та однозначно трактувати успішність проходження тесту.

Функція «Завантаження даних для побудови моделі»

Передумови:

Адміністратор авторизований в системі

Кроки:

Відкрити головну сторінку адмін панелі (рис. 3.15).

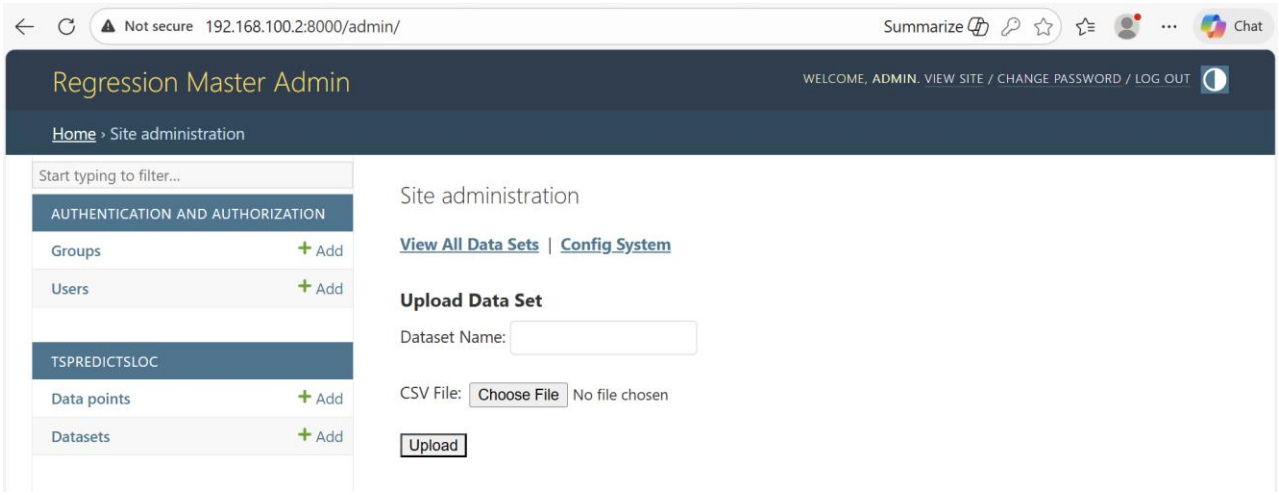


Рисунок 3.15 – Головна сторінка адмін панелі

Ввести назву набору даних

Завантажує *.csv файл з даними

Натиснути на кнопку «Upload»

Очікуваний результат:

Отримати повідомлення, що дані успішно завантажені (рис. 3.16).

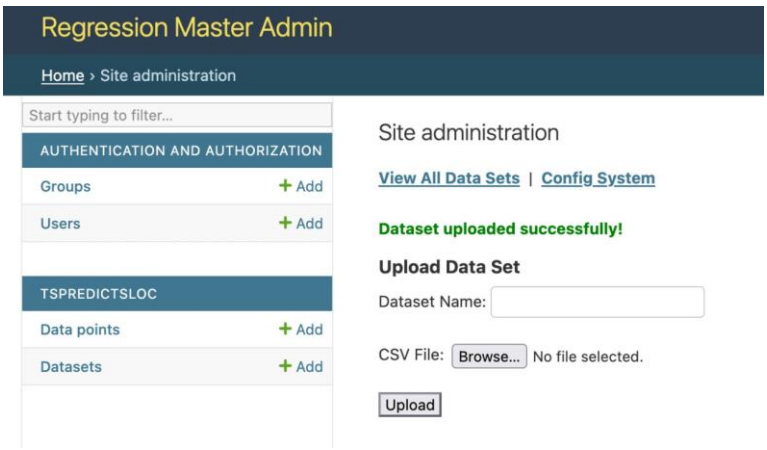


Рисунок 3.16 – Підтвердження успішного завантаження набору даних

В списку наборів даних з’явився, новий набір з іменем вказаним при завантаженні (рис. 3.17).

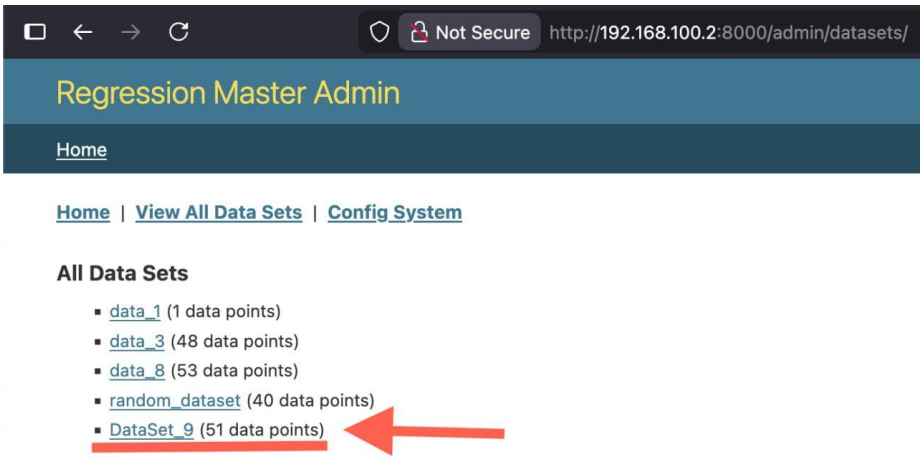


Рисунок 3.17 – Перевірка додавання нового набору даних

Новий набір даних містить конкретні дані (рис. 3.18).

#	X1 NOC	X2 AAV	X3 TR	Y SLOC (TS)
1	642	8.023364486	4751	82548
2	196	6.948979592	832	14896
3	34	2.0	34	1422
4	144	1.111111111	574	7578
5	50	1.0	154	4078

Рисунок 3.18 – Перевірка коректності імпортованих даних

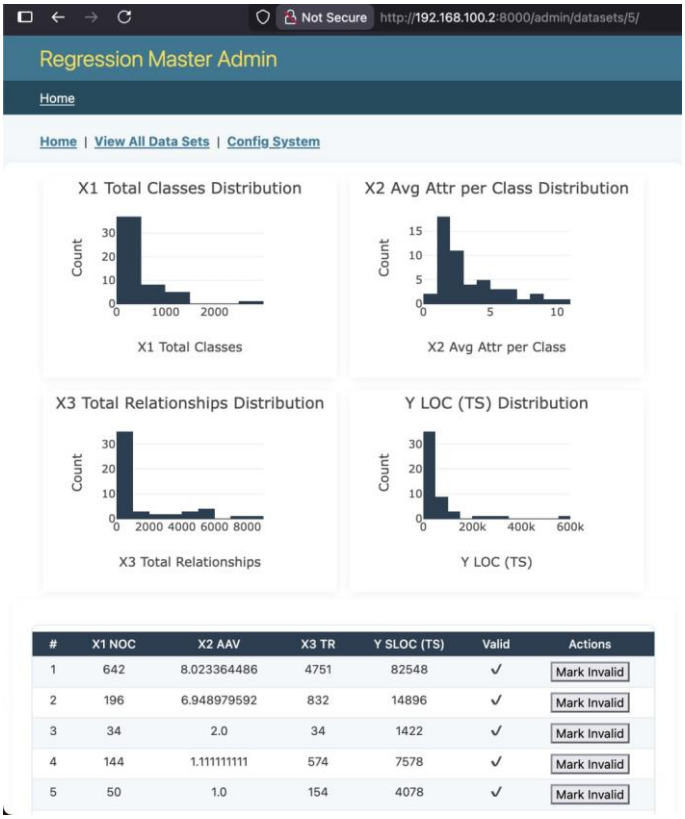


Рисунок 3.19 – Перевірка коректності імпорту набору даних

Тестування функціоналу «Завантаження даних для побудови моделі» пройшло успішно. В системі було створено новий набір даних з назвою «DataSet_9», дані до якого імпортовано з файлу data_9.csv. Як видно з рисунків 3.16-3.19, отримано повідомлення про успішне завантаження, а до списку наборів даних додано новий елемент «DataSet_9». Система коректно побудувала гістограми розподілу метрик, а повний список завантажених даних вивела в табличному вигляді.

Функція «Побудова нелінійної регресійної моделі»

Передумови:

Адміністратор авторизований в системі

Завантажено набір даних

Кроки:

Перейти на сторінку набору даних

В правій половині сторінки натиснути на кнопку «*Non-Linear Model*»

Очікуваний результат:

В правій половині сторінки виведено звіт з побудови нелінійної регресійної моделі (рис. 3.20).

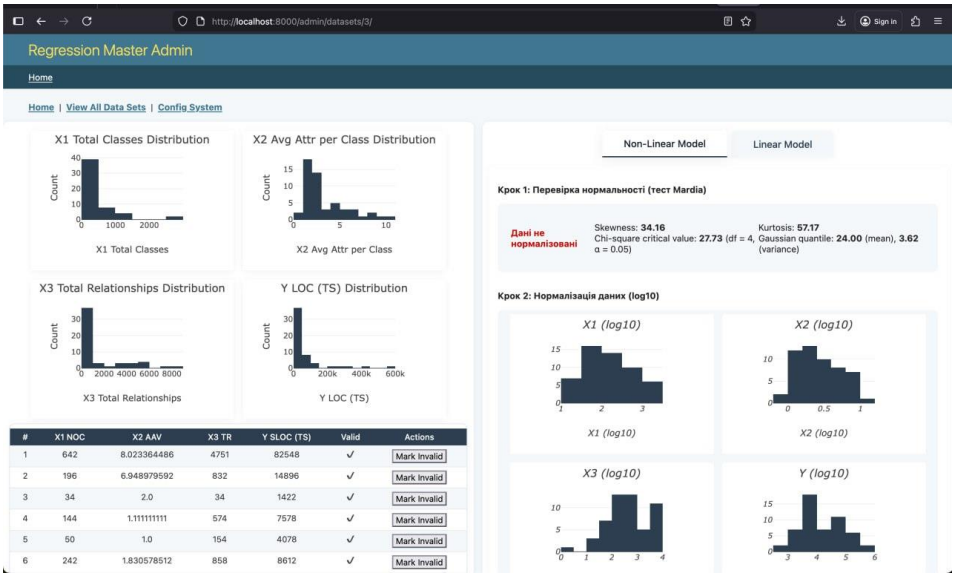


Рисунок 3.20 – Звіт побудови нелінійної моделі (частина 1)

Звіт повинен містити:

1. Результат перевірки тесту Мардіа до нормалізації даних (рис. 3.20).

2. Гістограми розподілення даних після застосування нормалізуючого перетворення (рис. 3.20).

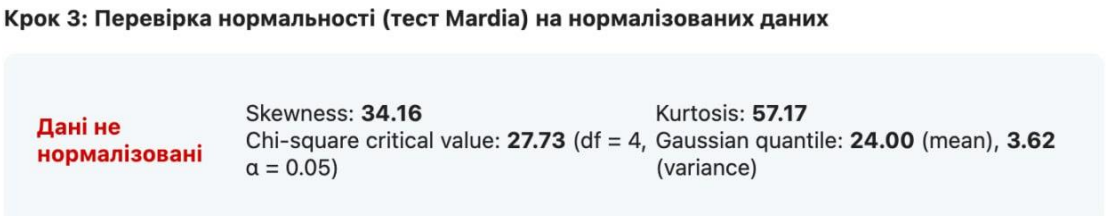


Рисунок 3.21 – Звіт побудови нелінійної моделі (частина 2)

3. Результат перевірки тесту Мардіа після нормалізації даних (рис. 3.21).

Крок 4: Знаходження та видалення викидів

ІТЕРАЦІЯ	ІНДЕКС ВИКИДУ	ЗНАЧЕННЯ	SMD НАЙБІЛЬШОГО ВИКИДУ	КРИТИЧНЕ ЗНАЧЕННЯ	РІЗНИЦЯ (SMD - КРИТИЧНЕ)
1	19	X1: 45 X2: 3.73 X3: 3 Y: 307972	25.75	9.49	16.26
2	24	x1: 11 x2: 0.91 x3: 14 y: 62714	19.65	9.49	10.16
3	37	x1: 59 x2: 2.02 x3: 128 y: 270559	21.64	9.49	12.15
4	50	x1: 354 x2: 2.19 x3: 1086 y: 592931	22.02	9.49	12.53
5	53	x1: 2730 x2: 3.56 x3: 7964 y: 52217	14.30	9.49	4.81
6	14	x1: 2807 x2: 2.43 x3: 3084 y: 402434	10.23	9.49	0.74
7	35	x1: 90 x2: 2.77 x3: 48	10.34	9.49	0.85

Рисунок 3.22 - Звіт побудови нелінійної моделі (частина 3)

4. Таблиця по знаходженню та видаленню викидів (рис. 3.22).

Крок 5: Перевірка нормальності на нормалізованих даних (тест Mardia) після видалення викидів

Дані нормалізовані

Skewness: **2.35**
Chi-square critical value: **28.00** (df = 4, α = 0.05)

Kurtosis: **20.52**
Gaussian quantile: **24.00** (mean), **4.17** (variance)

Крок 5: Формула побудованої моделі

$$Y = 10^{1.6202} X_1^{1.2835} X_2^{0.5953} X_3^{-0.2263}$$

Крок 6: Метрики якості моделі

$$R2 = 0.8995 | \text{MMRE} = 0.3720 | \text{Pred}(0.25) = 0.4286$$

Зберегти модель

Рисунок 3.23 - Звіт побудови нелінійної моделі (частина 4)

5. Результат перевірки теста Мардіа після нормалізації та вилучення викидів (рис. 3.23).

6. Отримана формула нелінійної регресійної моделі (рис. 3.23).

7. Отримані метрики якості нелінійної моделі (рис. 3.23).

Тестування функціоналу «Побудова нелінійної регресійної моделі» пройшло успішно. На основі набору даних «$DataSet\_9$» було розраховано нову трьохфакторну нелінійну модель для оцінювання розміру застосунків мовою TypeScript. Система згенерувала детальний звіт, що відображає всі етапи побудови моделі, як показано на рисунках 3.20-3.23. Звіт містить фінальну формулу регресії та розраховані метрики якості. Процес розрахунку моделі та формування звіту зайняв менше однієї секунди, що підтверджує відповідність системи вимогам щодо швидкодії.

Функція «Побудова лінійної моделі»

Передумови:

Адміністратор авторизований в системі

Завантажено набір даних

Кроки:

Перейти на сторінку набору даних

В правій половині сторінки натиснути на кнопку «*Linear Model*»

Очікуваний результат:

В правій половині сторінки виведено звіт з побудови регресійної моделі рисунок 3.24.

