МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова

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

«Допущений до захисту»
Завідувач кафедри
_____
«___» _____ 2024 р.

КВАЛІФІКАЦІЙНАРОБОТА
на здобуття ступеня вищої освіти «магістр»

на тему: Удосконалення математичної моделі для оцінювання розміру програмних застосунків на C# та розробка програми для її реалізації.

Виконав: студент 6151м групи
_____ Ларіонов В. А.
(підпис, ПІБ)

Керівник роботи:
ст. викл.кафедри ПЗАС, к.т.н.
(посада, науковий ступень вчене звання)
_____ Каіров В.О.
(підпис, ПІБ)

Миколаїв – 2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова

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

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

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

Студенту Ларіонову Вадиму Анатолійовичу
(Прізвище, ім'я, по батькові)

1. Тема роботи: «Удосконалення математичної моделі для оцінювання розміру програмних застосунків на C# та розробка програми для її реалізації».

Керівник роботи Каіров В. О.

Затверджені наказом ректора № 1070 УЧ уч від «11» _____10_____ 2024 р.

2. Термін подання роботи: 09.12.2024 р. _____

3. Вихідні дані по роботі: _____

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

5. Перелік презентаційних матеріалів 1. Тема 2. Мета та завдання дослідження, 3. Об'єкт та предмет дослідження, 4. Методи дослідження, 5. Наукова новизна одержаних результатів, 6. Практичне значення одержаних результатів, 7. Особистий внесок здобувача 8. Апробація результатів досліджень та публікації, 9. Аналіз сучасних методів оцінювання розміру програмного забезпечення. 10. Обґрунтування необхідності проведення досліджень за обраною темою 11. Емпіричні дані проектів на C#, 12. Нормалізація даних та вилучення викидів, 13. Нелінійна регресійна модель, 14. Лінійна регресійна моделі, 15. Порівняння лінійної та нелінійної моделей, 16. Діаграма варіантів використання ПЗ для оцінювання застосунків на C#, 17. Діаграма діяльності ПЗ для оцінювання застосунків на C#, 18. Діаграма класів ПЗ для оцінювання застосунків на C#, 19. Діаграма послідовності для варіанту використання «Виконати розрахунки щодо розміру», 20. Інтерфейс користувача ПЗ для оцінювання застосунків на C#. 21. Висновки.

6. Консультанти розділів роботи

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

7. Дата видачі завдання 14.10.2024 р. _____

К А Л Е Н Д А Р Н И Й П Л А Н

Номер	Назва етапів роботи	Терміни виконання	Примітка
1.	Підготовка розділу Вступ	16.10.2024	НС*
2.	Підготовка розділу з огляду літератури та обґрунтування необхідності проведення досліджень за обраною темою	18.10.2024	НС*
3.	Підготовка розділу (ів) з результатів власних досліджень	21.10.2024	НС*
4.	Підготовка розділу з проекту програмного забезпечення	27.11.2024	
5.	Підготовка організаційно-економічного розділу	29.11.2024	
6.	Підготовка розділу з охорони праці	02.12.2024	
7.	Підготовка розділу з охорони навколишнього середовища	04.12.2024	
8.	Підготовка розділу Висновки	05.12.2024	
9.	Оформлення списку використаних джерел та додатків	06.12.2024	
10.	Подання на кафедру ПЗАС тексту остаточного варіанту роботи, підписаного її керівником, у роздрукованому та електронному форматі разом із заявами щодо самостійності виконання роботи та ідентичності друкованої та електронної версії роботи (Додатки 1 і 2 «Порядку здійснення заходів з перевірки робіт на наявність текстових збігів/ідентичності/схожості із використанням програмно-технічних засобів», який введений в дію наказом ректора НУК за №20 від 20.01.2020 р.)	09.12.2024	не пізніше, ніж за 14 діб до захисту (згідно п.4.1 зазначеного Порядку)
11.	Підготовка презентації та доповіді	13.12.2024	
12.	Попередній захист роботи на засіданні кафедри ПЗАС	16.12.2024	
13.	Подання на кафедру ПЗАС електронних версії наступних документів у форматі pdf: кваліфікаційної роботи; файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК за №287-уч від 19.05.2020 р.); презентації доповіді	29.12.2024	

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

Студент _____ Ларіонов В. А.
(підпис) (ПІБ)

Керівник роботи _____ Каіров В. О.
(підпис) (ПІБ)

РЕФЕРАТ

Ларіонов Вадим Анатолійович

«Удосконалення математичної моделі для оцінювання розміру програмних застосунків на C# та розробка програми для її реалізації»

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

Обсяг роботи: 81 стор., 14 табл., 8 рис., 28 використаних джерел, 5 додатків

Актуальність теми: полягає в необхідності більш точного оцінювання розміру програмних застосунків на C# через відсутність побудованих трьохфакторних моделей для цієї мови.

Мета дослідження: метою роботи є підвищення достовірності оцінювання розміру програмних застосунків на C# за допомогою нелінійної регресійної моделі.

Об’єкт дослідження: процес оцінювання розміру програмних застосунків на C#.

Предмет дослідження: нелінійні регресійні моделі для оцінювання розміру програмних застосунків на C#.

Методи дослідження: методи ймовірнісного аналізу, математичної статистики, регресійного аналізу, оптимізації, чисельні методи, принципи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів: було вдосконалено трьохфакторну нелінійну регресійну модель для оцінювання розміру програмних застосунків на C#, яка базується на одновимірному нормалізуючому перетворенні десяткового логарифму та аналізі викидів регресії. У порівнянні з лінійною регресійною моделлю, побудованою за умови нормальності розподілу метрик, нова модель демонструє вищий множинний коефіцієнт детермінації R^2, нижчу середню величину відносної похибки MMRE та більший відсоток прогнозованих результатів PRED(0,25).

Практичне значення одержаних результатів. У результаті виконання роботи було створено інструмент, що дозволяє будувати нелінійні регресійні моделі для оцінювання програмних застосунків на C# на етапі планування проекту.

Апробація результатів роботи. Основні положення й результати кваліфікаційної роботи опубліковані на VII Всеукраїнській науково-практичній інтернет-конференції «Сучасні інформаційні системи та технології» (м. Херсон, м. Хмельницький, 29 листопада 2024 року).

Публікації: Основні положення й результати кваліфікаційної роботи опубліковано у 1 науковій праці – тезах конференції.

Ключові слова: оцінювання розміру програмного забезпечення, нелінійні регресійні моделі, нормалізуючі перетворення, C#.

ABSTRACT

Larionov Vadym Anatoliyovych

«Improvement of a mathematical model for estimating the size of software applications in C# and development of a program for its implementation»

Qualification work for obtaining a master's degree in specialty 121 – "Software Engineering". Admiral Makarov National University of Shipbuilding. Mykolaiv, 2024.

Volume: 81 p., 14 tables, 8 figures, 28 references, 5 appendices.

Topic Relevance: is the issue of more precise estimation of the size of C#-based application due to the lack of built regression models for this programming language.

Research goal. The goal is to increase the confidence of estimation the size of C#-based applications using a nonlinear regression model.

Object of research: the process of estimating the size of C#-based applications.

Subject of research: the nonlinear regression models for estimating the size of C#-based applications.

Methods of research: probabilistic analysis, mathematical statistics, regression analysis, optimization, object-oriented programming, and numerical methods.

Scientific contribution: A three-factor nonlinear regression model for estimating the size of C# applications was enhanced. The model uses a univariate normalizing transformation based on the decimal logarithm and regression outlier analysis. Compared to the linear regression model, which assumes normal distribution of metrics, the new model demonstrates a higher multiple coefficient of determination (R^2), lower mean magnitude of relative error (MMRE), and a greater percentage of predicted results (PRED(0.25)).

Practical value of obtained results. As a result of the work, a tool was developed to construct nonlinear regression models for estimating the size of C# applications at the project planning stage.

Approbation of the thesis results. . The results of the research presented in the paper were made public at the 7th All-Ukrainian Scientific and Practical Internet Conference of students, postgraduates and young scientists on the topic: "Modern

computer systems and networks in management" (Khmelnytskyi, Kherson, November 30, 2024).

Publications. The main provisions and results of the qualification work are published in 1 scientific work – theses of the conference.

Keywords: LOC estimation, code number lines, non-linear regression models, normalizing transformations, C#.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

ПК – Персональний комп’ютер

COCOMO – COnstructive COst Model

NOC – Number Of Classes

DIT – Depth of Inheritance Tree

KLOC – Thousand lines of code

LB – Lower Bound

UP – Upper Bound

LOC – Lines Of Code

MMRE – Mean Magnitude of Relative Error

MRE – Magnitude of Relative Error

PHP – Hypertext Preprocessor

PRED – Percentage of prediction

SMD – the Square Mahalanobis Distance

VIF – Variance Influence Factor

ВСТУП

Актуальність теми. Визначення розміру програмного забезпечення (ПЗ) на початкових етапах є однією з ключових задач для керівників проектів, адже саме від достовірності оцінювання залежать прогнози трудомісткості, тривалості та витрат. Помилки в оцінюванні розмірів можуть стати причиною невдалих проектів [1].

Попри широкий вибір моделей і методів, точність оцінювання на ранніх етапах залишається низькою [2-6]. Побудова спеціалізованих моделей для певних мов програмування або конкретних типів застосунків може допомогти вирішити цю проблему [7]. Методи нелінійного регресійного аналізу, доповнені нормалізуючими перетвореннями, відкривають значні перспективи для достовірного оцінювання [7-11].

Мова C# є найпопулярнішою для платформи .NET – безкоштовного, кросплатформного середовища розробки з відкритим кодом. C# дозволяє створювати програми для різних пристроїв: від IoT до хмарних рішень, включаючи застосунки для телефонів, ПК і серверів. C# – це мова загального призначення, що забезпечує високу продуктивність і ефективність. На основі об’єктно-орієнтованих принципів, вона включає функції з інших парадигм, зокрема функціонального програмування.

Для оцінювання розміру програмного забезпечення методом LOC створено численні лінійні та нелінійні моделі, зокрема для мов C++, Java, Kotlin, PHP, Visual Basic [7-14]. Вони базуються на метриках класових діаграм, що дають змогу оцінювати структуру класів ще на стадії проєктування. Для C# розроблено двофакторну модель для веб-застосунків і однофакторну модель для мобільних застосунків [15,16]. Відсутність трьохфакторних моделей для C# слугує причиною проведення досліджень.

Для побудови нелінійної регресійної моделі вибрано нормалізуюче перетворення log10. Це перетворення має значний досвід успішного застосування

у таких моделях, як COCOMO, ISBSG, COCOMO II, а також у багатьох інших нелінійних регресійних моделях.

Мета та завдання дослідження: метою роботи є удосконалення математичної моделі для оцінювання розміру програмних застосунків на C# за допомогою нелінійної регресійної моделі.

Для досягнення цієї мети слід виконати такі завдання:

– Провести аналіз існуючих методів оцінювання розміру програмних застосунків на C#.

– Зібрати дані, необхідні для побудови нелінійної регресійної моделі оцінювання розміру програмних застосунків на C#.

– Удосконалити нелінійну регресійну модель із використанням нормалізуючих перетворень.

– Розробити програмне забезпечення для реалізації розрахунків за створеною моделлю.

Об’єкт дослідження: процес оцінювання розміру програмних застосунків на C#.

Предмет дослідження: нелінійні регресійні моделі для оцінювання розміру програмних застосунків на C#.

Методи дослідження: методи ймовірнісного аналізу, математичної статистики, регресійного аналізу, оптимізації, чисельні методи, принципи об'єктно-орієнтованого програмування.

Наукова новизна одержаних результатів: Було вдосконалено трьохфакторну нелінійну регресійну модель для оцінювання розміру програмних застосунків на C#, яка базується на одновимірному нормалізуючому перетворенні десяткового логарифму та аналізі викидів регресії. У порівнянні з лінійною регресійною моделлю, побудованою за умови нормальності розподілу метрик, нова модель демонструє вищий множинний коефіцієнт детермінації R^2, нижчу середню величину відносної похибки MMRE та більший відсоток прогнозованих результатів PRED(0,25).

Практичне значення одержаних результатів. У результаті виконання роботи було створено інструмент, що дозволяє будувати нелінійні регресійні моделі для оцінювання програмних застосунків на C# на етапі планування проекту.

Особистий внесок здобувача. Кваліфікаційна робота є самостійно виконаною науковою працею. Усі наукові результати отримано автором особисто. У праці [17], яка була опублікована у співавторстві, автору належить нелінійна регресійна модель для оцінювання розміру програмних застосунків на C# на основі нормалізуючого перетворення десяткового логарифму.

Апробація результатів роботи. Основні положення й результати кваліфікаційної роботи опубліковано на VII Всеукраїнській науково-практичній інтернет-конференції «Сучасні інформаційні системи та технології» (м. Херсон, м. Хмельницький, 29 листопада 2024 року).

Публікації: Основні положення й результати кваліфікаційної роботи опубліковано у 1 науковій праці – тезах конференції.

1 АНАЛІЗ НАЯВНИХ ПІДХОДІВ ДО ОЦІНЮВАННЯ РОЗМІРУ C#-ЗАСТОСУНКІВ І ВИЗНАЧЕННЯ ДОЦІЛЬНОСТІ ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

1.1 Дослідження наявних методів і математичних моделей для визначення розмірів програмних застосунків

Визначення розміру ПЗ на початкових етапах є однією з ключових задач для керівників проєктів, адже саме від точності оцінювання залежать прогнози трудомісткості, тривалості та витрат. Помилки в оцінюванні розмірів можуть стати причиною невдалих проєктів [1].

Попри широкий вибір моделей і методів, достовірність оцінювання на ранніх етапах залишається низькою [2-6]. Побудова спеціалізованих моделей для певних мов програмування або конкретних типів застосунків може допомогти вирішити цю проблему [7]. Методи нелінійного регресійного аналізу, доповнені нормалізуючими перетвореннями, відкривають значні перспективи для достовірного оцінювання [7-11].

Мова C# є найпопулярнішою для платформи .NET – безкоштовного, кросплатформного середовища розробки з відкритим кодом. C# дозволяє створювати програми для різних пристроїв: від IoT до хмарних рішень, включаючи застосунки для телефонів, ПК і серверів. C# – це мова загального призначення, що забезпечує високу продуктивність і ефективність. На основі об’єктно-орієнтованих принципів, вона включає функції з інших парадигм, зокрема функціонального програмування.