Звіт повинен містити:

1. Формулу лінійної моделі (рис. 3.24)

2. Метрики якості моделі (рис. 3.24)

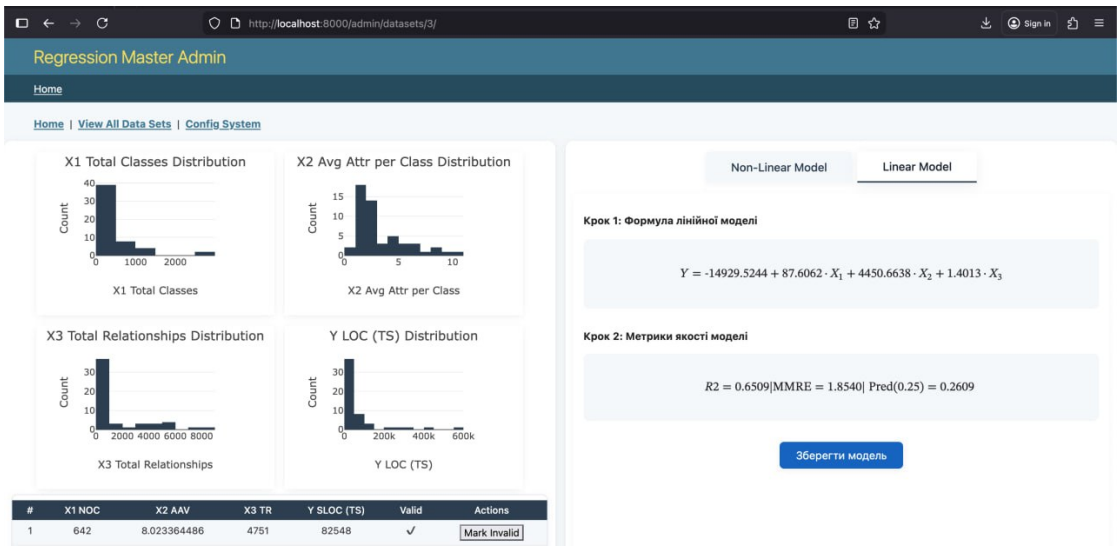


Рисунок 3.24 – Звіт побудови лінійної моделі

Тестування функції «Побудова лінійної моделі» пройшло успішно. Як видно з рисунка 3.24, система коректно виконала розрахунки та згенерувала звіт. В звіті відображено формулу лінійної моделі та ключові показники якості, що дозволяє адміністратору миттєво оцінити адекватність лінійного підходу для поточного набору даних. Операція була виконана без затримок, що відповідає вимогам до швидкодії.

Функція «Ручне керування вибіркою»

Передумови:

Адміністратор авторизований в системі

Завантажено набір даних

Кроки:

Перейти на сторінку набору даних

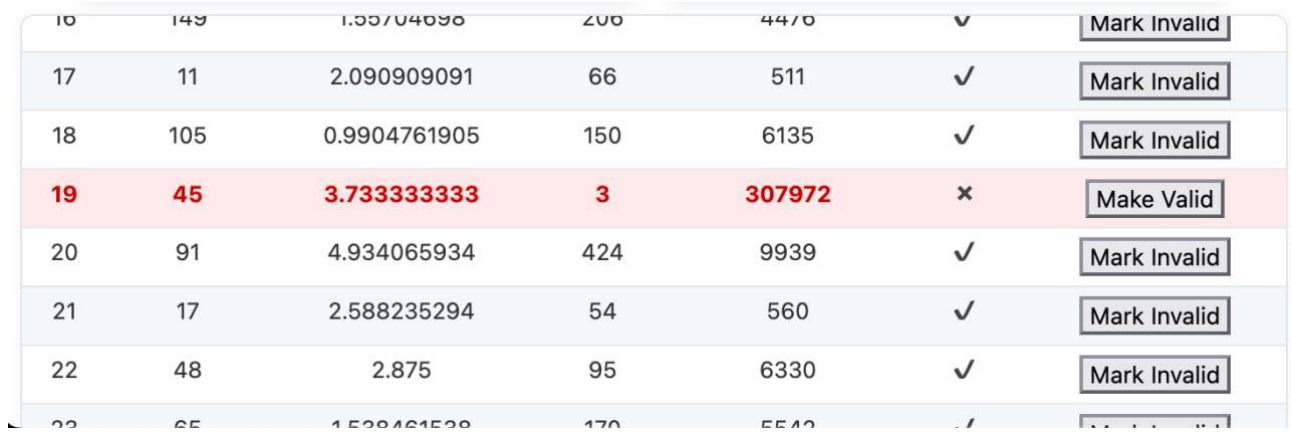
Вилучити значення з набору даних

Очікуваний результат:

Вилучене значення позначається червоним в таблиці значень

Звіт побудови моделі оновлюється відповідно до оновленого набору даних

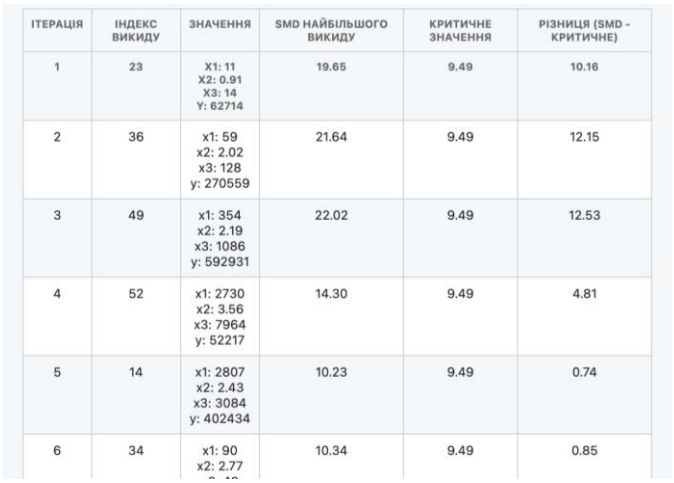
Для верифікації коректності роботи даної функції було проведено тест з ручним вилученням значення, який має суттєвий вплив на параметри моделі. Зокрема, було обрано спостереження з індексом 19 (рис. 3.25), яке, згідно з попереднім звітом (рис. 3.22), класифікувалося системою як викид. Метою тесту було перевірити, чи врахує система дану ручну зміну при повторному автоматичному аналізі.



16	149	1.55704698	206	4476	✓	Mark Invalid
17	11	2.090909091	66	511	✓	Mark Invalid
18	105	0.9904761905	150	6135	✓	Mark Invalid
19	**45**	**3.733333333**	**3**	**307972**	×	Make Valid
20	91	4.934065934	424	9939	✓	Mark Invalid
21	17	2.588235294	54	560	✓	Mark Invalid
22	48	2.875	95	6330	✓	Mark Invalid
23	65	1.538461538	170	5542	✓	Mark Invalid

Рисунок 3.25 – Вилучення значення з індексом 19

Одразу, після зміни статусу запису система автоматично перебудувала звіт нелінійної регресії. Аналіз оновленого звіту показав, що на етапі фільтрації Крок 4 кількість автоматично виявлених викидів зменшилася з 7 до 6 (рис. 3.26).



ІТЕРАЦІЯ	ІНДЕКС ВИКИДУ	ЗНАЧЕННЯ	SMD НАЙБІЛЬШОГО ВИКИДУ	КРИТИЧНЕ ЗНАЧЕННЯ	РІЗНИЦЯ (SMD - КРИТИЧНЕ)
1	23	X1: 11 X2: 0.91 X3: 14 Y: 62714	19.65	9.49	10.16
2	36	x1: 59 x2: 2.02 x3: 128 y: 270559	21.64	9.49	12.15
3	49	x1: 354 x2: 2.19 x3: 1086 y: 592931	22.02	9.49	12.53
4	52	x1: 2730 x2: 3.56 x3: 7964 y: 52217	14.30	9.49	4.81
5	14	x1: 2807 x2: 2.43 x3: 3084 y: 402434	10.23	9.49	0.74
6	34	x1: 90 x2: 2.77	10.34	9.49	0.85

Рисунок 3.26 – Звіт крок 4 після вилучення значення з вибірки даних

Отримані результати свідчать про те, що система коректно виключила значення із розрахунків ще до початку автоматичної фільтрації, і модель була успішно перерахована на основі актуалізованого набору даних. Функціонал ручного керування вибіркою працює коректно, забезпечуючи динамічне оновлення математичної моделі та візуалізацію змін в реальному часі. Тест пройдено успішно.

Функція «Обробка не коректно введених метрик, для оцінки розміру застосунку мовою TypeScript»

Передумови:

Користувач знаходиться в мережі з доступом до веб-застосунку

Кроки:

Відкрити сторінку оцінки розміру застосунку мовою TypeScript

Ввести не валідні значення метрик

Очікуваний результат:

При натисканні кнопки «Оцінити розмір застосунку TS» виводиться помилка

Розрахунок розміру застосунку не відбувається

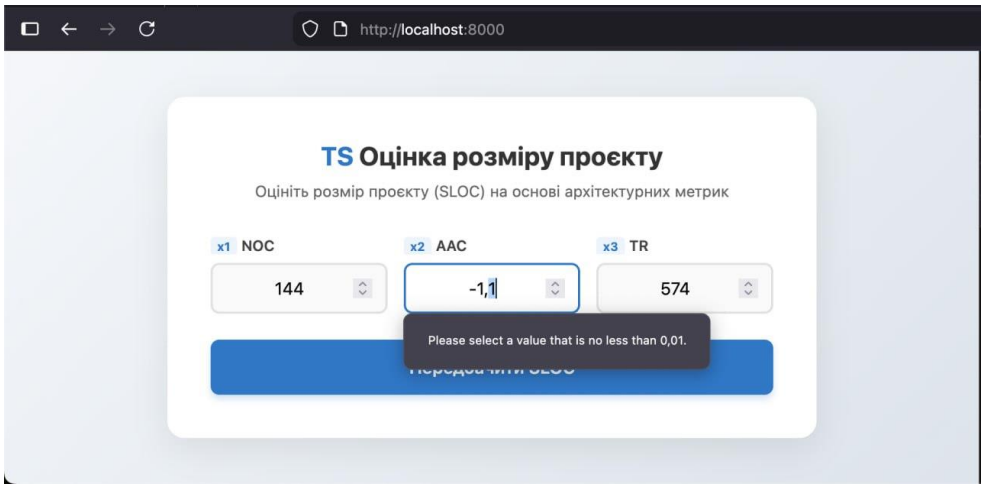


Рисунок 3.27 – Помилка при введенні не валідних даних

Під час перевірки, в поле метрики AAC було навмисно введено від'ємне значення. Система ідентифікувала помилку та вивела повідомлення: «Please enter a value greater than 0» Будь ласка, введіть значення більше 0 (рис. 3.27). При цьому,

розрахунок прогнозу було автоматично заблоковано. Система перейшла в режим очікування введення коректних даних, що свідчить про успішну реалізацію механізмів валідації на стороні клієнта.

Функція «Оцінювання розміру застосунку мовою програмування TypeScript»

Передумови:

Користувач знаходиться в мережі з доступом до веб-застосунку

Кроки:

Відкрити сторінку оцінки розміру застосунку мовою TypeScript

Очікуваний результат:

Виведення двох значень розмір проєкту в SLOC згідно нелінійної моделі, та згідно лінійної моделі

Для перевірки було введено параметри проєкту середньої складності з метриками $NOC = 144$, $AAC = 1.11$, $TR = 574$. Реальний розмір даного проєкту складає $Y\ SLOC(TS) = 7578$. Система успішно обробила запит і за час менше 1 секунди вивела результати на екран (рис. 3.28). Користувачеві було надано порівняльну характеристику: прогноз за удосконаленою нелінійною моделлю та прогноз за базовою лінійною.

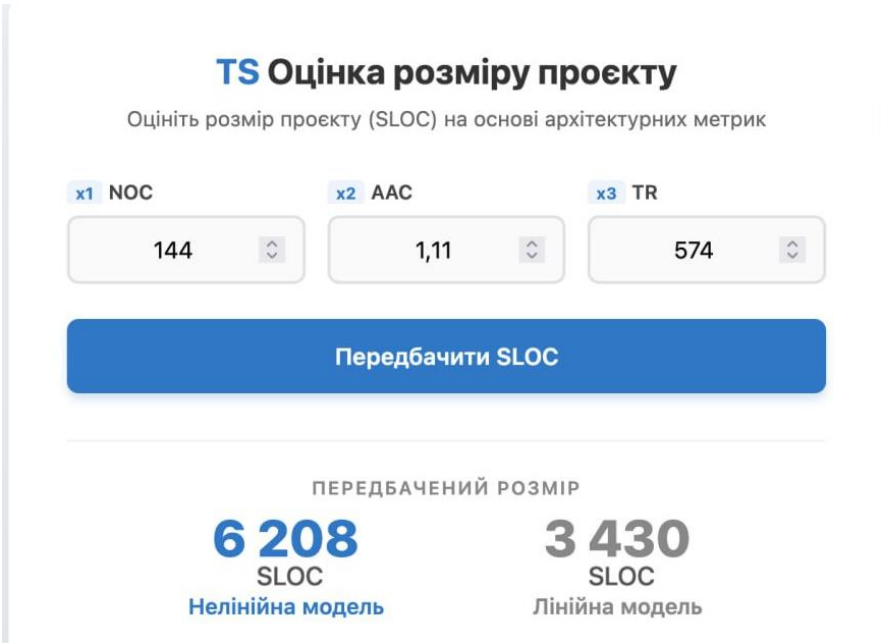


Рисунок 3.28 – Результат прогнозування розміру застосунку

Отримані результати дозволили наочно оцінити різницю в підходах та отримати більш обґрунтовані дані для прийняття управлінських рішень. Тест підтвердив працездатність математичної моделі системи та коректність інтеграції між клієнтською та серверною частинами.

В результаті проведеного тестування, можна стверджувати, що розроблений веб-застосунок функціонує коректно та стабільно. Перевірка за методикою «чорної скриньки» охопила всі ключові сценарії використання, від адміністративних функцій завантаження наборі даних, створення моделей, ручного вилучення значень з вибіркою даних до користувацьких запитів на прогнозування.

3.9 Висновки

В третьому розділі роботи вирішено комплекс інженерно-технічних завдань, спрямованих на практичну реалізацію удосконаленої математичної моделі у вигляді завершеного програмного продукту. Створення автоматизованої системи дозволило перевести теоретичні напрацювання попередніх етапів дослідження в площину практичного застосування, надавши користувачам зручний інструмент для оцінювання розміру проєктів. Ключовим досягненням даного етапу стала інтеграція розробленого математичного апарату, що включає методи нелінійної регресії, в сучасну веб-орієнтовану архітектуру, що забезпечило доступність та надійність процесу оцінювання розміру застосунку.

На початковому етапі проєктування було проведено детальну формалізацію функціональних та нефункціональних вимог до системи. Шляхом аналізу предметної області та потреб потенційних користувачів визначено, що критично важливими критеріями є кросплатформеність, простота розгортання оновлень моделі та захист інтелектуальної власності, закладеної в математичних алгоритмах. На основі даного аналізу, було обґрунтовано вибір архітектури клієнт-серверного веб-застосунку. Представлений підхід дозволив ізолювати обчислювальне ядро на серверній стороні, забезпечивши захист алгоритмів від несанкціонованого копіювання, та надати користувачам доступ до системи через стандартний веб-

браузер без необхідності встановлення додаткового програмного забезпечення чи налаштування середовища виконання. Дане рішення також спростило процес адміністрування, оскільки оновлення коефіцієнтів моделі відбувається централізовано і миттєво стає доступним для всіх користувачів.

Важливим етапом роботи стало обґрунтування та вибір технологічного стека. Порівняльний аналіз сучасних засобів розробки показав, що оптимальним рішенням для реалізації серверної частини є мова програмування Python в поєднанні з фреймворком Django та бібліотеками для наукових обчислень NumPy і Pandas. Даний вибір був продиктований необхідністю ефективної роботи з матричними операціями при розрахунку відстані Махаланобіса та реалізації методу найменших квадратів. Використання Django дозволило швидко розгорнути надійний веб-сервер з вбудованою системою адміністрування, що значно зекономило ресурси розробки. Для клієнтської частини було обрано нативні веб-технології, що дозволило уникнути надмірної складності архітектури та забезпечити високу швидкість завантаження інтерфейсу.

Процес розробки супроводжувався детальним моделюванням за допомогою уніфікованої мови UML. Побудова діаграм варіантів використання дозволила чітко розмежувати зони відповідальності між ролями «Адміністратор» та «Користувач», визначивши сценарії завантаження навчальних даних та безпосереднього прогнозування. Особливу увагу було приділено моделюванню динамічних аспектів системи через діаграми діяльності, які візуалізували складні процеси, зокрема ітераційну процедуру фільтрації викидів та логіку валідації вхідних даних. Отримана деталізація на етапі проєктування стала основою для безпомилкової імплементації бізнес-логіки та забезпечила відповідність програмного коду теоретичній моделі.

Реалізація рівня збереження даних була виконана з використанням технології ORM, що дозволило спроєктувати нормалізовану схему бази даних, орієнтовану на об’єктну модель Python. Розроблена структура даних, що включає сутності для зберігання історичних наборів даних, параметрів навчених моделей та системної конфігурації, забезпечує цілісність інформації та гнучкість системи. Важливим

архітектурним рішенням стало збереження не лише самих прогнозів, а й коефіцієнтів регресійних рівнянь та метаданих про методи нормалізації, що дозволяє системі динамічно адаптуватися до змін у вхідних даних без необхідності переписування програмного коду.

Завершальним етапом роботи, стало комплексне випробування програмного засобу за методикою «чорної скриньки», яке підтвердило працездатність та ефективність розробленого рішення. Тестування адміністративної панелі довело коректність роботи модулів імпорту даних та автоматичного навчання моделі, а перевірка інтерфейсу користувача підтвердила надійність механізмів валідації вхідних метрик. Ключовим результатом тестування стала верифікація точності прогнозування на реальному проєкті. Порівняльний аналіз показав, що розроблена система, використовуючи нелінійну модель з врахуванням трьох факторів, надає прогноз, який значно ближчий до фактичного розміру проєкту, ніж результати базової лінійної моделі. Зокрема, для тестового проєкту нелінійна модель продемонструвала адекватну оцінку обсягу коду, тоді як лінійна модель суттєво занизила результат через неможливість врахувати фактор архітектурної зв’язності.

В третьому розділі розроблено повнофункціональну, надійну та захищену інформаційну систему. Вона автоматизує застосування удосконаленої трьохфакторної нелінійної моделі, забезпечуючи високу точність прогнозування розміру TypeScript-застосунків та зручність використання для кінцевого споживача. Програмний продукт повністю відповідає висунутим функціональним та нефункціональним вимогам, успішно пройшов етап верифікації та готовий до впровадження у виробничий процес для підтримки прийняття рішень на ранніх етапах розробки програмного забезпечення.

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВІД РОЗРОБКИ ТА ВПРОВАДЖЕННЯ ВЕБ-ДОДАТКУ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ЗАСТОСУНКІВ МОВОЮ TYPESCRIPT

4.1 Вступ

Розробка та впровадження нових програмних засобів у виробничий процес ІТ-компанії завжди пов’язані з виділенням значних матеріальних, трудових та фінансових ресурсів. Тому, окрім вирішення суто технічних задач, критично важливим етапом проєктування є обґрунтування економічної доцільності створення продукту.

Дана кваліфікаційна робота присвячена розробці веб-застосунку для оцінювання розміру програмного забезпечення на мові TypeScript використовуючи удосконалену нелінійну регресійну модель. Актуальність впровадження даної системи зумовлена необхідністю мінімізації ризиків на етапі планування проектів. Як свідчить статистика, помилки в оцінці трудомісткості на ранніх етапах призводять до зриву дедлайнів та перевищення бюджету, що тягне за собою штрафні санкції та втрату репутації компанії-розробника.

Економічна ефективність розроблюваної системи визначається шляхом зіставлення витрат на її створення з отриманим корисним ефектом від її експлуатації. В контексті даної роботи, джерелами економічного ефекту є пряма економія часу, оскільки, автоматизація розрахунків за удосконаленою нелінійною моделлю дозволяє скоротити час роботи висококваліфікованих фахівців з декількох годин ручного аналізу до декількох хвилин. Підвищення якості управлінських рішень – використання більш точної моделі дозволяє уникнути фінансових втрат, пов’язаних з недооцінкою складності проєкту.

Основним критерієм доцільності розробки виступає річний економічний ефект $E_{\text{р}}$ – показник, що характеризує загальну економію коштів підприємства за рік використання програми за вирахуванням нормативних витрат на її створення.

4.2 Розрахунок витрат на створення та впровадження програмного забезпечення

Витрати на створення програмного продукту визначаються як сума всіх грошових витрат, понесених в процесі проєктування, кодування, тестування та впровадження системи. Для розрахунку кошторису витрат використовується метод прямої калькуляції за статтями витрат.

- основна та додаткова заробітна плата розробників.
- відрахування на соціальні заходи.
- матеріальні витрати, канцелярські товари, папір тощо.
- витрати на електроенергію.
- оренда серверних потужностей, хостинг, домен.
- амортизаційні відрахування.
- накладні витрати.

Розробка веб-застосунку виконується одним інженером-програмістом Fullstack-розробником протягом 3 місяців включаючи етапи аналізу, розробки та тестування. Місячний посадовий оклад (O_m) встановлено на рівні середньої ринкової зарплати для спеціаліста – 30 000 грн.

Фонд основної заробітної плати ($З_{осн}$) розраховується за формулою:

$$З_{осн} = O_m \cdot T_p, \quad (4.1)$$

де T_p – тривалість розробки (3 місяці).

$$З_{осн} = 30000 \cdot 3 = 90000 \text{ грн.}$$

Додаткова заробітна плата ($З_{дод}$) враховує виплати за відпустки, премії та надбавки. Вона приймається в розмірі 15% від основної заробітної плати:

$$З_{дод} = З_{осн} \cdot 0.15 = 90000 \cdot 0.15 = 13500 \text{ грн.} \quad (4.2)$$

Відрахування на соціальні заходи ЄСВ становлять 22% від суми основної та додаткової заробітної плати:

$$З_{соц} = (З_{осн} + З_{дод}) \cdot 0.22 = (90000 + 13500) \cdot 0.22 = 22770 \text{ грн.} \quad (4.3)$$

Для виконання робіт використовується персональний комп'ютер вартістю 35000 грн. Норма амортизації комп'ютерної техніки (N_a) становить 25% на рік. Амортизаційні відрахування (А) за період розробки 3 місяці або 0.25 року розраховуються як:

$$А = C_{ПК} \cdot N_a \cdot T_{роки} = 35000 \cdot 0.25 \cdot 0.25 = 2187.5 \text{ грн.} \quad (4.4)$$

Витрати на електроенергію ($З_{ел}$) визначаються за формулою:

$$З_{ел} = W \cdot T_{год} \cdot K_{вик} \cdot C_{кВт}. \quad (4.5)$$

де

W = 0.2 кВт – сумарна потужність обладнання (ноутбук + монітор);

$T_{год} = 21 \cdot 8 \cdot 3 = 504$ години – загальний фонд робочого часу 21 робочий день, 8 годин, 3 місяці;

$K_{вик} = 0.9$ – коефіцієнт використання потужності;

$C_{кВт} = 8.00$грн – середній тариф на електроенергію для комерційних споживачів.

$$З_{ел} = 0.2 \cdot 504 \cdot 0.9 \cdot 8.00 = 725.76 \text{ грн.}$$

Специфіка веб-застосунку вимагає витрат на оренду VPS-сервера для розгортання середовища (Backend + Frontend) та придбання доменного імені.

Враховано витрати на супутні матеріали (папір, канцелярія). Розрахунок наведено в таблиці 4.1.

Таблиця 4.1 – Розрахунок матеріальних витрат та оренди серверів

№	Найменування витрат	Вартість одиниці, грн	Кількість	Сума, грн
1	Оренда VPS-сервера	450,00	3	1350,00
2	Реєстрація доменного імені	550,00	1	550,00
3	Матеріальні витрати	300,00	1	300,00
	Разом ($З_{мат}$)			2200,00

Накладні витрати ($З_{накл}$) покривають витрати на управління, утримання приміщення, зв'язок та організацію робочого процесу. Вони розраховуються у відсотковому відношенні до основної заробітної плати. Приймаємо коефіцієнт накладних витрат рівним 40%:

$$З_{накл} = З_{осн} \cdot 0.40 = 90000 \cdot 0.40 = 36000 \text{ грн.} \quad (4.6)$$

Підсумуємо всі розраховані статті витрат для визначення повної собівартості програмного продукту (К) у таблиці 4.2.

Таблиця 4.2 – Калькуляція кошторисної вартості розробки ПЗ

№	Стаття витрат	Сума, грн	Питома вага, %
1	Основна заробітна плата	90 000,00	53,7%
2	Додаткова заробітна плата	13 500,00	8,1%
3	Відрахування на соціальні заходи ЄСВ	22 770,00	13,6%
4	Матеріальні витрати та оренда серверів	2 200,00	1,3%
5	Амортизація обладнання	2 187,50	1,3%
6	Електроенергія	725,76	0,4%
7	Накладні витрати	36 000,00	21,5%
	Повна собівартість К	167 383,26	100%

Загальні капітальні вкладення на створення та впровадження веб-застосунку становлять 167383,26 грн. Отримана сума є базовою величиною для розрахунку показників економічної ефективності та терміну окупності проєкту.

4.3 Розрахунок економічної ефективності розробки програмного забезпечення

Економічна доцільність впровадження розробленого програмного забезпечення визначається шляхом оцінки прямих фінансових вигод, отриманих в результаті автоматизації процесів. За експертною оцінкою фахівців підприємства, впровадження програмного забезпечення еквівалентне заміні 0,6 менеджера, оскільки система суттєво прискорює виконання розрахунків, автоматизує рутинні операції та дозволяє регулярно виконувати обчислення для аналізу динаміки показників без залучення додаткових трудових ресурсів. Місячний посадовий оклад менеджера встановлено на рівні середньої ринкової зарплати для спеціаліста – 40 000 грн.

Річна заробітна плата одного менеджера становить:

$$40000 \cdot 12 \ = 480\,000\text{грн.}$$

Відповідно, сума вивільнених коштів в рік за рахунок заміщення 0,6 штатної одиниці дорівнює:

$$480000 \cdot \ 0.6 \ = 288\,000\text{грн.}$$

Експлуатаційні витрати складаються з ціни оренди серверів, володіння домену, та складають:

$$\text{В}_{\text{екс}} = 3_{vps} \cdot 12 + 3_{domen} \ = 450 \cdot 12 + 550 = 5950\text{грн} \qquad (4.7)$$

З урахуванням експлуатаційних витрат на функціонування програмного забезпечення, які становлять 5950 грн на рік, чисті вивільнені кошти складуть:

$$\Delta C_n = C_{\text{вив}} - C_{\text{експ}} = 288000 - 5950 = 282050\text{грн.} \quad (4.8)$$

Річний економічний ефект від упровадження програмного продукту визначається за формулою:

$$E_{\text{рік}} = \Delta C_n - E_n \cdot k, \quad (4.9)$$

де

E_n – нормативний коефіцієнт ефективності, що приймається рівним коефіцієнту амортизації (0,6);

k – одноразові витрати на розробку та впровадження програмного забезпечення.

Підставивши значення, отримаємо:

$$E_{\text{рік}} = 282050 - 0{,}6 \cdot 167\,383{,}26 = 181620{,}04$$

Отриманий результат підтверджує позитивний економічний ефект від упровадження програмного забезпечення вже протягом першого року експлуатації. Строк окупності системи визначається за формулою:

$$T = \frac{k}{\Delta C_n} = \frac{167383{,}26}{282050{,}00} = 0{,}59 \text{ року} \quad (4.10)$$

Строк окупності витрат на розробку та впровадження веб-додатку становить 7-8 місяців.

4.4 Висновки

В четвертому розділі були виконані розрахунки витрат на створення, впровадження та експлуатацію програмного забезпечення для оцінювання розміру застосунків мовою TypeScript. Визначено одноразові витрати на розробку програмного продукту та щорічні експлуатаційні витрати, необхідні для його функціонування в умовах практичного використання.

В ході розрахунків, було проведено оцінку економічної ефективності впровадження розробленого програмного забезпечення з урахуванням скорочення трудових витрат управлінського персоналу за рахунок автоматизації процесів розрахунку та аналізу даних. Показано, що використання програмного продукту дозволяє підвищити продуктивність роботи, зменшити часові витрати на виконання аналітичних операцій та забезпечити регулярний моніторинг показників.

На основі отриманих результатів, визначено річний економічний ефект і строк окупності програмного забезпечення. Розрахунки підтверджують, що строк окупності становить менше одного року, що є граничним показником економічної ефективності програмних рішень. Розроблене програмне забезпечення є економічно обґрунтованим, доцільним та вигідним для впровадження.