Основні причини застосування C# та .NETсьогодні:

– Кросплатформова розробка – C# та .NET дозволяють створювати програми для Windows, macOS, Linux, iOS і Android, забезпечуючи широку аудиторію користувачів.

– Інтеграція з продуктами Microsoft – володіння C# та .NET спрощує роботу з платформами Microsoft, такими як Azure і Visual Studio, а також відкриває можливості для участі в AI-проектах, як-от ChatGPT і Bing AI.

– Масштабованість – ці технології підходять для малих і великих застосунків, включаючи десктопні, веб, мобільні та ігрові програми.

– Попит на ринку праці – розробники C# та .NET користуються великим попитом серед стартапів і великих корпорацій, що підвищує шанси працевлаштування.

– Активна спільнота розробників – широка база знань, форуми та документація допомагають розвиватися як новачкам, так і досвідченим розробникам.

– Легкість у навчанні – C# має простий і потужний синтаксис, що дозволяє швидко освоїти мову, особливо тим, хто знайомий із Java чи C++

Для оцінювання розміру програмного забезпечення за методом LOC було розроблено чимало як лінійних, так і нелінійних регресійних моделей для мов програмування, таких як C++, Java, Kotlin, PHP, Visual Basic тощо [7-14]. Ці моделі базувалися на метриках діаграми класів, оскільки вже на стадії проектування можна отримати концептуальне уявлення про структуру класів. Для мови програмування C# відомо про двофакторну модель [15] для оцінювання розміру веб застосунків з предикторами кількість класів та середня кількість методів на клас, а також відомо про однофакторну нелінійну регресійну модель для мобільних застосунків з використанням платформи Xamarin [16]. Більша частина відомих та ефективних нелінійних регресійних моделей для оцінювання розміру програмного забезпечення, є трьохфаторними, що разом з відсутністю трьохфакторних моделей є аргументом на користь застосування методів багатофакторного регресійного аналізу.

Для побудови нелінійної регресійної моделі вибрано нормалізуюче перетворення log10. Це перетворення має значний досвід успішного застосування у таких моделях, як COCOMO, ISBSG, COCOMO II, а також у багатьох іншних нелінійних регресійних моделях.

Для нормалізованих даних використовується лінійна регресійна модель [18]

$$Y = \hat{Y} + \varepsilon = b_0 + b_1 X_1 + b_2 X_2 + \ldots + b_k X_k + \varepsilon, \quad (1.1)$$

де $\hat{Y}$ – прогнозне значення, b_i $(i = \overline{0,k})$ – коефіцієнти моделі, X_i $(i = \overline{1,k})$ – значення незалежних змінних, ε – гаусівська випадкова величина, k – кількість предикторів.

Довірчий інтервал [18]

$$\hat{Y} \pm t_{\alpha/2,\mathrm{v}} S_Y \left\{ \frac{1}{N} + (\mathbf{x}^+)^T [\mathbf{S_X}]^{-1} (\mathbf{x}^+) \right\}^{1\backslash 2}, \quad (1.2)$$

Інтервал передбачення [18]

$$\hat{Y} \pm t_{\alpha/2,\mathrm{v}} S_Y \left\{ 1 + \frac{1}{N} + (\mathbf{x}^+)^T [\mathbf{S_X}]^{-1} (\mathbf{x}^+) \right\}^{1\backslash 2}, \quad (1.3)$$

де $S_Y^2 = \frac{1}{\mathrm{v}} \sum_{i=1}^{N} \left(Y_i - \hat{Y}_i \right)^2$; $\mathrm{v} = N - k - 1$; $\mathbf{S_X} - k \times k$ матриця [18]

$$\mathbf{S_X} = \begin{pmatrix} S_{X_1 X_1} & S_{X_1 X_2} & \ldots & S_{X_1 X_k} \\ S_{X_1 X_2} & S_{X_2 X_2} & \ldots & S_{X_2 X_k} \\ \ldots & \ldots & \ldots & \ldots \\ S_{X_1 X_k} & S_{X_2 X_k} & \ldots & S_{X_k X_k} \end{pmatrix}, \quad (1.4)$$

де $S_{X_q X_r} = \sum_{i=1}^{N} \left[X_{q_i} - \bar{X}_q \right] \left[X_{r_i} - \bar{X}_r \right]$, $q, r = \overline{1,k}$.

Негаусівські початкові дані вимагають додаткових зусиль для побудови якісної нелінійної регресійної моделі. Для цього існують три основні підходи: метод підбору, лінеаризація даних або нормалізація. Метод нормалізації є найбільш раціональним, оскільки дозволяє знизити витрати часу й отримати стабільну модель з високою точністю.

Алгоритм нормалізації включає шість ключових кроків [19, 20]:

– Застосування обраного нормалізуючого перетворення до початкових даних

– Видалення викидів із трансформованих даних.

– Побудова лінійної регресійної моделі на очищених даних.

– Тестування залишків моделі за критерієм Пірсона. Якщо вони не відповідають нормальному розподілу, рядок із найбільшим відхиленням видаляється, і цикл повторюється.

– Затосування зворотного перетворення для створення остаточної нелінійної моделі.

– Розрахунок довірчих інтервалів й відкидання рядків, що виходять за їх межі. У випадку відсутності таких рядків, модель вважається побудованою.

Для нормалізації ненормальних даних застосовується функція перетворення

$$\mathrm{T} = \psi(\mathrm{P}), \tag{1.5}$$

тут $\mathrm{P} = \{Y, X_1, X_2, \dots, X_k\}^T$ – початковий вектор даних, $\mathrm{T} = \{Z_Y, Z_1, Z_2, \dots, Z_k\}^T$ – нормалізований вектор, $\psi = \{\psi_Y, \psi_1, \psi_2, \dots, \psi_k\}^T$ – використане нормалізуюче перетворення.

Обернене перетворення описується рівнянням

$$\mathrm{P} = \psi^{-1}(\mathrm{T}). \tag{1.6}$$

Нелінійна регресійна модель представлена рівнянням [18]

$$Y = \psi_Y^{-1}(\hat{Z}_Y + \varepsilon) = \psi_Y^{-1}(b_0 + b_1 Z_1 + b_2 Z_2 + \dots + b_k Z_k + \varepsilon), \tag{1.7}$$

де ψ_Y – виконує нормалізацію залежної змінної. (1.5).

Довірчий інтервал розраховується за формулою [18]

$$\Psi_Y^{-1}\left(\hat{Z}_Y \pm t_{\frac{\alpha}{2},\nu} S_{Z_Y}\left\{\frac{1}{N} + (\boldsymbol{z}_X^+)^T[\mathbf{Z_X}]^{-1}(\boldsymbol{z}_X^+)\right\}^{1\backslash 2}\right). \tag{1.8}$$

Інтервал передбачення включає додаткову складову [18]

$$\psi_Y^{-1}\left(\hat{Z}_Y \pm t_{\frac{\alpha}{2},v} S_{Z_Y}\left\{1+\frac{1}{N}+(\boldsymbol{z}_X^+)^T[\mathbf{Z_X}]^{-1}(\boldsymbol{z}_X^+)\right\}^{1\backslash 2}\right). \quad (1.9)$$

1.2 Обґрунтування необхідності проведення досліджень

Лінійні регресійні моделі, хоча й широко використовуються в оцінюванні розміру програмного забезпечення, мають серйозні обмеження. Зокрема, вони базуються на передумовах, таких як нормальність розподілу залишків і лінійна залежність між змінними, які рідко виконуються в реальних умовах.

На відміну від цього, нелінійні регресійні моделі не вимагають строгого дотримання цих умов і можуть враховувати більш складні взаємозв'язки. Наприклад, використання методів нормалізуючих перетворень дозволяє працювати з негаусівськими даними та забезпечує точнішу адаптацію моделі до специфіки задачі.

Такий підхід особливо корисний для випадків із великою кількістю метрик і змінних, де традиційні лінійні методи виявляються недостатньо ефективними. Крім того, нелінійні моделі забезпечують більшу стійкість до аномалій у даних (викидів) [19, 20].

C# є найпопулярнішою мовою для .NET – кросплатформного середовища розробки з відкритим кодом. Вона дозволяє створювати програми для різних пристроїв і платформ, забезпечуючи масштабованість, високу продуктивність і інтеграцію з продуктами Microsoft, такими як Azure. Мова має простий синтаксис, значну підтримку спільноти, а також попит серед роботодавців. Численні дослідження щодо LOC оцінювання програмного забезпечення показують, що є потреба будувати нелінійні регресійні моделі для мови програмування, фреймворків, типів проектів, тощо [7-14]. Трьохфакторні нелінійні регресійні моделі показують гарні результати при оцінювання програмного забезпечення. Відсутність трьохфакторних нелінійних регресійних моделей для оцінювання

розміру програмних застосунків на C# ставить питання про проведення досліджень за цією темою.

1.3 Висновки до розділу 1

У розділі 1 виконано аналіз сучасних підходів до оцінювання розміру ПЗ, зокрема для C#. Встановлено, що лінійні моделі мають обмеження у складних реальних задачах, де порушуються передумови їх застосування. Натомість нелінійні моделі, особливо трьохфакторні, демонструють високу ефективність у таких задачах. Було обґрунтовано необхідність створення трьохфакторних моделей для C#, враховуючи важливість цієї мови для кросплатформної розробки, що в свою чергу відкриває перспективи для підвищення достовірності такого оцінювання.

2 СТВОРЕННЯ НЕЛІНІЙНОЇ РЕГРЕСІЙНОЇ МОДЕЛІ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ПРОГРАМНИХ ЗАСТОСУНКІВ НА C#

2.1 Дослідження даних для побудови нелінійної регресійної моделі оцінювання розміру C#-застосунків: ідентифікація викидів та аналіз мультиколінеарності

Нелінійну регресійну модель для програмних застосунків на C# буде побудовано на основі 40 проектів, вибраних із платформи GitHub (https://github.com/). Використання плагіну Visual Studio Metrics дозволило отримати метрики діаграми класів: кількість рядків коду у тисячах (KLOC, Y), кількість класів (X_1), також глибину дерева успадкування (DIT, X_2), зв'язність класів (X_3), перелік даних наведено у таблиці 2.1.

Таблиця 2.1 – Емпіричні дані з метрик

№	Y	X_1	X_2	X_3	SMD_X
1	2	3	4	5	6
1	11,939	221	1,113	6,439	1,5741
2	28,368	507	1,229	8,588	0,6915
3	35,835	253	1,071	15,399	6,7964
4	163,373	1440	1,140	10,115	2,2799
5	345,963	2896	1,233	9,227	2,8149
6	17,562	119	1,319	6,336	3,0203
7	97,386	1469	1,100	6,792	1,4495
8	503,955	7040	1,640	7,885	5,1760
9	531,565	5504	1,151	6,567	4,9026
10	79,208	137	1,095	5,971	**13,9549**
11	37,286	704	1,224	6,548	0,5654
12	325,499	1585	1,517	12,729	4,8812
13	97,735	671	1,027	5,058	5,6115
14	492,065	5425	1,385	9,871	3,8191
15	34,313	290	1,245	7,569	0,5853
16	12,424	119	2,076	6,765	**10,5000**
17	361,592	1805	1,639	10,983	4,9684
18	95,203	1717	1,164	6,997	1,2114

Продовження табл. 2.1

1	2	3	4	5	6
19	34,469	319	1,859	7,564	5,0853
20	7,901	134	1,060	11,075	5,0116
21	21,214	295	1,546	9,095	1,8628
22	30,459	459	1,094	9,704	1,7731
23	4,534	68	1,088	6,074	3,7645
24	14,112	385	0,977	5,239	3,9594
25	33,601	1281	1,402	6,567	3,3871
26	19,466	121	1,347	18,281	8,7009
27	84,992	1516	1,131	12,894	5,3296
28	106,324	1072	1,478	9,338	1,0736
29	33,601	1281	1,402	6,567	3,3871
30	58,782	2117	1,009	6,341	4,6160
31	2,067	40	1,325	8,525	6,2805
32	20,617	209	0,919	6,967	3,6435
33	101,499	1364	1,365	7,778	0,5630
34	25,053	716	1,437	7,212	2,2612
35	13,175	88	1,216	6,670	3,0466
36	10,067	131	1,221	7,221	1,3792
37	15,115	140	1,029	5,679	3,4341
38	45,823	2755	1,358	3,870	9,1938
39	28,443	231	1,779	11,463	4,8556
40	119,854	1466	1,485	5,846	2,5891

Для обчислення відстані Махаланобіса (SMD) для і-тої точки багатовимірного набору даних застосовується формула [18]

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

де X_i – i-та точка вибірки X; $\overline{X}$ – вектор середніх значень; S_N – коваріаційна матриця, яка визначається [18]

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

Для ряду спостережень (10, 16) значення d_i^2 перевищують критичний поріг (9,488) для рівня значущості 0,005, що вказує на ненормальний розподіл метрик.

Для подальшого підтвердження використовувався тест на багатовимірний ексцес $\hat{\beta}_2$, обчислюваний за формулою [21]

$$\hat{\beta}_2 = \frac{1}{N}\sum_{i=1}^{N}\{(X_i - \bar{X})^T S_N^{-1}(X_i - \bar{X})\}^2. \quad (2.3)$$

Для даних метрик отримано $\hat{\beta}_2 = 87{,}3$, що перевищує критичне значення $\beta_2 = m(m+2) = 24$ (де $m = 4$). Це ще раз доводить, що метрики програмних застосунків на C# не відповідають нормальному розподілу.

Мультиколінеарність визначає зв'язок між незалежними змінними в регресійній моделі. Вона заважає оцінювати вплив кожного регресора на залежну змінну, коли змінні сильно корелюють.

Використання VIFs (Variance Inflation Factors) допомагає виявити проблему. Значення VIFs у межах 1–5 вказують на її відсутність, тоді як показники понад 10 вимагають втручання [22].

Мультиколнеарність відсутня, адже отримано значення для предикторів X_1, X_2, X_3 1,283, 1,301 і 1,183 відповідно.