5 ОХОРОНА ПРАЦІ

5.1 Загальні визначення та актуальні питання охорони праці

Охорона праці – це комплекс правових, соціально-економічних, організаційних, технічних, санітарно-гігієнічних та профілактичних заходів і засобів, спрямованих на забезпечення безпеки, збереження здоров’я та підтримання працездатності працівників, в процесі виконання трудових обов’язків.

Оскільки абсолютно безризикових та повністю безпечних виробництв не існує, головною метою охорони праці є максимальне зниження ймовірності травмування або захворювання працівника та створення умов, що забезпечують комфорт і високу продуктивність. Основоположним принципом даної сфери є визнання життя та здоров’я людини вищою цінністю порівняно з виробничими результатами. Роботодавець зобов’язаний формувати безпечні умови праці, запроваджувати заходи щодо усунення ризиків, організовувати навчання персоналу та дотримуватися вимог чинного законодавства.

Питання забезпечення охорони праці має державний пріоритет. Відповідно, 14 жовтня 1992 року було ухвалено Закон України «Про охорону праці», в якому визначено ключові положення щодо регулювання представленої сфери. Даний закон поширюється на всі підприємства та організації незалежно від форми власності та виду діяльності.

Стаття 4 Закону України «Про охорону праці» визначає базові принципи державної політики в сфері безпеки праці. Вони одночасно є принципами правового інституту охорони праці, які застосовуються у випадках відсутності конкретних норм та задають стратегічний напрям розвитку законодавства. Розглянуті принципи відображають зобов’язання держави щодо забезпечення безпечних умов праці та покликані регулювати діяльність відповідних органів в майбутньому.

З огляду на це, принципи, передбачені статтею 4 Закону «Про охорону праці», мають важливе регулятивне значення. В документі визначено наступні основні засади:

– пріоритетне значення життя та здоров'я працівників над будь-якими виробничими показниками, а також, повна відповідальність роботодавця за створення безпечних та нешкідливих умов праці;

– комплексний підхід до розв'язання питань охорони праці на основі державних цільових програм з врахуванням соціально-економічних тенденцій, а також, досягнень науки, техніки та медицини;

– забезпечення соціального захисту працівників та гарантія повного відшкодування шкоди особам, які постраждали внаслідок нещасних випадків на виробництві або отримали професійні захворювання;

– встановлення єдиних нормативних вимог в сфері охорони праці для всіх підприємств, незалежно від форми власності, галузі діяльності та організаційної структури;

– застосування економічних механізмів в системі управління охороною праці, зокрема впровадження податкових пільг, що стимулюють створення безпечних умов праці, а також, участь держави у фінансуванні відповідних заходів;

– організація навчання населення, професійної підготовки та підвищення кваліфікації працівників щодо вимог, правил та сучасних методів забезпечення охорони праці;

– забезпечення узгодженої взаємодії між державними органами, установами, громадськими організаціями та іншими суб'єктами, які займаються питаннями безпеки праці, гігієни та охорони здоров'я, а також, розвиток співробітництва в даній сфері.

Управління системою охорони праці на державному рівні здійснюють такі органи: Кабінет Міністрів України, центральні органи виконавчої влади, зокрема державний комітет з нагляду за охороною праці, відповідні міністерства, місцеві ради та їхні виконавчі структури. Контроль за дотриманням законодавства та нормативних документів в сфері охорони праці забезпечують державний комітет з

нагляду за охороною праці, державний комітет з питань атомної та радіаційної безпеки, пожежний нагляд, а також, санітарно-епідеміологічні служби. Громадський контроль покладено на трудові колективи через уповноважених представників та профспілкові організації.

Функцію вищого нагляду за неухильним виконанням трудового законодавства органами влади, підприємствами, установами, організаціями та посадовими особами здійснює Генеральний прокурор України.

Стандартизація відіграє ключову роль у формуванні безпечних та здорових умов праці. Вона дає можливість впроваджувати ефективні технічні заходи, систематизувати та вдосконалювати нормативно-технічну документацію, що регулює вимоги безпеки. Починаючи з 1972 року створюється Система стандартів безпеки праці – комплекс взаємопов'язаних документів, спрямованих на підвищення рівня охорони праці. Дана система визначає загальні норми та вимоги щодо небезпечних і шкідливих виробничих факторів, правила безпечної експлуатації обладнання та технологічних процесів, вимоги до засобів індивідуального захисту, а також, методи оцінки рівня безпечності умов праці.

Охорона праці є одним з найважливіших інститутів трудового права та має безпосереднє практичне значення. Порушення вимог безпеки становить реальну загрозу життю та здоров'ю працівників. Водночас, роботодавець або уповноважена ним особа несе сувору відповідальність, в тому числі, кримінальну, за недотримання правил охорони праці.

Однією з характерних рис сучасного суспільного розвитку є розширення сфер діяльності, пов'язаних з використанням інформаційних технологій. Масове застосування персональних комп'ютерів, з одного боку, спростило роботу людини, а з іншого, загострило проблеми, пов'язані зі збереженням здоров'я користувачів, що зумовлює необхідність удосконалення існуючих вимог та створення нових підходів до організації робочих місць, проведення профілактичних заходів і мінімізації негативного впливу роботи з ПК на організм людини. Питання забезпечення охорони праці під час роботи з комп'ютерною технікою залишається актуальним та важливим.

5.2 Аналіз небезпечних і шкідливих факторів при роботі з персональним комп’ютером

В сучасних умовах, комп’ютерна техніка стала невід’ємною частиною життя людини. Вона використовується вдома, в робочих кабінетах, в торгівлі, на виробничих підприємствах і навіть, в побутових приладах. Іншими словами, комп’ютери глибоко інтегрувалися у повсякденні процеси, а масштаби їх застосування продовжують зростати.

В офісному середовищі, комп’ютери виконують функцію ключових інструментів для обробки інформації, що суттєво змінило характер роботи службовців. Разом з поширенням цифрових технологій, зросли й вимоги до організації робочого простору та забезпечення належного рівня охорони праці.

Ігнорування правил безпеки під час роботи з комп’ютером може призвести до помітного дискомфорту вже після кількох годин діяльності. Працівники часто стикаються з головним болем, печінням або різзю в очах, підвищеною втомлюваністю та дратівливістю. В деяких користувачів, можуть виникати порушення сну, погіршення зору, біль в руках, шиї чи попереку та інші негативні зміни стану здоров’я.

З позиції впливу небезпечних та шкідливих факторів під час роботи за персональним комп’ютером можна виділити такі основні групи:

– дія електромагнітного та електростатичного випромінювання, що виникає під час роботи комп’ютерного обладнання;

– негативний вплив недостатнього або неправильно організованого освітлення робочої зони;

– ризик ураження електричним струмом у разі несправності техніки або порушення правил експлуатації;

– вплив підвищеного рівня шуму, який може створювати периферійне чи офісне обладнання;

– негативна дія несприятливих параметрів мікроклімату, зокрема температури, вологості та повітряних потоків;

– наслідки неправильного розміщення комп’ютерного обладнання та нераціональної організації робочого місця, що призводять до фізичного перевантаження та дискомфорту.

Тривале перебування людини під дією комбінованих несприятливих факторів, під час роботи з комп’ютером, здатне спричинити розвиток професійних захворювань. Тому, необхідно детально розглянути, як саме кожен із зазначених факторів впливає на стан здоров’я користувача ПК.

Під час роботи персонального комп’ютера утворюються два основні види випромінювань – електростатичне та електромагнітне. Електростатичний заряд на поверхні монітора сприяє накопиченню пилу, що підсилює забруднення повітряної зони перед екраном. Електромагнітні поля формуються внаслідок роботи магнітних елементів всередині пристрою. За даними медичних досліджень, пил, що електризується, може провокувати запальні реакції шкіри, тоді як електромагнітні випромінювання здатні негативно впливати на біологічні процеси організму, порушувати сон, знижувати артеріальний тиск, впливати на ритм серця та навіть змінювати склад крові.

Висока напруженість зорової роботи, що виникає через неякісне або нерівномірне освітлення, може стати причиною погіршення функцій зорового аналізатора та призвести до зниження гостроти зору, а в окремих випадках, навіть до його втрати. Рівень зорової втоми визначається інтенсивністю процесів, пов’язаних із сприйняттям візуальної інформації. Під час роботи за комп’ютером, саме очі зазнають найбільшого навантаження, тому правильна організація освітлення робочого місця є критично важливою.

Виробниче освітлення залежно від джерела світла може бути природним, штучним або комбінованим. Під час роботи з ПК в денний час за сприятливих погодних умов перевага надається природному освітленню, що надходить через віконні прорізи. У випадку недостатнього природного освітлення або ввечері застосовують штучне, його забезпечують газорозрядні лампи. Важливу роль у

цьому випадку відіграють правильне розташування світильників, їх тип, а також, кількість, оскільки саме ці параметри визначають рівень комфортності та безпеки для зору.

Коректно організована освітленість робочої зони позитивно впливає на працездатність користувача комп'ютера та мінімізує негативний вплив на органи зору, забезпечуючи комфортні умови для тривалої роботи.

Оптимальний рівень освітлення для робочого місця користувача ПК має становити 400-750 лк, що відповідає нормам безпечної та комфортної зорової діяльності.

Функціонування комп'ютерної техніки та периферійного обладнання забезпечується подачею електричного струму, тому небезпека ураження працівника завжди існує. Для запобігання травмуванню необхідно суворо дотримуватися правил електробезпеки та експлуатації технічних засобів.

Особливість небезпеки електричного струму полягає в тому, що людина не здатна визначити наявність напруги без спеціальних приладів, а сама дія струму відбувається миттєво, небезпечний вплив стає очевидним лише після ураження. Електротравми можуть спричиняти як локальні пошкодження тканин, так і загальні порушення роботи організму. До відчутної дії змінного струму промислової частоти людина чутлива вже при силі 0,6-1,5 мА, а постійного, 5-7 мА. Такі значення не становлять критичної небезпеки, адже зазвичай дозволяють самостійно розірвати контакт зі струмоведучими елементами. Для змінного струму промислової частоти, небезпечним є так званий «невідпускаючий» струм силою 6-20 мА. Постійний струм не спричиняє такого ефекту, проте викликає виражений больовий вплив при силі 15-80 мА і вище.

Під шумом розуміють будь-який небажаний звук, що заважає людині та ускладнює сприйняття важливої інформації. Це хаотичне поєднання звуків різної частоти та інтенсивності, яке може погіршувати здатність розрізняти колірні сигнали, зменшувати швидкість зорового сприйняття, впливати на гостроту зору та порушувати процес обробки візуальної інформації. Відомо, що шум здатен знизити

продуктивність праці на 5-12%. При тривалому впливі шуму з рівнем звукового тиску близько 90 дБ падіння продуктивності може досягати 30-60%.

Робота комп'ютера може супроводжуватися появою шумів, які виникають під час функціонування рухомих елементів усередині системного блока або підключених периферійних пристроїв. До таких джерел шуму належать вентилятори охолодження, дискові накопичувачі, принтери та інше обладнання. Інтенсивний шум, залежно від його рівня, здатен підвищувати нервову напругу працівника та зменшувати його працездатність.

Мікрокліматичні умови приміщення, температура повітря, його вологість та швидкість руху, істотно впливають як на самопочуття працівника, так і на стабільність функціонування комп'ютерного обладнання.

Перегріте або надто вологе повітря в приміщенні негативно позначається на стані організму, спричиняючи перегрів тіла, втому та зниження працездатності. Важливо враховувати, що увімкнена комп'ютерна техніка додатково сприяє підвищенню температури в кімнаті.

Оптимальними параметрами мікроклімату для комфортної роботи вважається температура на рівні 23 (±2)°C в теплий період року та 18 (±2)°C в холодний, за відносної вологості 55 (±5)%.

Робоче місце – це частина простору, де користувач персонального комп'ютера виконує більшість своїх професійних завдань. Раціональна організація місця повинна забезпечувати зручність, економію часу та фізичних зусиль, відповідати ергономічним вимогам та сприяти дотриманню правил охорони праці.

Під час планування робочого місця необхідно враховувати зони досяжності для монітора, клавіатури та маніпулятора «миша». Ці зони визначаються антропометричними характеристиками людини. Робочий простір має бути організований так, щоб на користувача не впливали сторонні подразники, включаючи невдале забарвлення обладнання чи інтер'єру.

Відповідно до чинних нормативних документів, площа робочого місця користувача ПК повинна становити щонайменше 4,5 м². В приміщенні потрібно щоденно проводити вологе прибирання та забезпечувати регулярне провітрювання,

не менше одного разу на годину роботи. Обладнання, що створює підвищений рівень шуму, таке як принтери, сканери чи сервери, повинно розміщуватися за межами робочих зон співробітників, якщо рівень шуму перевищує встановлені норми.

Робочі столи необхідно розташовувати таким чином, щоб монітори були повернуті боковою поверхнею до вікон, а природне освітлення надходило переважно зліва, що зменшує засліплення та підвищує комфорт під час роботи.

Під час організації робочих місць слід дотримуватися наступних параметрів: відстань між столами має становити не менше 2 метрів, а між бічними сторонами моніторів, не менше 1,2 метра. Працівникам, які виконують творчі завдання або потребують високої концентрації уваги, бажано забезпечити робочі місця, відокремлені перегородками висотою не менше 1,5 метра. Конструкція робочого столу повинна дозволяти зручно розмістити все необхідне обладнання. Оптимальна висота поверхні столу складає 725 мм, його ширина повинна бути в межах 800-1400 мм, а глибина, 800-1000 мм. Простір для ніг має бути не менше 600 мм заввишки, 500 мм завширшки, 450 мм в глибині на рівні колін і не менше 650 мм, на рівні витягнутих ніг.

Робочий стілець або крісло повинні забезпечувати зручне та ергономічне положення тіла працівника, дозволяючи змінювати позу для зменшення статичної напруги м’язів плечового поясу та спини. Сидіння має бути підйомно-поворотним, з регулюванням висоти, кута нахилу сидіння та спинки, а також, відстані спинки від переднього краю сидіння. Кожен параметр повинен регулюватися окремо, легко фіксуватися та забезпечувати надійну підтримку.

Клавіатуру необхідно встановлювати на поверхні столу на відстані 100-300 мм від краю, зверненого до працівника, або на спеціальній висувній чи окремій панелі, що забезпечує зручне розташування рук.

Екран монітора повинен розташовуватися від очей користувача на відстані 600-700 мм, але не ближче ніж 500 мм, щоб зменшити навантаження на зір та забезпечити комфортну роботу.