Обчислення SMD (квадрата відстані Махаланобіса) для ненормалізованих даних вказує на відхилення розподілу від нормального. Для обробки ненормальних даних застосовувався метод, який поєднує нормалізуючі перетворення та розрахунок SMD [23, 24]. Формула для нормалізованих даних [18]:

$$d_i^2 = (Z_i - \overline{Z})^T S_N^{-1}(Z_i - \overline{Z}) , \quad (2.4)$$

де Z_i – i-та точка багатовимірного нормалізованого вектора; $\overline{Z}$ – вектор середніх значень; S_N – коваріаційна матриця, обчислювана як [18]

$$S_N = \frac{1}{N}\sum_{i=1}^{N}(Z_i - \overline{Z})\,(Z_i - \overline{Z})^T. \quad (2.5)$$

Нормалізуюче перетворення log10 визначається через $\psi = \{\lg Y, \lg X_1, \lg X_2, \lg X_3\}^T$

Для багатовимірного нормального розподілу SMD наближаються до розподілу χ^2 , і для великих вибірок ($N - m > 25$) значення d_i^2 можна вважати незалежною випадковою величиною $\chi^2_{m,\alpha}$ [25].

Аналіз вихідних даних показав, що за рівня значущості 0,005 всі викиди були видалені за дві ітерації. Детальні значення SMD на кожній ітерації наведено в таблиці 2.2.

Таблиця 2.2 – Пошук та вилучення викидів

№	SMD_1	SMD_2	SMD_3	№	SMD_1	SMD_2	SMD_3
1	1,5741	1,5245	1,4674	21	1,8628	1,8668	2,4050
2	0,6915	0,7432	0,6985	22	1,7731	1,7353	1,8293
3	6,7964	6,6123	6,7773	23	3,7645	3,8659	3,8566
4	2,2799	2,3107	2,4718	24	3,9594	3,8379	3,8110
5	2,8149	2,9185	2,9321	25	3,3871	4,1520	4,1135
6	3,0203	4,2685	4,8624	26	8,7009	8,4823	8,2559
7	1,4495	1,4641	1,5378	27	5,3296	5,7864	6,2417
8	5,1760	5,0511	5,0445	28	1,0736	1,0283	1,1263
9	4,9026	5,2453	5,2877	29	3,3871	4,1520	4,1135
10	**13,9549**	-	-	30	4,6160	4,7926	5,2136
11	0,5654	0,5264	0,4994	31	6,2805	6,2772	6,6685
12	4,8812	4,9647	4,8655	32	3,6435	4,2704	4,3523
13	5,6115	8,0971	7,8737	33	0,5630	0,5270	0,5291
14	3,8191	3,7020	3,6235	34	2,2612	2,7321	2,8544
15	0,5853	1,0525	1,1096	35	3,0466	4,3353	4,6048
16	10,5000	**10,3106**	-	36	1,3792	1,4120	1,5361
17	4,9684	5,1581	5,2961	37	3,4341	4,6404	4,5104
18	1,2114	1,1564	1,1774	38	9,1938	9,4876	9,3714
19	5,0853	5,0139	7,1125	39	4,8556	4,7150	5,8740
20	5,0116	5,0628	5,0285	40	2,5891	2,7210	3,0663

У подальшому аналізі використовується очищений набір із 38 проектів програмних застосунків на C#, з якого видалено всі викиди. Одним із ключових критеріїв, що теоретично обґрунтовує застосування лінійної регресії, є

нормальність розподілу залишків регресії ε. Для великих вибірок ($n > 30$) нормальність розподілу залишків можна перевірити за допомогою критерію Пірсона [26].

Значення χ^2 розраховується за формулою [26]

$$\chi^2 = \sum_{j=1}^{m} \frac{\left(n_j - np_j\right)^2}{np_j}, \tag{2.6}$$

де m – кількість інтервалів, на які розбивається діапазон $[x_{min}, x_{max}]$; n_j – абсолютна частота в j-му інтервалі; p_j – ймовірність потрапляння значення X у j-ий інтервал.

Кількість інтервалів m визначається за формулами:

– Формула Старджеса: $m = \log_2 n + 1 = 3{,}3lgn + 1$.

– Формула Брукса і Карузера: $m = 5lgn$.

Ймовірність p_j знаходиться інтегруванням щільності ймовірності [26]

$$\boldsymbol{p_j = \int_{x_{j-1}}^{x_j} f(x)dx}, \tag{2.7}$$

де $f(x)$ – функція щільності ймовірності.

Щільність ймовірності нормального розподілу описується [26]

$$\boldsymbol{f(x) = \frac{1}{\sigma_x\sqrt{2\pi}} * e^{\frac{-(x-m_x)^2}{2\sigma_x^2}}}, \tag{2.8}$$

де m_x – математичне сподівання величини, σ_x – середьоквадратичне відхилення.

Ступені свободи ν визначаються як $\nu = m - k - 1$, де $k = 2$ – кількість параметрів розподілу для нормальної функції. Розраховане значення χ^2 порівнюється з критичним значенням для рівня значущості α і відповідної кількості

ступенів свободи. Якщо $\chi^2 < \chi^2_{\text{кр}}$, нульова гіпотеза приймається з ймовірністю 1−α, в іншому випадку вона відкидається.

2.2 Розробка нелінійної регресійної моделі для оцінювання розміру C#-застосунків

Для побудови нелінійної регресійної моделі спершу необхідно побудувати лінійну модель для нормалізованих даних, у яких відсутні викиди. Модель може бути описана як

$$Z_Y = \hat{Z}_Y + \varepsilon = \hat{b}_0 + \hat{b}_1 Z_1 + \hat{b}_2 Z_2 + \hat{b}_3 Z_3 + \varepsilon, \quad (2.9)$$

де $Z_y = \lg Y; Z_i = \lg X_i, i = \overline{1,3}$.

Після побудови моделі перевіряється нульова гіпотеза про нормальність розподілу залишків (ε) за допомогою критерію Пірсона. Емпіричне значенняχ^2, розраховане за формулою (2.6), має бути меншим за 7,814 (при 3 ступенях свободи і рівні значущості 0,05). У разі відхилення нульової гіпотези найбільше за модулем значення залишків (ε) виключається як викид.

На основі рівняння (2.9) та перетворення (1.5) отримується кінцевий вигляд нелінійної моделі

$$Y = 10^{\varepsilon + \hat{b}_0} X_1^{\hat{b}_1} X_2^{\hat{b}_2} X_3^{\hat{b}_3}. \quad (2.10)$$

Після побудови моделі знаходяться інтервали передбачення, і всі значення, які виходять за їх межі, виключаються як викиди. Потім процес повторюється із етапу застосування SMD для виявлення викидів.

Процес ітераційної побудови нелінійної регресійної моделі був проведений наступним чином:

– Перша ітерація – Для даних без викидів побудовано лінійну регресійну модель з параметрами: $\hat{b}_0 = -1{,}7136, \hat{b}_1 = 0{,}9233, \hat{b}_2 = 0{,}2606, \hat{b}_3 = 0{,}8673$. Емпіричне значення $\chi^2 = 7{,}98$, що перевищує критичне значення 7,814. Нульову

гіпотезу відхилено, рядок 13 визнано викидом та виключено. Процес розпочато заново.

– Друга ітерація – Для даних без проектів 10, 13, 16 емпіричне значення $\chi^2 = 9{,}4$. Рядок 25 визначено як викид і виключено. Модель побудована знову.

– Третя ітерація – Видалено проекти 10, 13, 16, 25. Проект 38 визнано викидом за SMD, його виключено.

– Четверта ітерація – Дані без проектів 10, 13, 16, 25, 38. Значення $\chi^2 = 9{,}64$, ідентифіковано новий викид — рядок 29, який також виключено.

– П’ята ітерація – Дані без проектів 10, 13, 16, 25, 38, 29. Значення $\chi^2 = 5{,}29$, що менше критичного значення. Інтервали передбачення не виявили нових викидів.

Фінальні параметри моделі:

$$\hat{b}_0 = -1{,}6047, \hat{b}_1 = 0{,}9545, \hat{b}_2 = 0{,}7950, \hat{b}_3 = 0{,}6169$$

Оцінки якості моделі [27, 28]:

– $R^2 = 0{,}851$, відповідає критичному значенню ($R^2 > 0{,}75$)

– $MMRE =$ 0,346б перевищує критичне значення ($MMRE \leq 0{,}25$);

– $PRED(0{,}25) = 0{,}441$ нижче критичного значення ($PRED(0{,}25) > 0{,}75$).

Матрицю S_Z було використано для отримання довірчих інтервалів та інтервалів передбачення за (1.8) і (1.9)

$$S_Z = \begin{pmatrix} 11{,}3679 & 0{,}2751 & 0{,}0924 \\ 0{,}2751 & 0{,}1899 & 0{,}0858 \\ 0{,}0924 & 0{,}0858 & 0{,}5442 \end{pmatrix}.$$

Для оцінювання розміру програмних застосунків на C# було побудовано багатофакторну лінійну регресійну модель із припущенням про нормальність розподілу метрик. Подальше порівняння цієї моделі з нелінійною регресійною моделлю дозволяє оцінити переваги нелінійного підходу щодо достовірності прогнозів. Лінійна модель описується рівнянням

$$Y = \hat{Y} + \varepsilon = \hat{b}_0 + \hat{b}_1 X_1 + \hat{b}_2 X_2 + \hat{b}_3 X_3 + \varepsilon. \quad (2.12)$$

Оцінки параметрів для цієї моделі становлять:

$$\hat{b}_0 = -120{,}0907, \hat{b}_1 = 0{,}0847, \hat{b}_2 = 58{,}9595, \hat{b}_3 = 6{,}0023.$$

Перевірка нормальності розподілу залишків є проводилася за критерієм Пірсона, розрахованим за формулою (2.7). Для залишків регресії, отриманих із 34 проектів, емпіричне значення критерію Пірсона становить $\chi^2 = 47{,}53$. Критичне значення для трьох ступенів свободи на рівні значущості 0,05 дорівнює $\chi^2 = 7{,}814$ Оскільки $\chi^2 > \chi^2_{\text{кр}}$, нульова гіпотеза про гаусівський розподіл залишків відхиляється.

Додатково визначено характеристики залишків: математичне 111,77 та середньоквадратичне відхилення 156,95.

Оцінки якості моделі:

– $R^2 = 0{,}867$, відповідає критичному значенню ($R^2 > 0{,}75$)

– $MMRE =$ 0,989 перевищує критичне значення ($MMRE \leq 0{,}25$);

– $PRED(0{,}25) = 0{,}235$ нижче критичного значення ($PRED(0{,}25) > 0{,}75$).

Матрицю S_X було використано для отриманні довірчі інтервалів та інтервали передбачення за (1.2) і (1.3)

$$S_X = \begin{pmatrix} 93263650{,}1176 & 2396{,}5558 & 7601{,}8377 \\ 2396{,}5558 & 1{,}7736 & 4{,}3984 \\ -7601{,}8377 & 4{,}3984 & 277{,}9799 \end{pmatrix}$$

Результати аналізу, представлені в таблиці 2.3, свідчать про те, що довірчі інтервали нелінійної регресії, розраховані за формулою (1.8), у 76,47% випадків (26 із 34) є вужчими, ніж для лінійної моделі, обчисленої за формулою (1.2).

Таблиця 2.3 – Довірчі інтервали лінійної та нелінійної регресії

№	KLOC (Y)	Межі довірчих інтервалів				Ширина інтервалу	
		Лінійна регресія		Нелінійна регресія		Лінійна регресія	Нелінійна регресія
		LB	UB	LB	UB		
1	2	3	4	5	6	7	8
1	11,939	-28,5980	34,4233	11,7119	18,5890	63,0213	**6,8771**
2	28,368	23,7357	69,9925	36,0645	49,1785	46,2569	**13,1140**
3	35,835	-4,4920	118,3560	18,1140	42,9201	122,8480	**24,8060**
4	163,373	101,1250	158,6111	92,1227	153,4659	57,4861	**61,3433**
5	345,963	221,6178	285,0891	180,0186	301,2026	63,4713	121,1840
6	17,562	-26,8333	38,4551	6,9402	12,3604	65,2884	**5,4202**

Продовження таблиці 2.3

1	2	3	4	5	6	7	8
7	97,386	80,6795	139,3309	71,9112	117,9980	58,6514	**46,0868**
8	503,955	541,0664	699,7644	421,8847	908,1040	158,6981	486,2193
9	531,565	390,7115	516,3876	233,7437	465,8269	125,6761	232,0832
10	37,286	23,8388	78,2756	39,8945	59,2037	54,4368	**19,3092**
11	325,499	140,0995	219,9717	138,9190	255,4368	79,8723	116,5178
12	492,065	422,1456	538,8343	354,7973	662,5474	116,6887	307,7500
13	34,313	-2,3776	48,9923	19,4075	27,4814	51,3698	**8,0739**
14	361,592	154,8456	236,0140	152,7418	280,8667	81,1684	128,1248
15	95,203	109,2250	162,8449	90,3068	143,6785	53,6199	**53,3717**
16	34,469	-2,8030	126,6905	22,8489	52,9327	129,4936	**30,0838**
17	7,901	-17,3385	57,7715	8,9263	16,9451	75,1100	**8,0188**
18	21,214	14,2196	87,0469	24,1555	40,3725	72,8273	**16,2170**
19	30,459	11,6395	71,4183	29,9137	47,3757	59,7788	**17,4620**
20	4,534	-48,0418	20,6184	3,2844	6,2712	68,6602	**2,9868**
21	14,112	-39,4345	42,5508	14,3278	27,6087	81,9853	**13,2809**
22	19,466	4,7008	153,9301	11,4270	29,6104	149,2293	**18,1834**
23	84,992	108,8297	196,0085	101,6610	204,2007	87,1788	102,5397
24	106,324	85,6149	142,2023	84,7623	129,8966	56,5875	**45,1343**
25	58,782	118,7241	194,8909	84,2459	161,9095	76,1668	77,6636
26	2,067	-14,4274	39,6072	2,7831	5,5845	54,0346	**2,8014**
27	20,617	-46,8944	34,0890	9,1305	17,4069	80,9834	**8,2764**
28	101,499	98,7604	146,5477	90,9268	134,9573	47,7872	**44,0306**
29	25,053	37,7064	99,4962	47,8810	74,0409	61,7898	**26,1600**
30	13,175	-30,7257	28,9126	5,0780	8,8859	59,6383	**3,8079**
31	10,067	-21,2873	34,0190	8,2049	13,0552	55,3063	**4,8503**
32	15,115	-51,1909	24,1919	6,0992	11,2785	75,3828	**5,1793**
33	28,443	17,4569	128,9238	21,7407	46,7961	111,4669	**25,0554**
34	119,854	88,9811	164,5562	77,2556	146,5089	75,5750	**69,2533**

Дані таблиці 2.4 підтверджують, що у 79,41% випадків (27 із 34) нелінійна модель має менші інтервали передбачення. Для лінійної моделі зафіксовано випадки від'ємних значень як у довірчих інтервалах, так і в інтервалах передбачення, що є неприпустимим.

Таблиця 2.4 – Інтервали передбачення лінійної та нелінійної регресії

№	KLOC (Y)	Межі інтервалів передбачення				Ширина інтервалу	
		Лінійна регресія		Нелінійна регресія		Лінійна регресія	Нелінійна регресія
		LB	UB	LB	UB		
1	2	3	4	5	6	7	8
1	11,939	-123,52	129,35	5,930	36,711	252,8720	**30,7803**
2	28,368	-77,75	171,48	17,204	103,094	249,2233	**85,8907**
3	35,835	-80,06	193,92	10,448	74,410	273,9785	**63,9615**
4	163,373	4,09	255,64	47,484	297,737	251,5497	**250,2537**
5	345,963	126,86	379,85	92,935	583,441	252,9846	490,5058
6	17,562	-120,91	132,53	3,663	23,422	253,4465	**19,7593**
7	97,386	-15,90	235,91	36,863	230,186	251,8185	**193,3232**
8	503,955	474,51	766,32	236,654	1618,881	291,8178	1382,2263
9	531,565	315,92	591,18	128,031	850,451	275,2582	722,4199
10	37,286	-74,38	176,49	19,689	119,960	250,8704	**100,2713**
11	325,499	51,24	308,83	74,113	478,796	257,5891	404,6828
12	492,065	344,85	616,13	190,267	1235,478	271,2726	1045,2112
13	34,313	-101,80	148,42	9,401	56,730	250,2228	**47,3286**
14	361,592	66,43	324,43	81,489	526,454	257,9940	444,9650
15	95,203	10,69	261,38	45,769	283,492	250,6944	**237,7226**
16	34,469	-76,57	200,45	13,096	92,355	277,0220	**79,2591**
17	7,901	-107,86	148,29	4,813	31,427	256,1525	**26,6137**
18	21,214	-77,11	178,38	12,465	78,234	255,4925	**65,7681**
19	30,459	-84,51	167,57	15,135	93,637	252,0835	**78,5016**
20	4,534	-140,88	113,46	1,774	11,609	254,3360	**9,8342**
21	14,112	-127,57	130,68	7,763	50,954	258,2521	**43,1902**
22	19,466	-64,07	222,70	6,753	50,104	286,7786	**43,3507**
23	84,992	22,45	282,39	55,823	371,876	259,9475	316,0531
24	106,324	-11,76	239,58	42,356	259,951	251,3458	**217,5950**
25	58,782	28,58	285,04	45,609	299,071	256,4643	**253,4618**
26	2,067	-112,80	137,98	1,528	10,173	250,7834	**8,6456**
27	20,617	-135,37	122,57	4,930	32,238	257,9358	**27,3079**
28	101,499	-2,10	247,41	44,878	273,438	249,5119	**228,5602**
29	25,053	-57,68	194,89	24,008	147,663	252,5679	**123,6550**
30	13,175	-126,93	125,12	2,664	16,941	252,0502	**14,2776**
31	10,067	-119,16	131,90	4,159	25,758	251,0605	**21,5998**
32	15,115	-141,62	114,62	3,260	21,101	256,2326	**17,8404**
33	28,443	-61,34	207,72	12,195	83,424	269,0677	**71,2287**
34	119,854	-1,38	254,91	41,642	271,809	256,2892	**230,1668**

Таблиця 2.5 містить результати оцінки якості лінійної та нелінійної моделей для оцінювання розміру програмних застосунків на C#.

Таблиця 2.5 – Оцінки якості побудованих моделей

Оцінки якості моделі	Модель	
	Лінійна	Нелінійна
R^2	0,867	0,851
$MMRE$	0,989	0,346
$PRED(0{,}25)$	0,235	0,441

Отримані результати свідчать про незадовільну якість обох моделей, втім, для нелінійної моделі значення оцінки $MMRE$ є відчутно кращім.

2.3 Висновки до розділу 2

У розділі представлено побудову нелінійної регресійної моделі для оцінювання розміру програмних застосунків на C# із застосуванням нормалізуючого перетворення log10.

Основні результати:

– Виконано пошук проектів на платформі GitHub, за допомогою Visual Studio Metrics було зібрано метрики діаграми класів, необхідні для аналізу.

– Проведено тест на мультиколінеарність, який показав її відсутність. Аналіз квадрата відстані Махаланобіса підтвердив ненормальність розподілу даних.

– Здійснено видалення викидів із набору даних із використанням нормалізуючих перетворень і відстані Махаланобіса. У результаті з 40 проектів було видалено два викиди.

– Побудовано лінійну та нелінійну регресійні моделі для очищених даних. Лінійна модель використовувалася для порівняння, результати показали низьку якість обох моделей.

– Побудовано довірчі та передбачувані інтервали для обох моделей. Нелінійна модель продемонструвала менші ширини цих інтервалів.

– Проведено порівняння лінійної моделі з нелінійною, нелінійна має значно краще значення коефіцієнту $MMRE$.

3 СТВОРЕННЯ ПРОЕКТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ПРОГРАМНИХ ЗАСТОСУНКІВ НА C#

3.1 Формулювання завдання на розробку проекту ПЗ для оцінювання розміру програмних застосунків на C#

Одним із ключових етапів при розробці програмного забезпечення є вибір мови програмування, яка найкраще відповідатиме поставленим завданням. У рамках цієї роботи важливою задачею є створення програми для оцінки розміру програмних застосунків на C#. Враховуючи специфіку проєкту та вимоги до функціональності, Python було обрано як основний інструмент реалізації. У цьому розділі наведено обґрунтування цього вибору.

Python – це мова програмування високого рівня з динамічною типізацією, яка поєднує простоту використання та потужний функціонал. Вона підтримує широкий спектр парадигм програмування, включно з об’єктно-орієнтованим, процедурним та функціональним програмуванням, що робить її універсальним інструментом для розробки різноманітного програмного забезпечення. Крім того, Python має широкий набір бібліотек і фреймворків, що спрощує реалізацію складних завдань без необхідності розробляти базові алгоритми з нуля.

Однією з ключових переваг Python є широкий набір бібліотек, які дозволяють прискорити розробку і підвищити якість кінцевого продукту. У рамках цієї роботи особливу цінність мають такі бібліотеки:

– NumPy – це бібліотека, яка забезпечує підтримку багатовимірних масивів і матриць, а також широкий набір математичних функцій для роботи з ними. Вона є стандартом де-факто для наукових обчислень у Python і дозволяє легко реалізовувати складні математичні розрахунки.

– SciPy надає набір модулів для наукових і інженерних розрахунків, включно з обробкою сигналів, оптимізацією, інтерполяцією та чисельним інтегруванням. Це робить її ідеальним вибором для реалізації завдань аналізу та обробки даних.

– Matplotlib – це бібліотека для створення статичних, інтерактивних і анімаційних графіків. Вона забезпечує можливості для візуалізації даних, що особливо важливо для аналізу результатів роботи програми та створення наочних звітів.

Для реалізації зручного та інтуїтивно зрозумілого користувацького інтерфейсу планується використати бібліотеку PyQt. PyQt надає потужні інструменти для створення графічних інтерфейсів, які легко інтегруються з функціоналом, реалізованим на Python. Ця бібліотека дозволяє розробникам створювати інтерфейси будь-якої складності, від простих форм до багатокомпонентних застосунків. Крім того, PyQt є кросплатформенною бібліотекою, що забезпечує можливість запуску програми на різних операційних системах без суттєвих змін коду.

Python є кросплатформенною мовою, що означає можливість її використання на операційних системах Windows, macOS та Linux. Це особливо важливо для забезпечення доступності програми для широкого кола користувачів. Крім того, екосистема Python включає інструменти для спрощення упаковки застосунків і створення виконуваних файлів для різних платформ.

Python славиться своєю лаконічністю та читабельністю коду, що спрощує процес розробки і подальшого супроводу програмного забезпечення. Завдяки інтуїтивно зрозумілому синтаксису розробники можуть швидше реалізовувати функціонал, зосереджуючись на логіці програми, а не на технічних деталях реалізації.

Python має одну з найбільших спільнот розробників у світі. Це означає, що для більшості завдань уже існують готові рішення, доступні у вигляді бібліотек, документації або обговорень на форумах. Крім того, наявність активної спільноти спрощує процес пошуку відповідей на виникаючі питання та вирішення можливих проблем у процесі розробки.

Враховуючи всі перелічені переваги, Python є оптимальним вибором для реалізації програми в рамках цієї роботи. Наявність спеціалізованих бібліотек, таких як NumPy, SciPy та Matplotlib, дозволяє ефективно вирішувати завдання

аналізу даних і візуалізації. Можливості PyQt забезпечують зручність створення графічного інтерфейсу. Кросплатформенність і підтримка широкої спільноти роблять Python інструментом, який відповідає всім вимогам проєкту і сприяє успішному завершенню розробки програмного забезпечення.

Розробка програмного забезпечення здійснюватиметься за технічним завданням, наведеним у Додатку А.

3.2 Створення ескізного проекту ПЗ для оцінювання розміру програмних застосунків на C#

Ескізний проєкт передбачає концептуальне моделювання створюваної системи, для чого буде побудовано діаграму варіантів використання та діаграму активності.

3.2.1 Створення моделі прецедентів ПЗ для оцінювання розміру програмних застосунків на C#

Діаграма варіантів використання є інструментом для моделювання поведінки системи з точки зору її користувачів. Вона дозволяє візуалізувати ключові сценарії взаємодії зовнішніх учасників (акторів) із системою, допомагаючи розробникам і зацікавленим сторонам зрозуміти основні функції продукту.

На діаграмі акторами можуть бути:

– Користувачі (наприклад, адміністратор або звичайний клієнт).

– Інші системи, що обмінюються даними. Варіанти використання зображуються як овали, а взаємодії – стрілками, які демонструють, які сценарії доступні для кожного актора.

Таблиця 3.1 – Виявлені варіанти використання

Основні актори	Найменування	Формулювання
Користувач	Підвантажити файл з метриками проектів на C#	Підвантажує файл з метриками проектів на C#
Користувач	Ввести кількість класів	Вводить число класів у застосунку на C#
Користувач	Ввести зв’язність класів	Вводить зв’язність об’єктів застосунків на C#
Користувач	Ввести глибину успадкування	Вказує глибину успадкування проектів на C#
Користувач	Ввести довірчу ймовірність	Встановлює рівень довірчої ймовірності
Користувач	Провести нормалізацію даних	Отримує нормалізовані значення за допомогою логарифмічного перетворення
Користувач	Збудувати регресійну модель	Формує модель і надає результати її роботи
Користувач	Виконати розрахунки щодо розміру	Розраховує розмір системи, інтервал передбачення та довірчий інтервал
Користувач	Оцінити інтервал передбачення	Визначає інтервал передбачення застосунків на C#
Користувач	Оцінити довірчий інтервал	Визначає довірчий інтервал застосунків на C#

Рисунок 3.1 показує діаграму варіантів використання

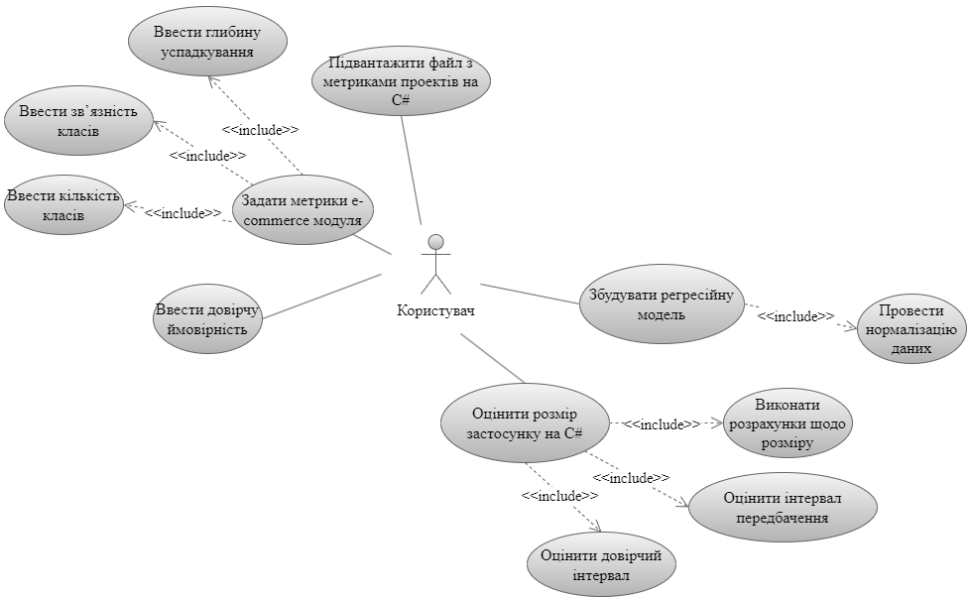

Рисунок 3.1 – Діаграма варіантів використання

3.2.2 Створення сценарію виконання ПЗ для оцінювання розміру програмних застосунків на C# за допомогою діаграми діяльності

Діаграма діяльності – це один із ключових інструментів у моделюванні систем, що відображає порядок виконання дій і залежності між ними. Вона використовується для того, щоб: Продемонструвати, як працює система або процес. Виявити можливі вузькі місця чи неефективності. Спланувати оптимальну послідовність дій у рамках системи. Графічна структура діаграми містить вузли (що позначають дії або стани) та стрілки (що відображають потік управління чи передачу даних). Її особливістю є можливість опису паралельних процесів, що особливо важливо для складних систем. Вона активно використовується не тільки у сфері розробки програмного забезпечення, але й у бізнес-аналітиці, для документування робочих процесів, побудови алгоритмів і підготовки до тестування програмних продуктів. Така діаграма є містком між технічними спеціалістами й управлінцями, забезпечуючи краще розуміння складних процесів.