5.3 Розрахунок системи загального освітлення приміщення з персональним комп'ютером

Раціонально організоване освітлення виробничих і офісних приміщень, є одним з ключових чинників, що визначають результативність трудової діяльності. Висока якість освітлення є необхідною умовою виконання більшості робочих операцій.

До систем освітлення висувається низка вимог, серед яких:

– відповідність рівня освітленості характеру зорових завдань;

– рівномірний розподіл яскравості в зоні робочих поверхонь та довколишньому просторі;

– стабільність освітленості протягом усього періоду роботи;

– оптимальна спрямованість світлового потоку, що формується приладами освітлення;

– довговічність та економічність, а також, пожежна й електрична безпека, зручність та простота експлуатації приладів.

В приміщеннях, де використовуються ПК, рекомендується застосовувати змішану систему освітлення, що поєднує природне світло та штучні джерела. Денне освітлення забезпечується через вікна збоку, тоді як штучне застосовується за умови недостатнього природного освітлення. В даному приміщенні передбачене загальне штучне освітлення, розрахунок якого виконують за методом світлового потоку з урахуванням відбитого світла від стін і стелі.

Необхідно провести розрахунок загального освітлення приміщення. Геометричні параметри: довжина $A = 9м$, ширина $B = 6м$, висота $H = 3м$. Висота робочої поверхні становить $h_p = 1м$. Передбачається використання світильників типу УПМ-15. Нормована мінімальна освітленість для ламп розжарювання становить $E_{\min} = 100лк$. Коефіцієнти відбиття, для стелі $S_n = 50\%$, для стін $S_c = 30\%$, для робочої поверхні $S_p = 10\%$. Напруга живлення – 220 В.

Визначимо параметри підвісу світильників. Відстань від стелі до робочої поверхні:

$$H_0 = H - h_p = 3 - 1 = 2м. \quad (5.1)$$

Відстань між стелею та світильником: $h_c = 0{,}2 \cdot H_0 = 0{,}2 \cdot 2 = 0{,}4м$

Висота підвісу світильника над робочою площиною:

$$h = H_0 - h_c = 2 - 0{,}4м = 1{,}6м. \quad (5.2)$$

Висота підвісу над підлогою:

$$H_n = h + h_p = 1{,}6 + 1 = 2{,}6м \quad (5.3)$$

Для досягнення максимально рівномірного освітлення приймається співвідношення $\frac{L}{h} = 1{,}5$. Тоді відстань між центрами світильників:

$$L = 1{,}5 \cdot h = 1{,}5 \cdot 1{,}6 = 2{,}4м. \quad (5.4)$$

Кількість світильників:

$$N = \frac{S}{L^2} = \frac{9 \cdot 6}{2{,}4^2} \approx 9{,}375. \quad (5.5)$$

Округлюючи, отримуємо 10 світильників, розташованих в двох рядах по п’ять.

Індекс приміщення обчислюється за формулою:

$$i = \frac{AB}{h(A + B)} = \frac{9 \cdot 6}{1{,}6 \cdot (9 + 6)} = 2{,}25. \quad (5.6)$$

Згідно з таблицею коефіцієнтів використання світлового потоку для світильників УПМ-15 за параметрами $i = 2{,}25, S_n = 50\%, S_c = 30\%, S_p = 10\%$ приймається $\eta = 0{,}53$.

Світловий потік однієї лампи визначаємо за формулою:

$$\Ф = \frac{E_{\min} \cdot S \cdot K_з \cdot Z}{N \cdot \eta} = \frac{100 \cdot 54 \cdot 1.5 \cdot 1.1}{10 \cdot 0.53} = 1681 \text{ лм} \qquad (5.7)$$

де $K_з = 1{,}5$ – коефіцієнт запасу для світильників з люмінесцентними лампами за умови їх очищення щонайменше двічі на рік, $Z = 1{,}1$ – коефіцієнт нерівномірності освітлення.

За отриманим значенням обираємо лампу типу Г215-225-150 потужністю 150 Вт, яка має світловий потік 2090 лм – найближчий до розрахункового.

Фактична освітленість робочої зони розраховується за формулою:

$$E_ф = \frac{Eл_{min}}{Ф_р \frac{100 \cdot 2090}{1681}} \text{ЛК.} \qquad (5.8)$$

Загальна встановлена потужність освітлення:

$$P_{заг} = P_л \cdot N = 150 \cdot 10 = 1500Вт = 1{,}5кВт. \qquad (5.9)$$

Для забезпечення нормативного освітлення в приміщенні слід встановити два ряди по п’ять світильників типу Г215-225-150, а загальна споживана потужність системи освітлення становитиме 1,5 кВт.

5.4 Розробка заходів щодо зменшення впливу шкідливих та небезпечних факторів при роботі з персональним комп’ютером

Зменшення ризику ураження електричним струмом передбачає передусім правильне розміщення обладнання й кабельних ліній. Всі інші заходи електробезпеки фактично збігаються з загальноприйнятими вимогами пожежної та електричної безпеки в робочих приміщеннях.

Як профілактику пожеж і електротравм рекомендується використовувати приховані електромережі, якісні розетки з пожежостійких матеріалів, а також силові кабелі, розраховані на навантаження, що в 3-5 разів перевищує фактичне. Увімкнення та вимкнення обладнання необхідно здійснювати лише штатними вимикачами. Важливим є регулярне очищення внутрішніх компонентів комп'ютерів та іншого обладнання від пилу, а самі ПК слід розташовувати на вогнетривких столах. Щоб уникнути іскріння, небажано часто вставляти чи виймати штепсельні вилки з розеток.

Монітор має бути встановлений так, щоб його поверхня була перпендикулярною до напряму погляду. Якщо екран нахилений, користувач може мимоволі сутулитися. Відстань між очима та дисплеєм повинна бути трохи більшою, ніж типова дистанція під час читання книги. Для старих моделей моніторів бажано використовувати захисний екран; за його відсутності слід працювати на відстані витягнутої руки. Для зменшення перенапруження зору варто формувати різнорідне поле зору, наприклад, розмістити на стінах спокійні за тоном картини чи пейзажні зображення.

Особливе значення має форма спинки крісла, вона повинна повторювати природну кривизну хребта. Висота сидіння має бути такою, щоб виключити тиск на стегна чи куприк. Бажано, щоб крісло було оснащене підлокітниками. Його слід відрегулювати так, щоб до клавіатури не доводилося тягнутися. Під час роботи необхідно періодично змінювати позу, рухатися та робити невеликі перерви, щоб уникнути статичного перенавантаження.

За інтенсивної роботи за комп'ютером необхідно щогодини робити перерву приблизно на 15 хвилин та переключатися на інший вид діяльності. Декілька разів на годину рекомендується виконувати легкі вправи для розслаблення м'язів і зниження загального напруження.

Техніка безпеки під час роботи на комп’ютері

Загальні положення

1. Під час роботи з персональними комп’ютерами працівник повинен дотримуватися вимог як загальних інструкцій з охорони праці, так і спеціальних правил, передбачених цією інструкцією.

2. До самостійної роботи за комп’ютером допускаються лише ті особи, які пройшли медичний огляд, необхідне професійне навчання, вступний інструктаж з охорони праці та первинний інструктаж на робочому місці. В подальшому, працівники повинні проходити повторний інструктаж один раз на шість місяців.

3. Під час роботи з комп’ютерами, що містять відеодисплейні термінали, можливий вплив шкідливих та небезпечних виробничих факторів.

а) фізичні фактори:

– підвищений рівень шуму, що створюється вентиляторами та іншими елементами ПК;

– висока напруга в електромережі, контакт з якою може призвести до електротравми;

– накопичення статичної електрики;

– електромагнітні випромінювання різного діапазону;

– надмірна напруженість електричного поля;

– пряма та відбита блискучість від поверхні екрана;

– нерівномірна яскравість об’єктів у полі зору.

б) психофізіологічні фактори:

– статичні та динамічні навантаження на опорно-руховий апарат;

– нервово-психічна напруга, включаючи інтелектуальне перенавантаження, перевтому зорового аналізатора, монотонність роботи та емоційні перевантаження.

4. Основними елементами робочого місця користувача ПК є: відеодисплейний термінал, системний блок, клавіатура, принтер, зовнішні носії інформації, сканер, маніпулятор «миша», блок безперебійного живлення, стіл і крісло. До допоміжного обладнання належать шафи, полиці, сейфи тощо.

Робочі місця з ВДТ повинні розташовуватися на відстані не меншій ніж 1,5 м від стіни з вікнами, не менше 1 м від глухих стін, і не ближче ніж 1,5 м одне від одного. Бажано, щоб природне освітлення падало збоку, переважно з лівого боку.

5. Робочі місця з ВДТ повинні бути організовані так, щоб уникати потрапляння прямого світла в очі користувача. Світильники слід розміщувати з обох боків екрану паралельно напрямку зору. Для зменшення відблисків від екрана чи клавіатури необхідно використовувати антивідблискові решітки, спеціальні захисні фільтри або козирки. На вікнах слід встановлювати регульовані жалюзі. Сітчасті фільтри з металу або нейлону небажані, оскільки вони спотворюють зображення через ефект інтерференції. Найкращу якість забезпечують скляні поляризаційні фільтри, які практично повністю усувають відблиски.

6. Під час роботи з текстовою інформацією найбільш комфортним для зору є відображення темних символів на світлому (білому) фоні.

7. Розміщувати ВДТ слід так, щоб екран знаходився в центрі поля зору на відстані 400-700 мм від очей. Бажано забезпечити однакову дистанцію від очей працівника до монітора, клавіатури та пюпітра для документів.

8. Комп’ютерні пристрої створюють електромагнітні поля різних частот, інтенсивність яких зменшується пропорційно квадрату відстані. Оптимальною відстанню до екрана ВДТ вважається близько 50 см.

9. Найбільш ергономічною позою для роботи за ПК є така, коли ступні стоять рівно на підлозі або на підставці, стегна розташовані горизонтально, передпліччя вертикально, кут в ліктьових суглобах становить 70-90°, зап’ястя зігнуте не більше ніж на 20°, а нахил голови, в межах 15-20°.

10. Для зниження статичної електрики в приміщенні необхідно підтримувати підвищену вологість за допомогою зволожувачів повітря. Не рекомендується використовувати одяг з синтетичних тканин, оскільки він легко накопичує електричні заряди.

Вимоги безпеки перед початком роботи:

1. Переконатися, що все обладнання на робочому столі встановлене стійко та надійно. ВДТ не повинен знаходитися на краю столу. Екран слід

повернути так, щоб користувачу було зручно дивитися на нього під прямим кутом і трохи згори вниз; сам дисплей має бути нахилений під невеликим кутом (нижній край ближче до оператора).

2. Перевірити технічний стан обладнання, цілісність проводки, кабелів живлення, з'єднувальних шнурів, вилок і розеток, а також, наявність та правильність заземлення захисного екрана.

3. Переконатися, що освітлення на робочому місці відповідає нормам та не створює відблисків чи надмірної яскравості.

4. Відрегулювати висоту стільця (крісла) та кут нахилу його спинки, забезпечивши максимально зручну позу для роботи.

5. Під'єднати до системного блоку необхідні периферійні пристрої, принтер, сканер тощо. Всі кабелі можна підключати або від'єднувати лише тоді, коли комп'ютер повністю вимкнений від електромережі.

6. Вмикати обладнання у відповідній послідовності, спочатку стабілізатор або блок безперебійного живлення, далі ВДТ, потім системний блок і лише після цього принтер (за потреби друку).

7. Відрегулювати яскравість, фокусування та контрастність екрана ВДТ. Надмірна яскравість небажана, оскільки перевантажує очі та скорочує термін служби електронно-променевої трубки.

Рекомендовані параметри:

– яскравість не менше 100 кд/м²;

– співвідношення яскравості екрана до інших поверхонь, не більше 3:1;

– мінімальний розмір світної точки до 0,4 мм для монохромних дисплеїв та до 0,6 мм для кольорових;

– контрастність зображення не нижче 0,8.

8. В разі виявлення несправності комп'ютерного обладнання необхідно негайно повідомити керівника.

Вимоги безпеки після завершення роботи:

1. Завершити роботу з файлами, зберегти всі внесені зміни та коректно вийти з операційної системи.

2. Вимкнути принтер та інші периферійні пристрої, після цього ВДТ і системний блок. Далі відключити стабілізатор (або блок безперебійного живлення). Витягнути штепсельні вилки з розеток. Накрити клавіатуру захисною кришкою, щоб запобігти потраплянню пилу.

3. Навести порядок на робочому місці.

4. Вимкнути освітлювальні прилади та інше електрообладнання.

Вимоги безпеки у разі аварійних ситуацій:

1. У випадку раптового припинення подачі електроенергії слід вимкнути всі пристрої ПК в такій послідовності: периферійні пристрої, ВДТ, системний блок, стабілізатор (або блок безперебійного живлення). Після цього потрібно від’єднати штепсельні вилки від електромережі.

2. Якщо з’явилися ознаки займання (дим, запах горілого), потрібно негайно вимкнути апаратуру. Далі встановити місце загоряння та застосувати всі можливі заходи для його ліквідації, попередивши про подію керівництво.

3. В разі виникнення пожежі слід негайно повідомити пожежну службу та керівництво, здійснити евакуацію людей з приміщення й розпочати гасіння полум’я наявними первинними засобами пожежогасіння.

5.5 Висновки

У даному розділі проведено комплексний аналіз умов праці інженера-програміста. Було визначено основні небезпечні та шкідливі фактори, серед яких ключовими є зорове навантаження, гіподинамія та електромагнітне випромінювання. Проведений розрахунок системи загального освітлення приміщення площею 54 м² дозволив обрати оптимальну кількість світильників 10 одиниць типу УПМ-15, що забезпечують нормовану освітленість 124,3 лк. Впровадження запропонованих заходів (раціональна організація робочого місця, режим праці та відпочинку) дозволяє мінімізувати ризики професійних захворювань та підтримувати високу продуктивність праці під час розробки ПЗ.

6 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

6.1 Вступ

Під охороною довкілля розуміють сукупність міжнародних, державних та регіональних нормативних документів, інструкцій та стандартів, які встановлюють загальнообов'язкові правові вимоги до всіх суб'єктів, що можуть спричиняти забруднення, та мотивують їх до виконання даних вимог. До охорони довкілля належать практичні природоохоронні заходи, спрямовані на реалізацію зазначених норм.

Система охорони навколишнього природного середовища включає три ключові компоненти:

– правову охорону, яка формує екологічні принципи у вигляді обов'язкових для виконання законодавчих норм;

– матеріальне стимулювання природоохоронної діяльності, спрямоване на те, щоб зробити екологічно відповідні дії економічно вигідними для підприємств;

– інженерний напрям, що розробляє екологічні, ресурсозберігаючі технології та технічні рішення.

Охороні підлягають такі об'єкти довкілля, як природні екосистеми, озоновий шар атмосфери, земельні ресурси та надра, підземні й поверхневі води, атмосферне повітря, лісові масиви й інша рослинність, тваринний світ, мікроорганізми, генетичні ресурси та природні ландшафти.

Особливому державному захисту підлягають природні заповідники, заказники, національні природні парки, пам'ятки природи, а також, рідкісні та зникаючі види рослин і тварин, і території їх природного існування.

До основних принципів охорони навколишнього середовища належать пріоритет створення сприятливих екологічних умов для життя, праці та відпочинку населення, науково обґрунтоване поєднання екологічних та економічних інтересів

суспільства, врахування законів природи, а також, можливостей самовідновлення й самоочищення природних ресурсів.

6.2 Види забруднення навколишнього природного середовища

Різні форми втручання людини в природні процеси біосфери можуть бути класифіковані за видами забруднень, під якими розуміють будь-які небажані антропогенні зміни, що порушують стан екосистем.

– інгредієнтне забруднення – це надходження в навколишнє середовище речовин або сумішей, що кількісно чи якісно істотно відрізняються від природних компонентів біогеоценозів;

– параметричне забруднення виникає внаслідок зміни природних параметрів середовища (шуму, рівня освітленості, радіації тощо), що призводить до порушення екологічної рівноваги;

– біоценотичне забруднення проявляється у впливі на структуру та чисельність популяцій живих організмів, що змінює природні біологічні співтовариства;

– стаціально-деструкційне забруднення пов’язане зі змінами природних ландшафтів і екологічних систем, що виникають в процесі господарської діяльності людини, зокрема, через руйнування місць існування популяцій.

На сучасному етапі, основна увага приділяється зменшенню рівня матеріального та енергетичного забруднення довкілля, оскільки саме дані види негативного впливу становлять найбільшу загрозу для екосистем та здоров’я людини.

6.3 Природоохоронна діяльність підприємств

Природоохоронною діяльністю вважають будь-які дії, спрямовані на підтримання якості навколишнього середовища на такому рівні, який забезпечує

стабільність біосфери. До даної категорії належать як масштабні державні програми, так і локальні заходи підприємств, пов’язані з очищенням стічних вод та газових викидів, скороченням використання природних ресурсів та зменшенням негативного впливу на довкілля.

В природоохоронній сфері виділяють два основні напрями діяльності. Перший передбачає очищення шкідливих викидів. Подібний підхід сам по собі не завжди ефективний, оскільки не гарантує повне припинення надходження забруднювальних речовин в біосферу. Більше того, зменшення забруднення одного компонента довкілля може призвести до підвищення навантаження на інший.

Другий напрямок полягає в усуненні першопричин виникнення забруднення, що потребує запровадження маловідходних, а в перспективі, безвідходних технологій. Представлені технології забезпечують більш повне використання сировини та дозволяють утилізувати максимальну кількість речовин, небезпечних для екологічних систем.

Однак, в багатьох галузях промисловості, поки що не існує оптимальних техніко-економічних рішень, які дозволяли б кардинально скоротити обсяг відходів та забезпечити їх ефективну переробку. Тому сьогодні доводиться поєднувати обидва підходи, і очищення викидів, і боротьбу з причинами їх утворення.

6.4 Розробка заходів щодо зменшення забруднення

Одним з важливих заходів зі зменшення негативного впливу комп’ютерної техніки на довкілля, є дотримання вимог стандартів, які регламентують безпечність та ергономічність моніторів та ЕОМ. Існують дві основні групи нормативів та рекомендацій, а саме, стандарти безпеки та стандарти ергономіки.

До стандартів першої групи належать UL, CSA, DHHS, CE, а також, скандинавські SEMRO, DEMKO, NEMKO та FCC Class B. Серед стандартів другої групи найпоширенішими є MPR-II, TCO’92, TCO’95, ISO 9241-3, EPA Energy Star, TUV Ergonomie. Далі наведено короткі характеристики деяких з них.

FCC Class B - Стандарт, розроблений Федеральною комісією зі зв'язку Канади, встановлює вимоги щодо зменшення радіоперешкод в закритих приміщеннях. Техніка, сертифікована за FCC Class B, не повинна створювати завад для роботи телевізійних або радіоприймачів.

MPR-II - Стандарт MPR-II був прийнятий в 1990 році Шведським національним департаментом та затверджений Європейським економічним співтовариством. Він обмежує рівень випромінювання комп'ютерних моніторів і офісної техніки.

TCO'92 (TCO'95) - Рекомендації TCO, розроблені Шведською федерацією профспілок спільно з NUTEK, визначають вимоги до безпечної взаємодії офісного обладнання з довкіллям. Вони вимагають зниження рівнів електричних та магнітних полів до технічно досяжних значень. Для отримання сертифіката TCO'92 монітор повинен відповідати стандартам Low Radiation, мати режим автоматичного зниження енергоспоживання, а також, відповідати нормам пожежної та електричної безпеки.

TUV Ergonomie - Стандарт TUV Ergonomie (Німеччина) підтверджує відповідність монітора вимогам електробезпеки (EN 60950) та ергономіки (ZN 1/618), а також, відповідність шведському стандарту низького випромінювання MPR-II.

EPA Energy Star / VESA DPMS - За даним стандартом монітор повинен мати три режими енергозбереження, stand-by, suspend і off. В режимі stand-by зображення на екрані зникає, але апаратура продовжує працювати зі зниженим енергоспоживанням (до 80% від номіналу). В режимі suspend, як правило, вимикаються високовольтні модулі, і споживання електроенергії падає до ≈ 30 Вт. В режимі off монітор споживає не більше 8 Вт, працює лише мікропроцесор.

Після натискання клавіші клавіатури або руху миші монітор автоматично повертається у робочий режим.

Суттєвою екологічною проблемою сучасності є стрімке накопичення електронних відходів. Екологічна стратегія Європейського Союзу базується на

шести великих Програмах дій в сфері охорони довкілля, значна частина яких присвячена врегулюванню поводження з відходами різного походження.

Утилізація офісної техніки, включаючи персональні комп'ютери, є обов'язковою вимогою для всіх підприємств. Неправильне поводження з електронними відходами може спричинити локальні техногенні аварії. Компоненти комп'ютерів містять ртуть, свинець, миш'як та інші токсичні елементи, що під впливом зовнішніх факторів перетворюються на небезпечні для довкілля сполуки.

В системних блоках та моніторах присутні дорогоцінні та кольорові метали, тому утилізацію комп'ютерної техніки мають здійснювати лише спеціалізовані підприємства. Завдяки цьому, цінні матеріали (чорні, кольорові та дорогоцінні метали) повертаються у виробничий цикл, а інші компоненти проходять відповідну переробку. Підприємства, що не мають відповідної ліцензії, зобов'язані звертатися до професійних організацій.

Для списання та подальшої утилізації комп'ютерної техніки юридичні особи повинні звернутися до спеціалізованого підприємства, яке здійснює експертизу стану обладнання. Вона видає висновок про моральне старіння техніки та неможливість її ремонту чи подальшої експлуатації. Після цього, підприємство може укладати договір на утилізацію відходів.

Ліцензію на переробку відходів з вмістом дорогоцінних металів видає Міністерство фінансів, на утилізацію кольорових та чорних металів – Міністерство промислової політики, а на утилізацію полімерів та токсичних речовин – Міністерство екології та природних ресурсів. Процедура є складною, але вона дозволяє підприємствам дотримуватися вимог законодавства. Хоча утилізація є витратною, уникнути даних витрат неможливо.

Вартість утилізації одного ПК становить: в Німеччині 15-30 євро, в Японії близько $40, в Україні приблизно 100 грн.

Враховуючи наведені дані, для підприємства, що впроваджує розроблене ПЗ, доцільно заздалегідь включити витрати на утилізацію до бюджету на підтримку IT-інфраструктури. Це не лише забезпечує відповідність екологічним нормам, а й дозволяє уникнути штрафних санкцій за неправильне поводження з небезпечними

відходами. Таким чином, відповідальне ставлення до життєвого циклу обладнання стає невід’ємною частиною корпоративної стратегії сталого розвитку компанії.

6.5 Висновки

Проведений аналіз екологічних аспектів експлуатації обчислювальної техніки в межах розробки програмного забезпечення дозволяє констатувати, що дотримання принципів охорони довкілля вимагає комплексного підходу. Розгляд сучасних стандартів ергономіки та безпеки підтвердив ключову роль сертифікації Energy Star та TCO у зниженні антропогенного навантаження через оптимізацію енергоспоживання робочих станцій. Водночас вивчення процедур утилізації електронних компонентів показало, що професійна переробка техніки є критично важливою для запобігання забрудненню середовища токсичними речовинами та важкими металами.

У результаті дослідження було обґрунтовано, що створення сучасних програмних засобів має супроводжуватися відповідальним ставленням до вибору апаратної бази та плануванням процесів списання обладнання. Такий підхід забезпечує не лише безпечні умови праці для розробників, а й мінімізує сумарний негативний вплив на екосистеми. Отже, розроблений проєкт повністю відповідає чинним екологічним нормам, а запропоновані заходи з охорони навколишнього середовища є економічно доцільними та технічно обґрунтованими для впровадження у практику ІТ-підприємства.

ВИСНОВКИ

В представленій роботі повністю реалізовано поставлену мету щодо підвищення точності оцінювання розміру програмних застосунків на TypeScript. Досягнення мети підтверджується послідовним виконанням усіх наукових та практичних завдань дослідження.

Проаналізовано існуючі методи та моделі оцінювання розміру програмного забезпечення, виявлено їх переваги та недоліки. Встановлено, що класичні моделі зокрема COCOMO та спрощені лінійні підходи мають обмежену точність при роботі з сучасними TypeScript-проєктами. Виявлено, що головним недоліком існуючих підходів є ігнорування структурної складності коду та високої зв'язності компонентів.

Виконано збір та аналіз статичних метрик коду на вибірці 53 відкритих проєктів на мові програмування TypeScript. Статистичний аналіз виявив, що розподіл даних відхиляється від нормального закону, що обумовило необхідність застосування логарифмічної нормалізації та очищення від викидів за відстанню Махаланобіса.

Розроблено удосконалену регресійну модель для прогнозування розміру застосунків. Побудовано нелінійну регресійну модель, яка, на відміну від попередніх розробок, додатково враховує метрику TR.

Проведено експериментальну перевірку розробленої моделі та виконано порівняльний аналіз її з лінійною регресійною моделлю. Доведено, що запропонована нелінійна модель забезпечує коефіцієнт детермінації $R^2 = 0{,}9$, що значно перевищує показник базової лінійної моделі $R^2 = 0{,}65$. Також досягнуто звуження довірчих інтервалів на 51,1%, що свідчить про вищу надійність прогнозу.

Проведено експериментальну перевірку розробленої моделі та виконано порівняльний аналіз її точності з існуючою моделлю. Порівняння з однофакторною моделлю представленою в роботі Чебакова Д.В. на тестовій вибірці показало

суттєве зниження середньої відносної похибки MMRE до 0,372, що підтверджує ефективність введення додаткових факторів для оцінювання складних проєктів.

Розроблено програму для оцінювання розміру застосунків мовою TypeScript. Спроектовано та реалізовано веб-застосунок на базі фреймворку Django, який автоматизує процеси завантаження даних, побудови моделі та розрахунку прогнозів. Виконані економічні розрахунки показали, що термін окупності розробки становить близько 9 місяців. Також у роботі розроблено заходи з охорони праці та рекомендації щодо екологічної безпеки.

Практичне значення отриманих результатів полягає у створенні готового інструментарію, який дозволяє менеджерам проєктів отримувати обґрунтовані оцінки розміру застосукнів мовою TypeScript на ранніх етапах проектування системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] *Developer survey*. (2025). stackoverflow. https://survey.stackoverflow.co/2025/

[2] Tan, H. B. K., Zhao, Y ., & Zhang, H. (2006). Estimating LOC for information systems from their conceptual data models. У software engineering: *The 28th international conference (ICSE ‘06)* (с. 321–330). https://doi.org/10.1145/1134285.1134331

[3] Prykhodko, S., & Prykhodko, N. (2020). mathematical modeling of non-gaussian dependent random variables by nonlinear regression models based on the multivariate normalizing transformations. У *advances in intelligent systems and computing* (с. 166–174). Cham. https://doi.org/10.1007/978-3-030-58124-4_16

[4] Павлюк, К. А. (2022). Нелінійна регресійна модель для оцінювання розміру програмних застосунків на TYPEScript [Кваліфікаційна робота магістра, Національний університет кораблебудування імені адмірала Макарова]. https://eir.nuos.edu.ua/handle/123456789/8282

[5] Чебаков, Д. В. (2020). Удосконалення регресійної моделі для оцінювання розміру програмних застосунків на TypeScript та розробка програми для її реалізації [Кваліфікаційна робота магістра, Національний університет кораблебудування імені адмірала Макарова]. https://eir.nuos.edu.ua/handle/123456789/3707

[6] Prykhodko, N. V ., & Prykhodko, S. B. (2018b). The non-linear regression model to estimate the software size of open source Java-based systems. *Radio Electronics, Computer Science, Control*,(3), 158–168. https://doi.org/10.15588/1607-3274-2018-3-17

[7] Sommerville, I. (2015). *Software Engineering (10th ed.)*. Pearson.

[8] Microsoft. (2024). TypeScript Documentation. https://www.typescriptlang.org/docs/

[9] Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D., & Steece, B. (2000). *Software cost estimation with COCOMO II*. Prentice Hall PTR.

[10] Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, *20*(6), 476–493. https://doi.org/10.1109/32.295895

[11] GeeksforGeeks. (n.d.). Software engineering | Functional point (FP) analysis. https://www.geeksforgeeks.org/software-engineering/software-engineering-functional-point-fp-analysis/

[12] Метрики складності програмного забезпечення. (б.д.). StudFiles. https://studfile.net/preview/14533278/

[13] Refactoring.Guru. (б.д.). Паттерн Фасад. https://refactoring.guru/design-patterns/facade

[14] Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012*).* Introduction to linear regression analysis *(5th ed.).* Wiley.

[15] McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308–320.

[16] Albrecht, A. J. (1979). Measuring application development productivity. *Proceedings of the IBM Applications Development Joint Share/Guide Symposium*.