На рисунку 3.2. показую діаграму діяльності розроблюваного програмного застосунку.

Рисунок 3.2 – Діаграма діяльності

3.3 Розробка технічного проекту ПЗ для оцінювання розміру програмних застосунків на C#

3.3.1 Модель статичної структури ПЗ для оцінювання розміру C#-застосунків, представлена у вигляді діаграми класів

Діаграма класів – це статична діаграма в UML, яка використовується для опису структури програмного забезпечення через взаємозв’язки між його класами.

Вона демонструє, як класи системи співвідносяться один з одним і які функціональні можливості вони мають.

Діаграма класів складається з таких основних елементів:

– Класів, що є фундаментальними одиницями об’єктно-орієнтованого програмування.

– Атрибутів, які визначають характеристики класів.

– Методів, що відображають їхню поведінку.

– Зв’язків (асоціації, залежності, агрегації, композиції), які показують, як класи взаємодіють між собою.

– Ієрархії успадкування, які описують відносини між базовими та похідними класами.

Цей вид діаграми дозволяє ефективно спроєктувати об’єктно-орієнтовану систему, забезпечуючи її модульність, масштабованість і зрозумілість. Вона є важливим інструментом для команд розробників, архітекторів і аналітиків, допомагаючи створювати чітку документацію системи та планувати її реалізацію.

На рисунку 3.3. зображено структуру розроблюваного програмного застосунку у вигляді діаграми класів.

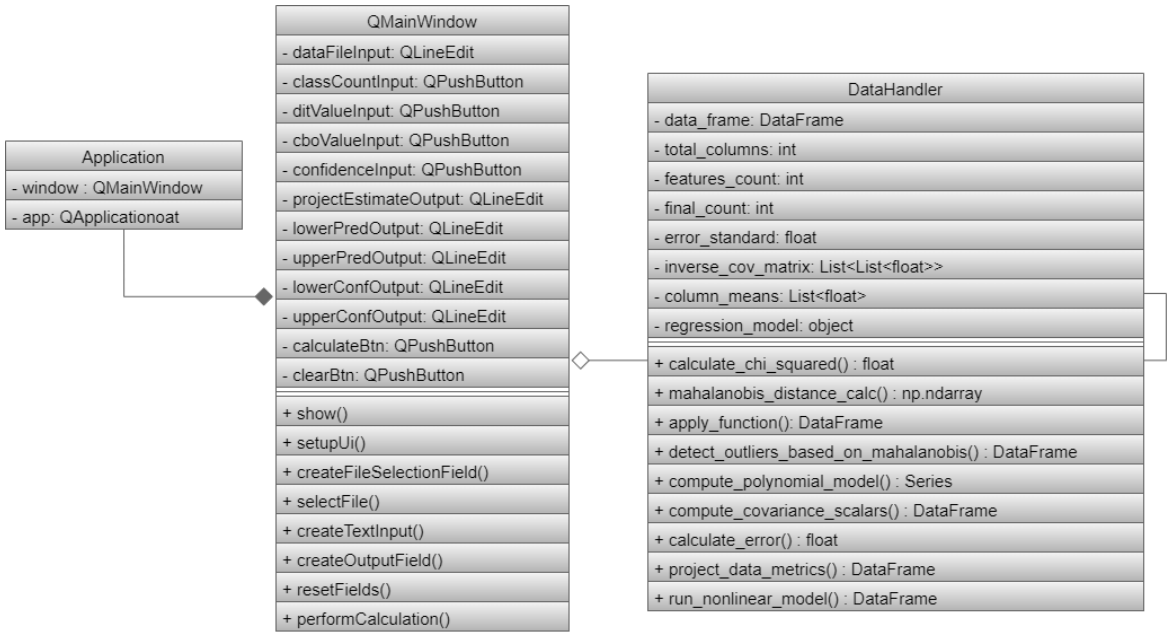

Рисунок 3.3 – Діаграма класів

3.3.2 Специфікації класів ПЗ для оцінювання розміру програмних застосунків на C#

– Клас QMainWindow – відповідає за реалізацію головного вікна програми.

– Клас QtWidget – базовий елемент бібліотеки Qt, що надає функціонал для створення графічного інтерфейсу.

– Клас QLineEdit – використовується для розробки текстових полів у GUI.

– Клас QValidator – забезпечує перевірку коректності даних під час їх введення.

– Клас Application – основний клас, який управляє функціонуванням програми.

– Клас DataHanlder – використовується для побудови моделей регресії, оцінювання розміру застосунків і отримання інтервальних оцінок.

3.3.3 Динамічна модель ПЗ для оцінювання розміру програмних застосунків на C# у вигляді діаграм послідовностей

Діаграма послідовностей — це інструмент UML, який моделює процес обміну повідомленнями між об’єктами системи у визначеній послідовності. Вона наочно показує, як взаємодіють різні компоненти системи, які виклики або повідомлення відправляються і в якій послідовності.

Ключові елементи діаграми:

– Актори – зовнішні суб’єкти, що ініціюють або отримують повідомлення.

– Об’єкти – учасники системи, які виконують певні дії.

– Повідомлення – взаємодії у вигляді викликів методів або відповідей.

– Життєвий цикл об’єкта – період, протягом якого об’єкт існує в системі.

Цей тип діаграми допомагає деталізувати сценарії використання, аналізувати виконання бізнес-логіки та забезпечити чітке уявлення про поведінку системи в реальному часі.

На рисунку 3.4. показано алгоритм варіанту використання «Ввести дані застосунку на C#» у вигляді діаграми послідовності

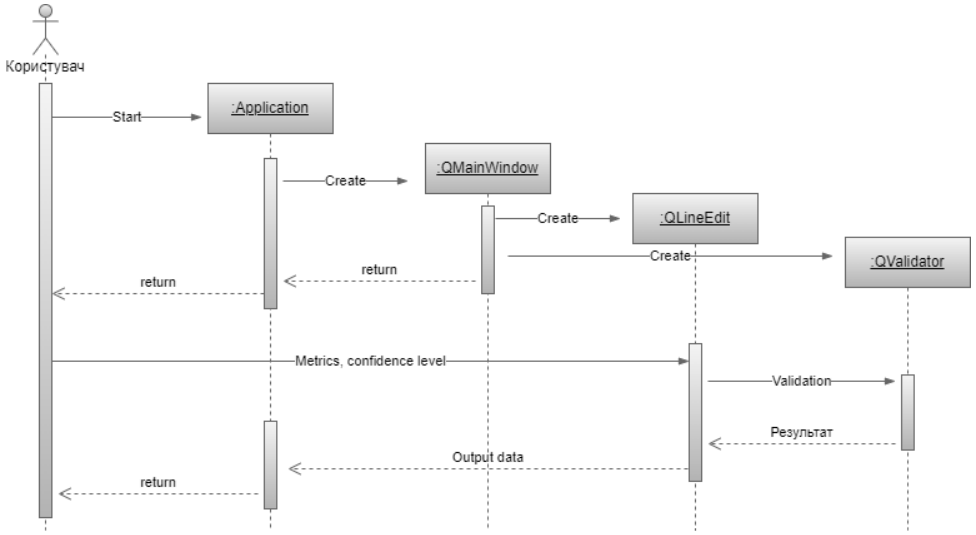

Рисунок 3.4 – Зображення алгоритму виконання прецеденту «Ввести дані застосунку на C#»

На рисунку 3.5. показано алгоритм варіанту використання «Виконати розрахунки щодо розміру» у вигляді діаграми послідовності

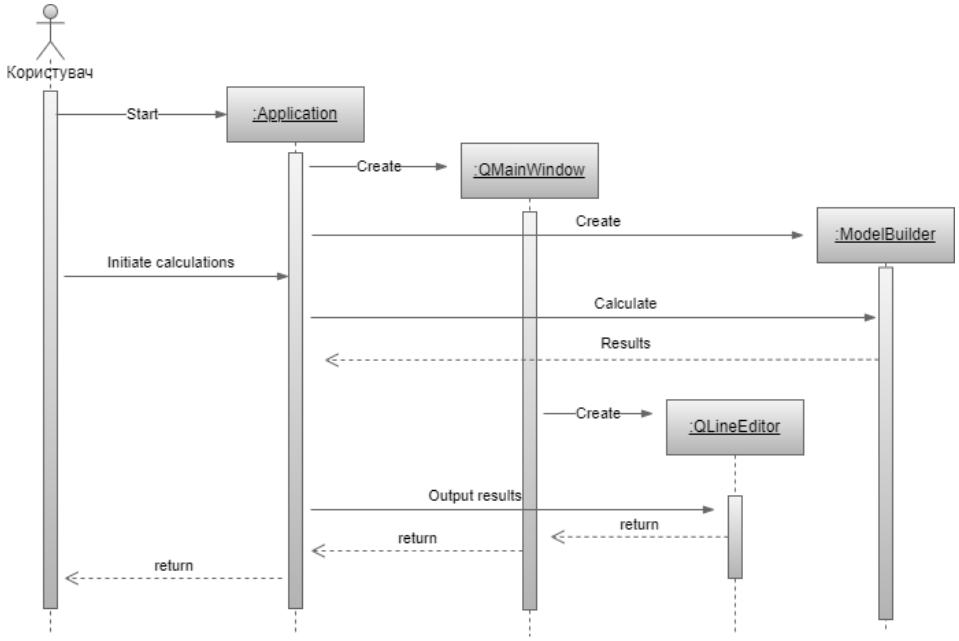

Рисунок 3.5 – Зображення алгоритму виконання прецеденту «Виконати розрахунки щодо розміру»

Розробка робочого проекту ПЗ для оцінювання розміру програмних застосунків на C#

3.3.4 Кодування ПЗ для оцінювання розміру програмних застосунків на C#

У межах проєкту створено програму для аналізу та оцінювання розміру програмних застосунків на C# у тисячах рядків коду. Було проведено тестування програми та підтверджено її функціональність.

Розробка виконана мовою Python, а для графічного інтерфейсу використано бібліотеку Qt. Повний текст програмного коду подано в додатку Г.

3.3.5 Тестування ПЗ для оцінювання розміру програмних застосунків на C#

Еквівалентне розбиття — це метод тестування чорного ящика, спрямований на оптимізацію процесу перевірки за рахунок виділення класів даних, які обробляються однаковим чином. Замість тестування кожного можливого значення, перевіряються лише представники кожного класу.

Основні етапи методу:

– Визначення діапазонів вхідних даних.

– Поділ даних на класи (еквівалентні групи).

– Вибір тестових значень з кожного класу.

Ця техніка економить ресурси та дозволяє уникнути надлишкового тестування.

Еквівалентне розбиття – це техніка тестування програмного забезпечення, яка передбачає розділення набору вхідних даних на класи еквівалентності. Усі значення в одному класі вважаються еквівалентними, оскільки вони повинні оброблятися програмою однаково. Ця методика дозволяє скоротити кількість тестових випадків, зберігаючи їх ефективність.

При тестуванні варіанту використання «Ввести кількість класів» виявлено класи еквівалентності, що розділені на правильні (натуральні числа) та неправильні (дійсні, від’ємні, текст, відсутність значення).

Таблиця 3.2 – Всі класи еквівалентності для варіанту використання «Ввести кількість класів»

Вхідні данні	*Правильні класи*	*Неправильні класи*
Кількість класів	Натуральне число (1)	Дійсне число (2). Від'ємне число (3). Не число (4) Без даних (5).

Таблиця 3.3 – Проведення тестування з правильними класами еквівалентності

№ тесту	Покритий клас	Вхідні дані	Вихідні данні	Результат тестування
1	1	100	Продовження роботи програми	Пройдено

Таблиця 3.4 – Проведення тестування з неправильними класами еквівалетності

№ тесту	Покритий клас	Вхідні дані	Вихідні данні	Результат тестування
2	2	201,11	Помилка на GUI	Пройдено
3	3	-100,3	Помилка на GUI	Пройдено
4	4	Some text 123%	Помилка на GUI	Пройдено
5	5	-	Помилка на GUI	Пройдено

При тестуванні варіанту використання «Ввести зв'язність класів» виявлено класи еквівалентності, що розділені на правильні (дійсні числа числа) та неправильні (від'ємні, текст, відсутність значення).

Таблиця 3.5 – Всі класи еквівалентності для варіанту використання «Ввести зв'язність класів»

Вхідні данні	*Правильні класи*	*Неправильні класи*
Зв'язність між класами (глибина дерева успадкування)	Дійсне додатне число (1)	Від'ємне число (2). Не число (3) Без даних (4).

Таблиця 3.6 – Проведення тестування з правильними класами еквівалетності

№ тесту	Покритий клас	Вхідні дані	Вихідні данні	Результат тестування
1	1	5,90	Подальша робота програми	Пройдено

Таблиця 3.7 – Проведення тестування з неправильними класами еквівалентності

№ тесту	Покритий клас	Вхідні дані	Вихідні данні	Результат тестування
2	2	100,3	Помилка на GUI	Пройдено
3	3	Some text 123%	Помилка на GUI	Пройдено
4	4	-	Помилка на GUI	Пройдено

Введення глибини дерева наслідування тестується аналогічно введенню методів, а довірча ймовірність перевіряється в межах від 0 до 1. Усі результати підтверджують правильну роботу програми.

3.3.6 Випробування ПЗ для оцінювання розміру програмних застосунків на C#

Випробування програмного забезпечення для оцінювання розміру програмних застосунків на C# проведено за програмою та методикою, описаною в Додатку Б. Було протестовано ключові функції: введення метрик, нормалізацію даних, побудову нелінійної регресійної моделі та оцінку розміру застосунку разом із довірчими й прогнозними інтервалами. Результати демонструють стабільну роботу програми, підтверджуючи її функціональність. На рисунках 3.6 і 3.7 наведено приклади результатів оцінювання для вибраного застосунку.