[17] Xu, A. (2020). System design interview – An insider's guide (2nd ed.). Independently published.

[18] Halvorsen, H.-P. (2020). *Exploring cross-platform development. Manning Publication*s. https://www.manning.com/books/exploring-cross-platform-development

[19] Purewal, S. (2014). *Learning web app development: Build quickly with proven JavaScript techniques*. O'Reilly Media. ISBN-10 1449370195; ISBN-13 9781449370190.

[20] Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The unified modeling language user guide (2nd ed.*). Addison-Wesley.

[21] Ecma International. (2025). *ECMAScript® 2025 language specification* (16th ed.). https://262.ecma-international.org/

[22] Waskom, M. (n.d.). *Example gallery*. Seaborn. https://seaborn.pydata.org/examples

[23] Vincent, W. S. (2022). *Django for professionals: Production-ready web sites with Python & Django (4.0).* William S. Vincent.

[24] Kawsar, M. (2021). *Design and implementation of web-based library management system. ResearchGate*. https://www.researchgate.net/figure/Django-REST-Framework_fig1_353479839

[25] Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley.

[26] Hasura. (n.d.). *ER modeling*. https://hasura.io/learn/database/microsoft-sql-server/er-modeling/

[27] Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing (3rd ed.).* John Wiley & Sons.

ДОДАТОК А
ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ЗАСТОСУНКІВ МОВОЮ TYPESCRIPT

Вступ

Назва програми – «Програмний засіб для оцінювання розміру застосунків мовою TypeScript». Дане програмне забезпечення призначене для автоматизації процесу прогнозування обсягу вихідного коду SLOC на основі архітектурних метрик, що дозволяє підвищити точність планування ресурсів при розробці програмних систем.

1. Підстави для розробки

Документом, на підставі якого ведеться розробка, є завдання на кваліфікаційну магістерську роботу на тему: «Удосконалення математичної моделі для оцінювання розміру застосунків мовою TypeScript за рахунок додаткової метрики та програма для її реалізації.

2. Призначення розробки

2.1. Функціональне призначення

Програмний продукт призначений для автоматизації складних статистичних розрахунків, мінімізації суб’єктивізму при оцінці обсягу ПЗ та скорочення часу на проведення аналізу.

2.2. Експлуатаційне призначення

Система використовується для оцінювання розміру застосунків мовою TypeScript шляхом розрахунку прогнозного значення SLOC на основі вдосконаленої нелінійної регресійної моделі.

3. Вимоги до програми

3.1. Вимоги до функціональних характеристик

3.1.1. Вимоги до складу функцій, що виконуються

Програма повинна забезпечувати можливість виконання таких функцій:

1) введення значень вхідних метрик проєкту (NOC, AAC, TR);

2) завантаження нових наборів емпіричних даних у форматі CSV;

3) виконання логарифмічної нормалізації даних та очищення вибірки від статистичних викидів;

4) можливість актуалізації (перенавчання) параметрів моделі при оновленні бази даних;

5) редагування та видалення наявних записів у базі даних.

3.1.2 Вимоги до організації вхідних та вихідних даних

Введення даних здійснюється через веб-інтерфейс. Вихідні дані для користувача: прогнозоване значення SLOC у графічному вигляді.

Вихідні дані для адміністратора: аналітичний звіт, що містить розраховані коефіцієнти моделі та показники її якості.

3.2 Вимоги до надійності

3.2.1. Верифікація вхідних даних

Програма має бути стійкою до введення некоректних параметрів. Система повинна здійснювати автоматичну перевірку вхідних метрик (NOC, AAC, TR) перед початком розрахунків. Зокрема, має бути реалізовано блокування обробки даних у разі введення логічно неможливих значень.

3.2.2. Обробка помилок та інформування користувача

При виявленні некоректно введених даних система не повинна припиняти роботу або видавати критичні помилки коду. Замість цього ПЗ має виводити зрозумілі текстові повідомлення, які вказують на конкретне джерело проблеми (наприклад, «Кількість класів не може бути меншою за одиницю»). Це дозволяє користувачу швидко зорієнтуватися та виправити помилку у відповідному полі форми.

3.2.3. Відмовостійкість застосунку

Програмний засіб має залишатися у стабільному робочому стані незалежно від дій користувача. Будь-які помилки введення чи спроби завантаження пошкоджених файлів CSV повинні оброблятися механізмами валідації. Після відображення повідомлення про помилку система має бути одразу готовою до повторного введення даних, або виконання інших функцій без необхідності перезавантаження сторінки чи перезапуску сервера.

3.3 Вимоги до складу і параметрів технічних засобів

Для роботи ПЗ необхідні технічні засоби з наступними мінімальними параметрами:

– Операційні системи: Windows 10/11, Linux, macOS.
– Оперативна пам’ять: не менше 1 Гб.
– Дисковий простір: не менше 500 Мб.
– Пристрої введення: стандартна клавіатура та маніпулятор «миша»

3.4 Умови експлуатації

Умови експлуатації програми повинні відповідати стандартним умовам роботи персонального комп’ютера в офісному приміщенні. Спеціальних вимог до середовища або додаткового технічного обслуговування програмного засобу не передбачено.

3.5 Вимоги до інформаційної та програмної сумісності

Програма повинна працювати в операційних системах Windows, Linux або macOS.

3.6 Вимоги до захисту інформації та програм

Програма повинна забезпечувати захист розроблених математичних моделей оцінювання, оскільки їх параметри та структура становлять комерційну таємницю розробника. Архітектура системи має виключати можливість прямого доступу кінцевого користувача до алгоритмічної та коефіцієнтів регресії з метою запобігання несанкціонованому копіюванню інтелектуальної власності.

3.7 Вимоги до маркування та упаковки

Вимоги до маркування та упаковки відсутні.

3.8 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання програми не висуваються.

4 Вимоги до програмної документації

До складу обов’язкової програмної документації повинні входити наступні документи:

– технічне завдання;

– текст програми (вихідний код основних модулів);

– опис програмного забезпечення;

– програма та методика випробувань;

– інструкція користувача.

5 Техніко-економічні показники

Економічна ефективність впровадження програми обґрунтована у розділі 4 роботи. За рахунок автоматизації аналітичних операцій та скорочення часу на проведення оцінки обсягу ПЗ, очікуваний термін окупності витрат на розробку становить 8 місяців.

6 Стадії та етапи розробки програмного забезпечення

Стадії та етапи розробки програми оцінювання розміру застосунків мовою TypeScript наведені в таблиці А.1

Таблиця А.1 – Стадії та етапи розробки

Стадії розробки	Етапи робіт	Контрольні точки
Технічне завдання	Обумовлення необхідності розробки	З 08.09.2025 до
	Розробка та затвердження технічного	З 13.09.2025 до
Ескізний проект	Розробка ескізного проєкту	З 21.09.2025 до
	Затвердження ескізного проєкту	З 16.10.2025 до
Технічний проект	Розробка технічного проєкту	З 21.10.2025 до
	Затвердження технічного проєкту	З 11.11.2025 до
Робочий проект	Розробка програми	З 15.11.2025 до
	Випробування програми	З 06.12.2025 до

7 Порядок контролю та приймання

Контроль за виконанням робіт здійснюється керівником магістерської роботи на кожній стадії розробки програмного забезпечення. Приймання готової програми проводиться за результатами успішного виконання всіх перевірок, передбачених «Програмою та методикою випробувань» Додаток Б, у встановлені графіком терміни. На підставі результатів випробувань виноситься рішення про відповідність програмного продукту вимогам даного технічного завдання та його готовність до експлуатації, що засвідчується підписами виконавця та керівника.

ДОДАТОК Б
ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1 Об’єкт випробувань

Об’єктом випробувань є програмний засіб для оцінювання розміру застосунків мовою TypeScript, реалізований як вебзастосунок на базі фреймворку Django.

2 Мета випробувань

Метою випробувань є перевірка працездатності функцій розрахунку прогнозу розміру ПЗ, підтвердження коректності обробки вхідних даних та верифікація адміністративних функцій щодо актуалізації математичної моделі.

3 Вимоги до програмної документації

Для проведення випробувань програмне забезпечення повинно бути представлене у складі:

– технічне завдання (Додаток А);

– текст програми (Додаток Г);

– опис програмного забезпечення (Додаток Д);

– інструкція користувача (Додаток В).

4 Засоби та умови проведення випробувань

Випробування проводилися на робочій станції під управлінням операційної системи Linux (Ubuntu 25.10). Для розгортання ПЗ використовувалася платформа контейнеризації Docker (v 24.0) та інструмент Docker Compose. Використання контейнерів забезпечило автоматичне налаштування всіх необхідних залежностей Python 3.10+, Pandas, SciPy. Перевірка інтерфейсу користувача та відображення аналітичних звітів здійснювалася за допомогою веббраузера Google Chrome.

5 Процедура та зміст випробувань

5.1 Перевірка розгортання системи

Проводиться запуск контейнерів командою docker-compose up. Випробування вважається успішним, якщо сервер Django стартує без помилок, база даних ініціалізується, а веб-інтерфейс стає доступним за локальною адресою у браузері Google Chrome.

5.2 Тестування обчислювальних функцій

У форми введення вносяться валідні архітектурні метрики (NOC, AAC, TR). Результат розрахунку SLOC порівнюється з еталонними значеннями, отриманими у другому розділі роботи. Успішним результатом є збіг прогнозного значення з розрахунковим при заданому рівні точності.

5.3 Верифікація обробки помилок та валідації

Проводиться імітація некоректних дій користувача: введення від’ємних значень або символьних даних у поля метрик. Система повинна ідентифікувати помилку, заблокувати виконання розрахунку та вивести відповідне попередження в інтерфейсі, залишаючись при цьому в стабільному робочому стані.

5.4 Випробування функцій адміністратора

У модулі адміністрування виконується завантаження статистичної вибірки у форматі CSV. Випробування успішне, якщо дані коректно імпортуються до бази даних, а система формує аналітичний звіт із актуальними показниками якості моделі.

6 Результати випробувань

Програма вважається такою, що пройшла випробування, якщо всі етапи перевірки завершилися згідно з очікуваними результатами. На основі проведених тестів робиться висновок про повну відповідність ПЗ вимогам технічного завдання та його готовність до експлуатації.

ДОДАТОК В
ІНСТРУКЦІЯ КОРИСТУВАЧА

1 Призначення програми

Програма призначена для автоматизованого розрахунку прогнозованого обсягу вихідного коду (SLOC) застосунків мовою TypeScript на основі архітектурних метрик.

2 Умови виконання програми

Для роботи із застосунком необхідно мати встановлене середовище Docker та веббраузер (Google Chrome). Програма є кросплатформною та запускається на ОС Linux, Windows або macOS.

3 Запуск програми

Відкрийте термінал у кореневій папці проєкту.

Виконайте команду: docker-compose up.

Після успішного старту відкрийте веббраузер та перейдіть за адресою: http://localhost:8000/.

4 Порядок роботи з інтерфейсом

Після завантаження головної сторінки ви побачите форму для введення архітектурних метрик проєкту. Необхідно заповнити поля NOC, AAC та TR. Загальний вигляд форми введення представлено на рисунку В.1.

Для отримання результатів необхідно натиснути кнопку «Передбачити SLOC».

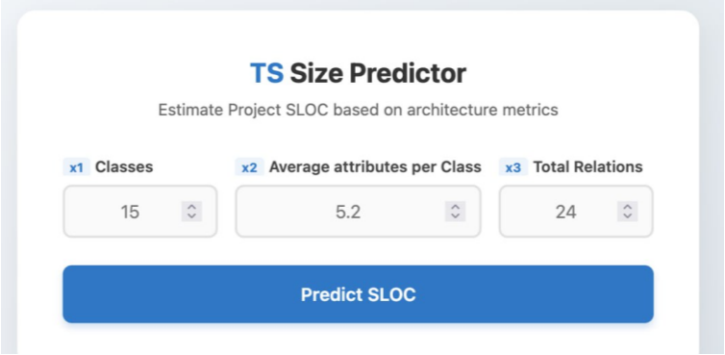


Рисунок В.1 – Інтерфейс введення вхідних метрик проєкту

5 Отримання результатів оцінювання

Після опрацювання запиту система виведе на екран прогнозні значення обсягу коду. Результати розраховуються паралельно за двома моделями: лінійною (базовою) та вдосконаленою нелінійною (основною). Приклад отриманих результатів наведено на рисунку В.2.

У разі введення некоректних даних (наприклад, від’ємних чисел), система заблокує розрахунок та відобразить повідомлення про помилку з вказівкою на конкретне поле.