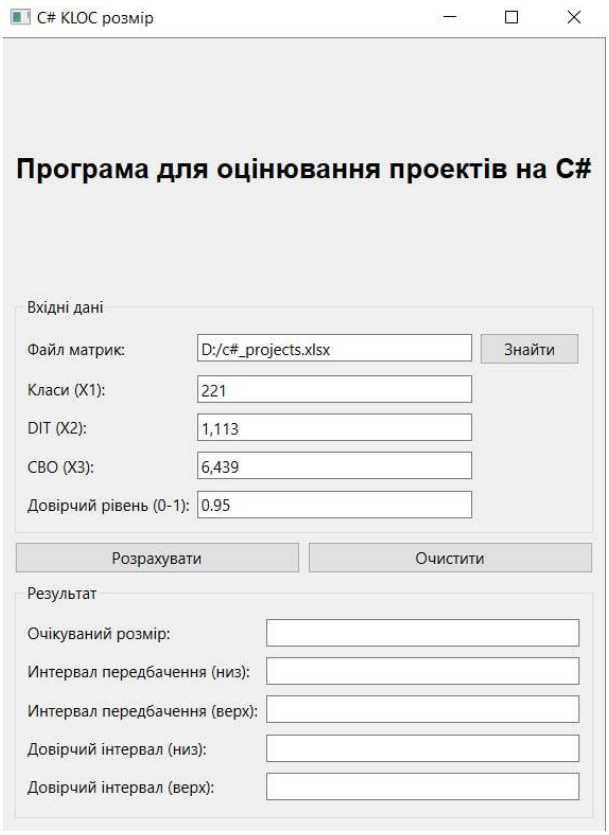

Рисунок 3.6 – Вікно програми перед оцінюванням

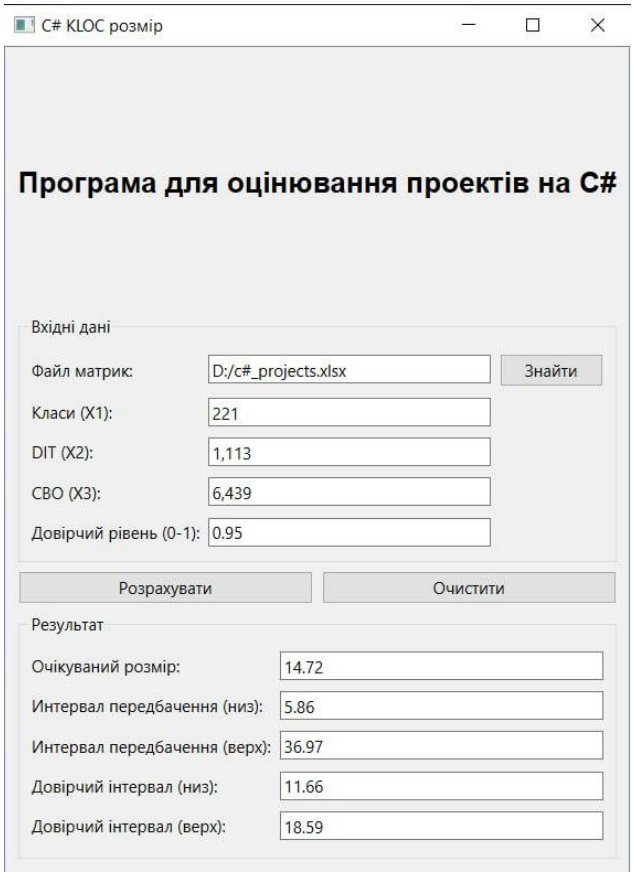

Рисунок 3.7 – Результат оцінювання програмного застосунку на C#

Отримані результати під час випробувань показують правильну роботу програмного забезпечення, тому вимоги з технічного завдання виконані.

3.4 Висновки до розділу 3

Розділ 3 описує розробку програмного забезпечення для оцінювання розміру застосунків на C#. Виконано етапи від постановки задачі до випробувань програми. Створено моделі у форматі UML, що охоплюють діаграми класів, діяльності та послідовностей. Застосування Python та PyQt забезпечило створення кросплатформного інтерфейсу. Тестування показало коректну роботу програми: введення метрик, нормалізація даних, побудова регресійної моделі та обчислення інтервалів виконані без помилок. Система підтвердила відповідність вимогам і готовність до практичного використання.

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВІД РОЗРОБКИ І ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ ПРОГРАМНИХ ЗАСТОСУНКІВ НА C#

Вступ

Економічну доцільність розробки програмного продукту можна обґрунтувати, якщо провести розрахунок його собівартості та очікуваного прибутку. Головним показником, що демонструє доцільність витрат, є річний економічний ефект.

Підтвердити доцільність витрат на розробку програмного забезпечення можна шляхом аналізу собівартості його створення та прибутків від реалізації. Основним критерієм тут виступає річний економічний ефект.

4.1 Розрахунок витрат на створення й експлуатацію програмного забезпечення

Витрати на розробку програмного забезпечення складаються з витрат на заробітну плату розробника, на амортизацію та експлуатацію ЕОМ, на якій виконується розробка, на засоби розробки та витрати на матеріали і комплектуючі. Розробка програмного забезпечення виконується спеціалістом, місячний оклад якого складає 19000 грн. Додаткова заробітна плата складає 20% від основної. Виходячи з цього, основна і додаткова заробітна плата розроблювача системи 22800 грн./міс , а вартість сучасної ПЗВМ складає 26000 грн. Вартість кіловат-години електроенергії дорівнює 4,32 грн. Витрати на допоміжні матеріали наведені в таблиці 4.1

Таблиця 4.1 – Витрати на допоміжні матеріали

№	Пункти витрат	Сума, грн.
1	Папір для принтера	300
2	Картридж та тонер для принтера	300
3	Використання флеш-накопичувача	300
4	Непередбачувані витрати	1500
	Разом	2400

Вартість розробки системи розраховується за формулою:

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

де Т – тривалість розробки у місяцях

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

$З_{зс}$ – відрахування на соціальні заходи (38% від основної і додаткової заробітної плати), грн.;

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

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

$З_{м}$ – витрати на допоміжні матеріали, грн.

$З_{е}$ при витратах в 0,5 кВт з тривалістю роботи на місяць 22*8 = 176 годин та вартості електроенергії 4,32 грн. становить

$$З_{е} = 176 * 4{,}32 * 0{,}5 = 380{,}16 \text{ грн.} \quad (4.1)$$

Представимо всі поточні витрати на розробку програмного забезпечення в таблиці 4.2

Таблиця 4.2 – Витрати на розробку програмного забезпечення

№	Пункти витрат	Одиниця	Кількість
1	Тривалість розробки (Т)	Міс.	2
2	Основна та додаткова заробітна плата (Ззп)	Грн.	22800
3	Відрахування на соціальні витрати (Ззс)	Грн.	8664
4	Загальногосподарські витрати (Ззг)	Грн.	2280
5	Витрати на допоміжні матеріали (Зм)	Грн.	2400
6	Витрати на електроенергію *(Зе)	Грн.	380,16

За формулою (4.1) виконаємо розрахунок вартості програми:

$$С_{пр} = (22800 + 8664 + 2280 + 380,16) * 2 + 2400 = 70648,32 \text{ грн.}$$

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

$$А_{об} = 26000 * 0,6 = 15600 \text{ грн.}$$

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

$$В_{м} = 26000 * 0,05 = 1300 \text{грн.}$$

Річний обсяг робіт ПК у годинах можна визначити як

$$Ф_{м} = 264,5 * Т_{з}, \quad (4.2)$$

де Тз – середнє навантаження на устаткування в місяць (6 годин), 264,5 – середня кількість робочих днів на рік.

Отже, річний обсяг роботи ПК складає:

$$Ф_{м} = 264,5 * 6 = 1587 \text{ годин}$$

Витрати на електроенергію Зе при 1578 годинах роботи устаткування на рік становлять:

$$З_{е} = 1587 * 4,32 = 6555,84$$

Експлуатаційні витрати на ПК на рік становлять:

$$З_{зр} = 15600 + 1300 + 6555,84 = 23455,84$$

Таким чином витрати на розробку та експлуатацію програмного забезпечення в перший рік становлять:

$$З_{р} = 70648,32 + 23455,84 = 94104,16$$

4.2 Економічна ефективність розробки та впровадження програмного забезпечення

Визначити пряму економічну ефективність можна, ґрунтуючись на тому, що впровадження програмного забезпечення вивільняє одного працівника (за експертною оцінкою фахівців підприємства). Зарплата одного працівника в рік складає

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

Річний економічний ефект розраховується по формулі:

$$E_{рік} = \Delta C_n - E_n * k, \quad (4.3)$$

де ΔC_n – кошти, що буде вивільнено при впровадженні програмного забезпечення (228000) без експлуатаційних витрат (23455,84), тобто, 204544,16 грн.;

E_n – коефіцієнт ефективності (такий же як й коефіцієнт амортизації – 0,6)

k – одноразові витрати на впровадження програмного забезпечення (70648,32 грн.).

$$E_{рік} = 204544{,}16 - 0{,}6 * 70648{,}32 = 218149{,}16 - 28362{,}19 = 162155{,}94$$

Термін, за який програмне забезпечення окупиться наступний:

$$T = \frac{k}{\Delta C_n} = \frac{70648{,}32}{204544{,}16} = 0{,}34 \text{ року}$$

Отже термін, за який програмне забезпечення окупиться 4 місяці.

4.3 Висновки до розділу 4

У цьому розділі проведено розрахунки, спрямовані на створення та інтеграцію програмного забезпечення для визначення розміру програмних застосунків на C#. Також було оцінено економічну ефективність розробки та визначено термін її окупності. Згідно з розрахунками, окупність програмного забезпечення становить 4 місяці впродовж першого року використання. Це свідчить про високу рентабельність проєкту.

5 ОХОРОНА ПРАЦІ

Охорона праці – це система, яка включає правові, соціальні, економічні, технічні та гігієнічні заходи, спрямовані на забезпечення безпечних і комфортних умов праці, збереження здоров'я і працездатності працівників.

Складові цієї системи:

– аналіз шкідливих факторів, що впливають на працівників;

– впровадження заходів для нейтралізації ризиків і створення належних умов праці;

– виконання норм техніки безпеки;

– захист окремих категорій працівників, таких як жінки, неповнолітні, люди зі зниженою працездатністю;

– спеціальні правила для роботи в шкідливих умовах.

Рівень безпеки праці залежить від компетенції керівників і впровадження стандартів охорони праці, що спрямовані на зниження рівня травматизму і професійних захворювань.

Закон України "Про охорону праці" гарантує право на безпеку праці, регулюючи відносини між працівниками та роботодавцями і встановлюючи єдиний підхід до організації заходів охорони праці в країні.

Покращення умов праці не лише позитивно впливає на економічні показники підприємства, але й має вагоме соціальне значення: підвищується рівень здоров’я працівників, їхнє задоволення роботою, зміцнюється дисципліна та активність.

Екологічна складова охорони праці включає зменшення забруднень, боротьбу з шумом і випромінюваннями, що водночас покращує стан довкілля.

5.1 Аналіз шкідливих та небезпечних факторів в офісі фірми

Співробітники відділу піддаються впливу таких небезпечних факторів, як:

– недостатня яскравість освітлення робочих зон;

– шум і вібрація, створювані офісною технікою;

– пожежонебезпечність через велику кількість паперових носіїв;

– високий рівень електромагнітних і електростатичних випромінювань від комп’ютерів;

– невідповідні умови мікроклімату, включаючи температуру, вологість і рух повітря.

Освітлення в приміщенні поділяється на три типи: природне, комбіноване та штучне. Вдень використовують природне освітлення через вікна, ввечері – штучне, а в похмурий час – комбіноване.

Джерела шуму включають вентилятори та комп’ютери. Для його зниження застосовуються звукопоглинальні матеріали. Рівень вібрації від техніки знаходиться в межах допустимого.

Меблі у відділі можуть сприяти швидкому поширенню вогню у разі пожежі. Електроприлади підключені через заземлені розетки, що мінімізує ризик електротравм.

Електромагнітне випромінювання комп’ютерів негативно впливає на продуктивність, а пил навколо моніторів може викликати подразнення шкіри.

Мікроклімат у приміщенні підтримується на належному рівні завдяки системам кондиціонування і вентиляції.

5.2 Заходи щодо зменшення впливу шкідливих факторів роботи за персональним комп’ютером

У процесі роботи співробітники відділу піддаються впливу небезпечних і шкідливих факторів, серед яких:

Для зменшення впливу небезпечних і шкідливих факторів у приміщенні необхідно здійснити ряд заходів.

– недостатнє освітлення робочого місця;

– підвищені рівні шуму і вібрації, спричинені функціонуванням вентиляторів і принтерів;

– висока ймовірність пожежі через велику кількість паперових матеріалів;

– надмірна напруга в електромережі, яка може спричинити ураження струмом;

– вплив неіонізуючого електромагнітного випромінювання та електростатичних полів, що виникають під час роботи комп’ютерів;

– невідповідні параметри мікроклімату, зокрема температура, вологість і швидкість руху повітря.

Робота співробітників вимагає значного зорового напруження, що робить правильне проектування та розташування освітлення важливим елементом організації робочого місця. Освітлення у приміщенні залежить від часу доби та погодних умов і поділяється на природне, штучне та комбіноване.

У денний час за ясної погоди використовується природне світло через вікна, що сприяє комфортній роботі. Увечері чи за несприятливих умов додається комбіноване освітлення, зокрема настільні лампи. Увечері взимку застосовують виключно штучне освітлення за допомогою ламп розжарювання. Рекомендована яскравість освітлення у відділі становить 750 лк.

Одним із основних шкідливих чинників є шум, джерелами якого виступають вентилятори та комп’ютери. Для зменшення його рівня у приміщенні використовуються звукопоглинаючі матеріали, такі як спеціальні перфоровані плити й мінеральна вата.

Рівень вібрації при роботі з електронно-обчислювальними машинами не перевищує допустимих норм. У відділі встановлено два комп’ютери та струменевий принтер, які забезпечують дотримання встановлених стандартів.

Окрему небезпеку становлять легкозаймисті меблі, які у разі пожежі можуть сприяти її швидкому поширенню. Електричний струм, проходячи через тіло людини, викликає теплові, хімічні та біологічні ефекти, що можуть призвести до травм. Усі електроприлади, такі як холодильник, чайник і касовий апарат, підключені через заземлені трьохконтактні вилки.

Робота комп’ютерів супроводжується електромагнітними та електростатичними випромінюваннями, які можуть підвищувати концентрацію

пилу навколо моніторів. Це може спричинити подразнення шкіри та зниження загальної продуктивності праці.

Параметри мікроклімату в приміщенні, такі як температура, вологість і швидкість руху повітря, впливають на самопочуття співробітників і ефективність роботи обладнання. У відділі забезпечені системи опалення та кондиціонування, які відповідають встановленим нормам і сприяють підтриманню продуктивності.

6 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Охорона природи передбачає комплекс дій, спрямованих на зменшення негативного впливу діяльності людини на екосистеми. Загальне управління у цій галузі здійснюють Кабінет Міністрів України та органи місцевої влади. Міністерство охорони природи та його регіональні підрозділи є основними структурами, що відповідають за реалізацію екологічної політики.

До основних законів, що регулюють природоохоронну діяльність, належать: «Про охорону навколишнього природного середовища» (1991 р.), «Про охорону атмосферного повітря» (1992 р.), «Про природно-заповідний фонд України» (1992 р.) та інші.

Правові, економічні та технічні засоби охорони довкілля працюють на забезпечення екологічного балансу. Вони спрямовані на обмеження забруднення, підтримку природних комплексів і ресурсозбереження.

Проблема забруднення атмосфери стає дедалі серйознішою. Газові викиди сприяють парниковому ефекту та руйнуванню озонового шару, що призводить до значних кліматичних змін. Танення льодовиків і підвищення рівня океанів ставлять під загрозу екосистеми багатьох регіонів.

Техногенне навантаження на атмосферу зростає, і щороку викидаються сотні мільйонів тонн шкідливих речовин. Серед них найбільш небезпечні – оксиди сірки, азоту, галогени та формальдегіди. Зменшення їх обсягів є важливим завданням.

6.1 Забруднення навколишнього середовища

Офісна комп’ютерна компанія, яка спеціалізується на створенні програмного забезпечення, хоча й не займається промисловим виробництвом, все ж створює певні екологічні ризики.

Джерелами забруднення можуть бути:

– резервні дизельні електростанції;

– системи опалення на твердому паливі;

– побутові відходи;

– електромагнітне випромінювання;

– списане обладнання.

Для уникнення перебоїв у роботі компанія використовує дизельну електростанцію, яка є джерелом викидів і шуму. У холодний період приміщення опалюються за допомогою пелетних котлів. Незважаючи на нижчий рівень викидів, ніж у традиційного палива, цей спосіб все ще має вплив на атмосферу.

У процесі діяльності накопичуються різноманітні побутові відходи, які вимагають правильної утилізації. Вони включають як рідкі стічні води, так і тверді залишки – папір, пластик, скло, харчові продукти.

Високі темпи розвитку техніки зумовлюють зростання кількості застарілих комп’ютерів та іншого обладнання, яке містить небезпечні речовини. Неправильне поводження з такою технікою може призвести до забруднення ґрунту і води.

Крім того, значна кількість працюючої техніки в офісі створює електромагнітне випромінювання, яке негативно впливає на самопочуття співробітників, спричиняючи головний біль і втому.

6.2 Розробка заходів щодо зменшення забруднення

Захист природного середовища, його раціональне використання та забезпечення екологічної безпеки є важливими складовими сталого розвитку України. Відповідно до Закону України «Про охорону навколишнього природного середовища» та інших нормативних актів, держава регулює екологічні відносини через земельне, водне, лісове законодавство, а також закони про надра, охорону атмосферного повітря і біорізноманіття.

Щоб зменшити шкоду довкіллю, необхідно вживати такі заходи:

– Очищення димових газів – електрофільтри та мокрі золоуловлювачі ефективно очищують викиди від золи, використовуючи відцентрову силу та водяні плівки. Їх ефективність сягає 90%.

– Оптимізація джерел енергії – перехід з деревних пелетів на природний газ забезпечує нижчий рівень шкідливих викидів, хоча економічні аспекти цього рішення мають бути враховані.

– Управління побутовими відходами – сортування, переробка вторинної сировини та повторне використання є важливими етапами зменшення обсягів сміття.

– Захист від електромагнітного випромінювання – регулярні перерви в роботі з технікою, її відключення у неробочий час і технічний контроль приладів допомагають мінімізувати шкідливий вплив.

– Екологічна утилізація техніки – переробка старих електронних пристроїв дозволяє ефективно повторно використовувати ресурси, зменшуючи кількість відходів на сміттєзвалищах.

ВИСНОВКИ

За результатами кваліфікаційної роботи досягнуто поставлену мету щодо розробки нелінійної регресійної моделі для оцінювання розміру програмних застосунків на C#.

Основні результати:

– Проведено аналіз існуючих підходів та моделей для оцінювання розміру програмного забезпечення в цілому, та для мови C# зокрема. Виявлено, що найбільш ефективними є нелінійні моделі, побудовані на основі метрик із діаграм класів. Обґрунтовано необхідність створення такої моделі для програмних застосунків на C#.

– Виконано збір даних із проектів на GitHub за допомогою Visual Studio Metrics. Зібрано метрики: кількість класів, глибину дерева наслідування, зв’язність класів

– Перевірено емпіричні дані на нормальність розподілу. Результати показали негаусівський характер розподілу й наявність викидів, які були усунені за допомогою нормалізації log10 та аналізу квадрату відстані Махаланобіса.

– Побудовано лінійну та нелінійну регресійні моделі. Нелінійна модель має значено краще значення $MMRE$, втім обидві моделі мають низьку якість.

– Виконано розрахунок довірчих інтервалів і інтервалів передбачення. У більшості випадків нелінійна модель забезпечує менші ширини інтервальних оцінок.

– Проведено розробку програмного забезпечення для реалізації розрахунків за побудованою моделлю.

– Пораховано економічну ефективність від розробки та впровадження такого програмного забезпечення, розрахунки кажуть про його окупність.

– Розроблено розділи з охорони праці та охорони навколишнього середовища.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ojha T. R. Critical Success Factors of Agile Software Development - A Systematic Literature Review. *SCITECH Nepal*. 2023. Vol. 17, no. 1. P. 49–57. URL: https://doi.org/10.3126/scitech.v17i1.60467 (date of access: 10.12.2024).

2. Boehm B.W. Software engineering economics [Text] / B. W. Boehm. – Englewood Cliffs, NJ : Prentice Hall, 1981. – 768 p. – ISBN 0-13-822122-7.

3. Boehm B.W. Software cost estimation with COCOMO II / [B. W. Boehm, C. Abts, A. W. Brown et al.]. – Upper Saddle River, NJ: Prentice Hall PTR, 2000. – 506 p.

4. AL-Saleem A. L., Hammo A. Y. Software Size Estimation: A survey. Technium: Romanian Journal of Applied Sciences and Technology. 2022. Vol. 4, no. 9. P. 62–70. URL: https://doi.org/10.47577/technium.v4i9.7251 (date of access: 10.12.2024).

5. Ferens D. V. Software size estimation techniques. IEEE 1988 National Aerospace and Electronics Conference, Dayton, OH, USA. URL: https://doi.org/10.1109/naecon.1988.195083 (date of access: 10.12.2024).

6. Kumar Y. Comparative Analysis of Software Size Estimation Techniques in Project Management. International Journal for Research in Applied Science and Engineering Technology. 2017. Vol. V, no. VIII. P. 1470–1477. URL: https://doi.org/10.22214/ijraset.2017.8208 (date of access: 10.12.2024).

7. Prykhodko S., Shutko I., Prykhodko A. Early size estimation of web apps created using CodeIgniter framework by nonlinear regression models. *RADIOELECTRONIC AND COMPUTER SYSTEMS*. 2022. No. 3. P. 84–94. URL: https://doi.org/10.32620/reks.2022.3.06 (date of access: 09.11.2024).

8. Prykhodko S. B., Prykhodko N. V., Makarova L. M. Estimating the Software Size of Open-Source PHP-Based Systems Using Non-Linear Regression Analysis. *Proceedings of International Conference "Advanced Computer Information Technologies" (ACIT-2018)* : CEUR Workshop Proceedings, Ceske Budejovice, CZECH REPUBLIC. Ceske Budejovice, 2019. P. 199–202.

9. Prykhodko N. V., Prykhodko S. B. Non-linear regression model to estimate the software size of VB-based information systems. Information Technology And Computer Engineering. 2018. Vol. 43, no. 3. P. 37–42. URL: https://doi.org/10.31649/1999-9941-2018-43-3-37-42 (date of access: 09.11.2024).

10. Prykhodko S. B., Prykhodko N. V., Koltsov A. V. A nonlinear regression model for early LOC estimation of open-source Kotlin-based applications. Radio Electronics, Computer Science, Control. 2024. No. 1. P. 85. URL: https://doi.org/10.15588/1607-3274-2024-1-8 (date of access: 15.11.2024).

11. Prykhodko N. V., Prykhodko S. B. The non-linear regression model to estimate the software size of open source Java-based systems. *Radio Electronics, Computer Science*, Control. 2018. No. 3. URL: https://doi.org/10.15588/1607-3274-2018-3-17 (date of access: 09.11.2024)

12. Kiewkanya M., Surak S. Constructing C++ software size estimation model from class diagram. *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, Khon Kaen, Thailand, 13–15 July 2016. 2016. URL: https://doi.org/10.1109/jcsse.2016.7748880 (date of access: 09.11.2024).

13. Tan H. B. K., Zhao Y., Zhang H. Estimating LOC for information systems from their conceptual data models. *Proceeding of the 28th international conference, Shanghai*, China, 20–28 May 2006. New York, New York, USA, 2006. URL: https://doi.org/10.1145/1134285.1134331 (date of access: 09.11.2024).

14. Tan H. B. K., Zhao Y., Zhang H. Conceptual data model-based software size estimation for information systems. *ACM Transactions on Software Engineering and Methodology.* 2009. Vol. 19, no. 2. P. 1–37. URL: https://doi.org/10.1145/1571629.1571630 (date of access: 09.11.2024).

15. Петриченко С. А., Пухалевич А. В. Багатофакторна нелінійна регресійна модель для оцінювання розміру вебзастосунків, що створюються з використанням мови програмування C#. *Інформаційні технології: моделі, алгоритми, системи* : зб. тез IV всеукраїнської науково-практичної інтернет конференції, 30-31 жовтня. 2023 р. С 40–43.

16. Макарова Л. М., Латанська Л. О., Нікітін О. В., Нікітіна О. Ю. МАТЕМАТИЧНІ МОДЕЛІ ДЛЯ ОЦІНЮВАННЯ РОЗМІРУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КРОС-ПЛАТФОРМНОЇ РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ ІЗ ВИКОРИСТАННЯМ ПЛАТФОРМИ XAMARIN. *Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки.* 2022. Т. 33 (72), № 1. С. 150–156.

17. Каіров В. О., Ларіонов В. А,. Ранне оцінювання розміру програмних застосунків мовою C#. *Сучасні інформаційні системи та технології* : зб. тез VII всеукраїнської науково-практичної інтернет конференції, 29 листопада. 2024 р. С 176.

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

19. PRYKHODKO N. V., PRYKHODKO S. B. Constructing the Nonlinear Regression Models on the Basis of Multivariate Normalizing Transformations. *Èlektronnoe modelirovanie*. 2018. Vol. 40, no. 6. P. 101–110. URL: https://doi.org/10.15407/emodel.40.06.101 (date of access: 09.11.2024).

20. Prykhodko S., Prykhodko N. Mathematical Modeling of Non-Gaussian Dependent Random Variables by Nonlinear Regression Models Based on the Multivariate Normalizing Transformations. *Advances in Intelligent Systems and Computing*. Cham, 2020. P. 166–174. URL: https://doi.org/10.1007/978-3-030-58124-4_16 (date of access: 09.11.2024).

21. MARDIA K. V. Measures of multivariate skewness and kurtosis with applications. Biometrika. 1970. Vol. 57, no. 3. P. 519–530. URL: https://doi.org/10.1093/biomet/57.3.519 (date of access: 09.11.2024).

22. Frost J. Regression Analysis: An Intuitive Guide for Using and Interpreting Linear Models. Statistics By Jim Publishing, 2020. 355 p.

23. Prykhodko S. B., Prykhodko N. V., Makarova L. M., Pugachenko K. M. Detecting Outliers in Multivariate Non-Gaussian Data on the basis of Normalizing Transformations. Proceedings of the 2017 IEEE First Ukraine Conference on Electrical

and Computer Engineering (UKRCON) : «Celebrating 25 Years of IEEE Ukraine Section». Kyiv, Ukraine, May 29-June 2. Kyiv, 2017. P. 846–849.

24. Prykhodko S. B., Prykhodko N. V., Makarova L. M. Pukhalevych A. V. Application of the Squared Mahalanobis Distance for Detecting Outliers in Multivariate 68 Non-Gaussian Data. Proceedings of 14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET) : LvivSlavske, Ukraine, February 20–24. Lviv-Slavske, 2018. P. 962–965.

25. An Introduction to Statistical Learning / G. James et al. New York, NY : Springer US, 2021. URL: https://doi.org/10.1007/978-1-0716-1418-1 (date of access: 09.11.2024).

26. Приходько С. Б., Макарова Л. М., Пугаченко К. С. Методичні вказівки та завдання до виконання лабораторних робіт з дисципліни «Обробка експериментальних даних на комп`ютері». Миколаїв : НУК, 2018. 76 с.

27. A simulation study of the model evaluation criterion mmre / T. Foss et al. IEEE Transactions on Software Engineering. 2003. Vol. 29, no. 11. P. 985–995. URL: https://doi.org/10.1109/tse.2003.1245300 (date of access: 09.11.2024).

28. Prykhodko S., Prykhodko N. Mathematical Modeling of Non-Gaussian Dependent Random Variables by Nonlinear Regression Models Based on the Multivariate Normalizing Transformations. Advances in Intelligent Systems and Computing. Cham, 2020. P. 166–174. URL: https://doi.org/10.1007/978-3-030-58124-4_16 (date of access: 09.11.2024).

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ

Вступ

Найменування програми: «C# LOC розмір».

1 Підстави для розробки

Підставою для розробки є наказ на затвердження тем кваліфікаційних робіт магістрів.

2 Призначення розробки

2.1 Функціональне призначення

Функціональним призначенням програми є надання користувачу можливості оцінити розмір програмних застосунків на C#.

2.2 Експлуатаційне призначення

Програма призначена для експлуатації на персональних комп’ютерах користувачів для оцінювання розміру програмних застосунків на C#.

3 Вимоги до програми або програмного продукту

3.1 Вимоги для функціональних характеристик

3.1.1 Вимоги до переліку виконуваних функцій

Розроблена програма має надавати можливість виконувати наступні функції:

– Оцінювання розміру програмних застосунків на C# у тисячах строк коду.