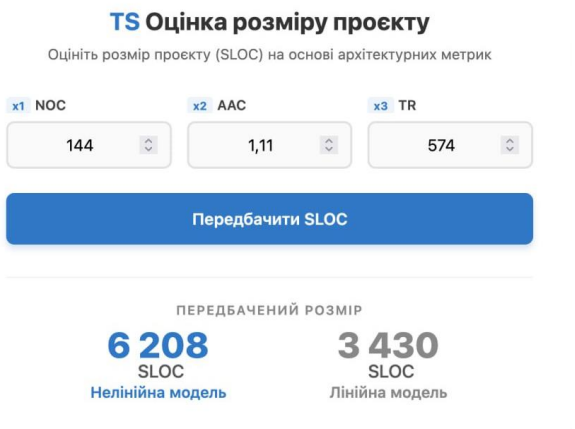


Рисунок В.2 – Відображення результатів розрахунку за двома моделями

ДОДАТОК Г

Текст програми

Текст модуля models.py

```
from django.db import models

class ConfigSystem(models.Model):
    curr_linear_model = models.ForeignKey('LinearModel', on_delete=models.SET_NULL,
null=True, blank=True, related_name='config_systems')
    curr_nonlinear_model = models.ForeignKey('NonLinearModel',
on_delete=models.SET_NULL, null=True, blank=True, related_name='config_systems')

class Dataset(models.Model):
    name = models.CharField(max_length=255, unique=True)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class DataPoint(models.Model):
    dataset = models.ForeignKey(Dataset, on_delete=models.CASCADE,
related_name='points')

    x1_value = models.PositiveIntegerField()
    x2_value = models.FloatField()
    x3_value = models.PositiveIntegerField()

    y_value = models.PositiveIntegerField()
    is_valid = models.BooleanField(default=True)

    def __str__(self):
        return f"DataPoint(x1={self.x1_value}, x2={self.x2_value},
x3={self.x3_value}, y={self.y_value})"

class LinearModel(models.Model):
    b0 = models.FloatField()
    b1 = models.FloatField()
    b2 = models.FloatField()
    b3 = models.FloatField()

    r2 = models.FloatField(default=0.0)
    mmre = models.FloatField(default=0.0)
    pred_25 = models.FloatField(default=0.0)

    dataset = models.ForeignKey(Dataset, on_delete=models.CASCADE,
related_name='linear_models', null=True, blank=True)

    def __str__(self):
```

```
        return f"LinearModel(b0={self.b0}, b1={self.b1}, b2={self.b2}, b3={self.b3})"

class NonLinearModel(models.Model):
    b0 = models.FloatField()
    b1 = models.FloatField()
    b2 = models.FloatField()
    b3 = models.FloatField()

    r2 = models.FloatField(default=0.0)
    mmre = models.FloatField(default=0.0)
    pred_25 = models.FloatField(default=0.0)

    dataset = models.ForeignKey(Dataset, on_delete=models.CASCADE,
related_name='nonlinear_models', null=True, blank=True)

    normalization_method = models.CharField(max_length=100, default='lg10')

    def __str__(self):
        return f"NonLinearModel(b0={self.b0}, b1={self.b1}, b2={self.b2},
b3={self.b3}), normalization_method={self.normalization_method})"
```

Текст модуля math_utils.py

```
from xml.parsers.expat import model
import pandas as pd
import numpy as np
import random
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy import special
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error,
r2_score
from sklearn.preprocessing import StandardScaler
from scipy.spatial.distance import cdist
from pingouin import multivariate_normality
from scipy.stats import chi2

def mardia_test(df, alpha=0.05):
    """
    Виконує тест Мардіа на багатовимірну нормальність.
    Повертає словник із результатами тестів на Асиметрію та Ексцес.
    """

    data = df.values
```

```
    n, k = data.shape  # n - кількість спостережень, k - кількість змінних

    if n < k + 1:
        return "Недостатньо даних для тесту"

    # 1. Центрування та коваріація
    mean_vec = np.mean(data, axis=0)
    cov_matrix = np.cov(data, rowvar=False)

    try:
        inv_cov = np.linalg.inv(cov_matrix)
    except np.linalg.LinAlgError:
        return "Коваріаційна матриця вироджена (сингулярна)"

    # 2. Розрахунок відстані Махаланобіса між усіма парами (для асиметрії)
    # Формула D_ij = (x_i - mean)^T * S^-1 * (x_j - mean)
    # Ефективна реалізація через матричне множення
    centered_data = data - mean_vec
    # Z = X * S^(-1/2) - відбілені дані, але простіше порахувати матрицю квадратів
    # term = (X - mu) * inv_cov * (X - mu).T
    term = np.dot(np.dot(centered_data, inv_cov), centered_data.T)

    # --- ТЕСТ НА АСИМЕТРІЮ (SKEWNESS) ---
    # b1,k = (1/n^2) * sum(sum(g_ij^3))
    mardia_skew = np.sum(term**3) / (n**2)

    # Статистика тесту: A = (n * mardia_skew) / 6
    # Розподіл хі-квадрат зі ступенями вільності df = k(k+1)(k+2)/6
    skew_stat = (n * mardia_skew) / 6
    df_skew = k * (k + 1) * (k + 2) / 6
    p_skew = 1 - stats.chi2.cdf(skew_stat, df_skew)

    # --- ТЕСТ НА ЕКСЦЕС (KURTOSIS) ---
    # b2,k = (1/n) * sum(D_ii^2) -> діагональні елементи матриці term - це квадрати
Махаланобіса
    # Увага: term[i,i] це і є квадрат відстані Махаланобіса для і-го елемента
    mahal_sq = np.diag(term)
    mardia_kurt = np.sum(mahal_sq**2) / n

    # Статистика тесту для ексцесу (наближена до нормального розподілу N(0,1))
    # E[b2,k] = k(k+2)
    # Var[b2,k] = 8k(k+2)/n
    expected_kurt = k * (k + 2)
    var_kurt = 8 * k * (k + 2) / n
    kurt_stat = (mardia_kurt - expected_kurt) / np.sqrt(var_kurt)
    # Двосторонній тест
    p_kurt = 2 * (1 - stats.norm.cdf(abs(kurt_stat)))

    z_critical = stats.norm.ppf(1 - alpha/2)
    kurt_critical_value = expected_kurt + z_critical * np.sqrt(var_kurt)
```

```
    # --- РЕЗУЛЬТАТИ ---
    is_normal_skew = bool(p_skew > alpha)
    is_normal_kurt = bool(p_kurt > alpha)
    is_multivariate_normal = is_normal_skew and is_normal_kurt

    return {
        "Skewness Statistic": float(mardia_skew),
        "Skewness p-value": float(p_skew),
        "Normal Skewness?": is_normal_skew,

        "Kurtosis Statistic": float(mardia_kurt),
        "Kurtosis p-value": float(p_kurt),
        "expected_kurt": float(expected_kurt),
        "var_kurt": float(var_kurt),
        "kurt_stat": float(kurt_stat),
        "kurt_critical_value": float(kurt_critical_value),
        "Normal Kurtosis?": is_normal_kurt,

        "MULTIVARIATE NORMAL?": is_multivariate_normal
    }


def iterative_mahalanobis_one_by_one(df, alpha=0.05):
    """
    Видаляє викиди по одному (найгіршому за ітерацію) з перерахунком параметрів.

    :param df: Pandas DataFrame з числовими даними
    :param alpha: Рівень значимості
    :return: Очищений DataFrame
    """
    df_clean = df.copy()
    removed_indices = []

    # Кількість ступенів вільності = кількість змінних (колонок)
    degrees_of_freedom = df_clean.shape[1]

    # Критичне значення залежить від кількості змінних, воно стале,
    # але ми будемо порівнювати з ним на кожному кроці.
    critical_value = stats.chi2.ppf(1 - alpha, degrees_of_freedom)
    print(f"Критичне значення Chi2 (поріг): {critical_value:.4f}")
    print("-" * 65)
    print(f"{'Ітерація':<10} | {'Індекс':<10} | {'SMD':<12} | {'chi2 critical':<10} |
{'Різниця':<10}")
    print("-" * 65)

    iteration = 1
    removed_indices_info = []

    while True:
        # 1. Перевірка на мінімальну кількість даних
```

```
        if len(df_clean) <= degrees_of_freedom + 1:
            print("Занадто мало даних для продовження.")
            break

        # 2. Перерахунок метрик на поточному наборі даних
        mu = df_clean.mean().values
        cov = df_clean.cov().values

        try:
            inv_cov = np.linalg.inv(cov)
        except np.linalg.LinAlgError:
            print("Матриця коваріації вироджена. Зупинка.")
            break

        # 3. Розрахунок SMD для ВСІХ поточних точок
        diff = df_clean.values - mu
        left_term = np.dot(diff, inv_cov)
        mahal_sq = np.sum(left_term * diff, axis=1)

        # 4. Знаходимо ОДНУ найгіршу точку (максимальний SMD)
        max_smd = np.max(mahal_sq)
        max_smd_pos = np.argmax(mahal_sq) # Позиція в масиві (0, 1, 2...)

        # Отримуємо реальний індекс з DataFrame (бо він може бути 10, 55, 100...)
        worst_index_label = df_clean.index[max_smd_pos]

        # 5. Перевіряємо, чи цей максимум є викидом
        if max_smd > critical_value:
            diff_val = max_smd - critical_value

            print(f"{iteration:<10} | {worst_index_label:<10} | {max_smd:<12.4f}
|  {critical_value:<12.4f} | +{diff_val:<10.4f}")


            worst_index_x1 = df_clean.loc[worst_index_label, 'X1']
            worst_index_x2 = df_clean.loc[worst_index_label, 'X2']
            worst_index_x3 = df_clean.loc[worst_index_label, 'X3']
            worst_index_y = df_clean.loc[worst_index_label, 'Y']
            removed_indices_info.append((iteration, worst_index_label,
[worst_index_x1, worst_index_x2, worst_index_x3, worst_index_y], max_smd,
critical_value, diff_val))

            # Видаляємо тільки цього одного "поганця"
            df_clean = df_clean.drop(worst_index_label)
            removed_indices.append(worst_index_label)

            iteration += 1
        else:
            # Якщо навіть найгірша точка вписується в ліміт – ми закінчили
            print("-" * 65)
            print("Більше викидів не знайдено. Процес завершено.")
```

```
            break

    print(f"\nВсього видалено точок: {len(removed_indices)}")
    print(f"Список видалених індексів: {removed_indices}")

    return df_clean, removed_indices_info

def evaluate_model_metrics(y_true, y_pred, l=0.25):
    """
    Computes R2, MMRE, and Pred(l).
    y_true: Array of actual values
    y_pred: Array of predicted values
    l: Threshold for Pred calculation (default 0.25 for 25%)
    """
    # Ensure numpy arrays
    y_true = np.array(y_true).flatten()
    y_pred = np.array(y_pred).flatten()

    # R2
    r2 = r2_score(y_true, y_pred)

    # Calculate MRE array
    # add epsilon to avoid division by zero if necessary: (y_true + 1e-10)
    mre = np.abs((y_true - y_pred) / y_true)

    # MMRE
    mmre = np.mean(mre)

    # Pred(l)
    pred_val = np.sum(mre <= l) / len(y_true)

    return r2, mmre, pred_val

def compute_non_linear_regression(df):
    marida_t1 = mardia_test(df)

    # normalization
    df_normal = df.copy()
    df_normal = np.log10(df_normal)

    marida_t2 = mardia_test(df)

    # filter anomalies
    # Таблиця 2.3 - ітеративне вилучення викидів
    df_normal, removed_indices_info =
iterative_mahalanobis_one_by_one(df_normal.copy())

    # Перевірка нормальності після вилучення викидів
    marida_t3 = mardia_test(df_normal)
```

```
    model = LinearRegression()

    X = df_normal[["X1", "X2", "X3"]]
    y = df_normal["Y"]
    model.fit(X, y)
    pred_train = model.predict(X)

    y_pred_diff = pd.DataFrame({
        'y': y,
        'pred_train': pred_train,
        'val_diff': abs(y - pred_train)
    })
    y_pred_diff = y_pred_diff.sort_values(by='val_diff')
    y_pred_diff_filtr = y_pred_diff.iloc[:-4]

    # Usage
    r2, mmre, pred25 = evaluate_model_metrics(10 ** y_pred_diff_filtr['y'], 10 **
y_pred_diff_filtr['pred_train'], l=0.25)
    print(f"R2: {r2:.4f}, MMRE: {mmre:.4f}, Pred(0.25): {pred25:.4f}")


    result = {
        'marida_t1': marida_t1,
        'marida_t2': marida_t2,
        'marida_t3': marida_t3,
        'removed_indices_info': removed_indices_info,
        'metric_results': {
            'R2': r2,
            'MMRE': mmre,
            'Pred(0.25)': pred25
        },
        'model_coefficients': {
            'b0': model.intercept_,
            'b1': model.coef_[0],
            'b2': model.coef_[1],
            'b3': model.coef_[2],
        }
    }
    return result

def compute_linear_regression(df):

    df2 = np.log10(df.copy())
    df_filter, removed_indices_info = iterative_mahalanobis_one_by_one(df2.copy())
    df_linear = 10 ** df_filter.copy()

    model = LinearRegression()

    X = df_linear[["X1", "X2", "X3"]]
    y = df_linear["Y"]
    model.fit(X, y)
```

```
pred_train = model.predict(X)

# Usage
r2, mmre, pred25 = evaluate_model_metrics(y, pred_train, l=0.25)
print(f"R2: {r2:.4f}, MMRE: {mmre:.4f}, Pred(0.25): {pred25:.4f}")


result = {
    'metric_results': {
        'R2': r2,
        'MMRE': mmre,
        'Pred(0.25)': pred25
    },
    'model_coefficients': {
        'b0': model.intercept_,
        'b1': model.coef_[0],
        'b2': model.coef_[1],
        'b3': model.coef_[2],
    }
}
return result
```

ДОДАТОК Д
ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1 Загальні відомості

1.1 Позначення і найменування програми

Найменування: «Програмний засіб для оцінювання розміру застосунків мовою TypeScript».

1.2 Мови програмування та середовище виконання

Програма реалізована мовою Python (3.10+) з використанням фреймворку Django. Для проведення обчислень використано бібліотеки NumPy, Pandas та SciPy. Система підготовлена до розгортання в ізольованому середовищі Docker.

2 Функціональне призначення

2.1 Класи вирішуваних завдань

Програмне забезпечення призначене для автоматизації оцінювання обсягу вихідного коду SLOC. Програма вирішує наступні задачі:

– збір та зберігання архітектурних метрик у базі даних;

– статистична обробка даних (очищення від викидів, логарифмічна нормалізація);

– розрахунок прогнозу розміру ПЗ на основі лінійної та нелінійної регресійних моделей;

– надання аналітичної звітності щодо якості побудованих моделей для адміністратора.

2.2 Особливості архітектури застосунку

Архітектура системи побудована за модульним принципом, що дозволяє чітко відокремити логіку математичних обчислень від компонентів управління базою даних та інтерфейсу користувача. Такий підхід забезпечує високу гнучкість системи, дозволяючи в майбутньому легко додавати нові методи статистичного аналізу або розширювати перелік вхідних метрик без зміни загальної структури коду. Використання контейнеризації Docker гарантує ідентичність середовища виконання на різних операційних системах, що спрощує процес впровадження та підтримки програмного засобу.

3 Використовувані технічні засоби

Для стабільної роботи ПЗ необхідні технічні засоби з наступними характеристиками:

– процесор з архітектурою x64 (2.0 ГГц і вище);

– обсяг оперативної пам’яті (RAM) – не менше 2 Гб;

– вільне місце на диску – від 500 Мб;

– операційна система: Linux (рекомендовано), Windows або macOS.

4 Виклик та завантаження

Запуск системи здійснюється в терміналі за допомогою команди docker-compose up. Після ініціалізації контейнерів доступ до функцій програми надається через веббраузер за адресою http://localhost:8000/.

5 Вхідні та вихідні дані

5.1 Вхідні дані

Вхідними даними для системи є значення метрик NOC, AAC, TR та статистичні вибірки у форматі CSV.

5.2 Вихідні дані

Вихідними даними є:

– розрахований прогноз розміру коду в одиницях SLOC;

– аналітичні звіти з параметрами регресії.

Розроблений програмний продукт є завершеним інженерним рішенням, яке поєднує в собі потужний математичний апарат та зручний інтерфейс користувача. Система повністю відповідає вимогам, визначеним у технічному завданні, та забезпечує високу надійність розрахунків при плануванні розробки програмного забезпечення мовою TypeScript.