– Оцінювання довірчого інтервалу значення розміру програмних застосунків на C#.

– Оцінювання інтервалу передбачення значення розміру програмних застосунків на C#.

3.1.2 Вимоги для організації вхідних та вихідних даних

Введення вхідних даних відбувається за рахунок заповнення полів графічного інтерфейсу користувача.

Результати виводяться у відповідні поля графічного інтерфейсу користувача.

Вхідними даними для побудови нелінійної регресійної моделі є набір метрик з проектів програмних застосунків на C#.

Вхідними даними для проведення оцінювання розміру проектів на C# є такі метрики програмного забезпечення як кількість класів (Classes), глибина дерева наслідування (DIT), зв’язність класів (CBO). Також для виконання оцінювання необхідно вказати довірчу ймовірність.

Кількість класів застосунку на C# – це ціле число, яке більше за одиницю.

Глибина дерева наслідування, зв’язність класів та довірча ймовірність, фактичний розмір – дійсні додатні числа. Довірча ймовірність приймає значення не більші за одиницю.

Вихідними даними є отримана оцінка розміру програмних застосунків на C# у тисячах рядків коду, оцінки довірчого інтервалу та інтервалу передбачення, які буде відображено у відповідних полях графічного інтерфейсу програми.

3.2 Вимоги до надійності

Надійне (стійке) функціонування програмного забезпечення повинно бути забезпечено виконанням сукупності організаційно-технічних заходів, виконанням валідації вхідних даних згідно вимог та відображення повідомлень про виявлення некоректних даних з закликом ввести коректні.

3.3 Умови експлуатації

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

3.4 Вимоги до технічної та програмної сумісності

Програма має експлуатуватись на обладнанні з параметрами не нижчими за наступні: 64-бітний процесор на архітектурі x86 або ARM, оперативна пам’ять 2Гб (1 Гб мінімум), простір на жорсткому диску 100 Мб мінімум.

Для запуску та подальшої роботи програмного забезпечення необхідна операційна система Windows 7/8/10/11, Ubuntu 17/18/19/20.

3.5 Вимоги до захисту інформації та програм

Вимоги до захисту інформації та програм не висуваються.

3.6 Вимоги до маркування та упаковки

Вимоги до маркування та упаковки не висуваються.

3.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4 Вимоги до програмної документації

Склад програмної документації має включати в себе:

1) Технічне завдання;
2) Програму та методику випробувань;
3) Інструкцію користувача;
4) Опис програмного забезпечення.
5) Текст програми

5 Техніко-економічні показники

Економічна ефективність від впровадження програмного забезпечення була розрахована у розділі 4. Виходячи з одержаних результатів, впровадження даного програмного забезпечення є доцільним, приблизний термін окупності – 4 місяці.

6 Стадії та етапи розробки програмного забезпечення

Стадії та етапи виконання розробки програмного забезпечення можна представити наступним чином:

Таблиця А.1 – Стадії та етапи розробки

Стадії розробки	Назва етапу	Термін виконання
Технічне завдання	Розробка технічного завдання	25.09.2024
	Затвердження технічного завдання	02.10.2024
Ескізний проект	Розробка ескізного проекту	17.10.2024
	Затвердження ескізного проекту	27.10.2024
Технічний проект	Розробка технічного проекту	10.11.2024
	Затвердження технічного проекту	26.11.2024
Робочий проект	Розробка програми	29.11.2024
	Розробка програмної документації	02.12.2024
	Випробування програми	04.12.2024

7 Порядок контролю та прийомки

Приймально-здавальні роботи мають проводитись під наглядом Замовника або уповноваженої ним особи.

Приймально-здавальні випробування повинні проводитись згідно документу «Програма та методика випробувань», який узгоджується між стороною розробника та стороною замовника.

Хід проведення приймально-здавальних робіт Замовник та Виконавець документують в «Протокол проведення випробувань».

На основі Протоколу проведення випробувань Виконавець та Замовник підписують «Акт прийому-здачі програми у експлуатацію».

ДОДАТОК Б – ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ

1 Об’єкт випробувань

Об’єктом дослідження є програмне забезпечення для оцінювання розміру програмних застосунків на C#

2 Мета випробувань

Мета випробувань – перевірка працездатності програмного застосунку, перевірка відповідності фактичних характеристик програмного застосунку тим вимогам, які було викладено у технічному завданні.

3 Вимоги до застосунку

Під час виконання випробувань перевіряються функціональні характеристики (вимоги), що були викладені в технічному завданні.

Розроблена програма має надавати можливість виконувати наступні функції:

– Оцінювання розміру програмних застосунків на C# у тисячах строк коду.

– Оцінювання довірчого інтервалу значення розміру програмних застосунків на C#.

– Оцінювання інтервалу передбачення значення розміру програмних застосунків на C#.

4 Вимоги до програмної документації

Програмна документація включає в себе наступні документи:

1) технічне завдання;

2) програму та методику випробувань;

3) інструкцію користувача;

4) опис програмного забезпечення.

5) текст програми

5 Засоби і порядок випробувань

5.1 Технічні засоби, що використовуються під час випробувань

Склад технічних засобів, що використовувались під час випробувань:

ASUS VivoBook 15 X1504VA-BQ207W.

– Дванадцятипотоковий Intel Core i5-1335U (1.3 - 4.6 ГГц).

– Оперативна пам’ять: 8 Гбайт, DDR4.

– Накопичувач даних: SSD 512 Гбайт.

– Графіка: Intel Iris Xe Graphics (вбудована).

– Екран: 15.6" IPS Full HD (1920x1080), 60 Гц.

5.2 Програмні засоби, що використовувались під час випробувань

Операційна система Windows 10 Pro 21H2 204.2225

5.3 Порядок проведення випробувань

Порядок проведення випробувань – перевірка відповідності функціональних характеристик програмного застосунку, перевірка вимог до програмної документації.

6 Методи випробувань

6.1 Методика проведення перевірки вимог до програмної документації

За відсутності конкретних вимог до оформлення та подання програмної документації, розроблені програмні документи перевіряються представником замовника візуально. Уповноважена особа перевіряє відповідність розроблених програмних документів з тими, що були вказані у пункті «Склад програмної документації, що пропонується на випробування».

6.2 Методика проведення перевірки вимог до функціональних характеристик програмного продукту

Перевірка працездатності програми виконується згідно з пунктом «Вимоги до функціональних характеристик» технічного завдання.

Перевірку можна вважати успішно завершеною у випадку відповідності виконуваних функції технічному завданню.

Порядок випробувань:

– Перевірка роботи застосунку на моніторах з різною розподільною здатністю.

– Перевірка відображення шрифтів.

– Перевірка кожної форми графічного інтерфейсу на можливість введення даних.

– Перевірка кожної форми графічного інтерфейсу на введення невалідних даних.

– Перевірка можливості виконання оцінювання розміру, визначення довірчого інтервалу та інтервалу передбачення програмних застосунків на C#.

7 Результати випробувань

За результатами випробувань було підтверджено відповідність функціональних можливостей висунутим вимогам до програмного забезпечення.

8 Відомості про відмови, збої та аварійні ситуації

Під час проведення випробувань відмов, збоїв та аварійних ситуацій помічено не було

9 Відомості про коригування параметрів об’єкта випробувань та технічної документації

Коригування параметрів об’єкта випробувань та технічної документації в процесі виконання випробувань не проводилось

ДОДАТОК В – ІНСТРУКЦІЯ КОРИСТУВАЧА

Вступ

Інструкція користувача призначена для ознайомлення користувача з правилами та порядком виконання операцій, які забезпечуються роботою програмного забезпечення.

Ця програма призначена для оцінювання розміру та визначення інтервальних оцінок розміру у тисячах строк коду програмних застосунків на C#. Програма є вільним програмним забезпеченням з графічним інтерфейсом користувача з можливостю вводити дані та переглядати результати.

Загальні відомості про програму

Найменування програми: «C# LOC розмір».

Використані мови програмування

Програма написана на мові Python, для розробки графічного інтерфейсу користувача було використано сумісну з Python бібліотеку Qt.

Призначення програми

Програма призначена для оцінювання розміру (в тисячах рядків коду) програмних застосунків на C# на основі таких метрик як кількість класів (Classes, глибина дерева наслідування (DIT, Depth of Inheritance Tree), зв’язність об’єктів (CBO, Coupling Between Objects)

Функціональні можливості програми

– Оцінювання розміру програмних застосунків на C# у тисячах строк коду.

– Оцінювання довірчого інтервалу значення розміру програмних застосунків на C#.

– Оцінювання інтервалу передбачення значення розміру програмних застосунків на C#.

Обмеження області застосування програми

Область застосування програми обмежена галуззю розробки програмного забезпечення.

Умови, необхідні для використання програми

Спеціальні умови для використання програми відсутні

Вимоги до технічної сумісності та програмного середовища

Програма має експлуатуватись на обладнанні з параметрами не нижчими за наступні: 64-бітній процесор на архітектурі x86 або ARM, оперативна пам’ять 2Гб (1 Гб мінімум), простір на жорсткому диску 100 Мб мінімум.

Для запуску та подальшої роботи програмного забезпечення необхідна операційна система Windows 7/8/10/11, Ubuntu 17/18/19/20.

Загальний вигляд стартового вікна програми «C# LOC розмір» представлено на рисунку В.1.

Інтерфейс програми містить поле для вибору файлу з метриками, необхідними для створення нелінійної регресійної моделі та виконання оцінки. Щоб обрати файл, користувач має натиснути кнопку «Знайти» та вибрати потрібний файл зі списку доступних на пристрої.

Після цього потрібно вказати відповідні метрики оцінюваного застосунку на C# та довірчу ймовірність:

– Поле «Класи (X1)» призначене для введення кількості класів у програмному застосунку на C#. Значення має бути невід’ємним числом, більшим за одиницю.

– Поле «CBO (X2)» дозволяє вказати зв’язність між класами застосунку. Вводиться дійсне невід’ємне число, більше одиниці.

– Поле «DIT (X3)» передбачене для введення середнього значення глибини дерева наслідування класів. Глибина вводиться у вигляді дійсного невід’ємного числа.

– Поле «Довірчий рівень (0-1)» використовується для встановлення довірчої ймовірності моделювання. Значення вводиться у межах від 0 до 1 включно.

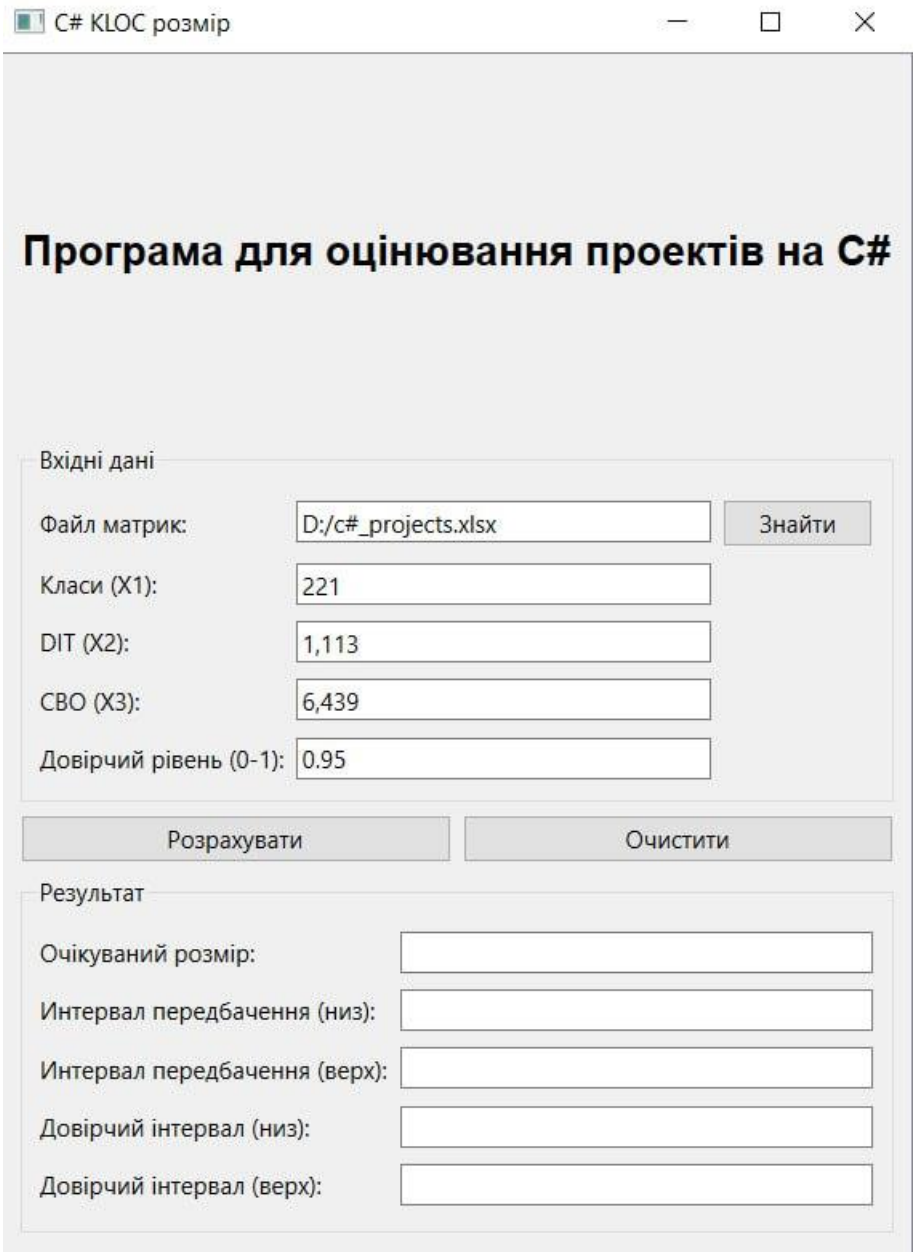

Рисунок В.1 – Графічний інтерфейс до оцінювання

Після заповнення вхідних даних і запуску процесу обчислень за допомогою відповідної кнопки, програма виконає розрахунок оцінок. Результати, включно з розміром програмного застосунку у тисячах рядків коду, довірчими інтервалами та інтервалами прогнозування нелінійної регресії, відображатимуться в нижній частині екрану.

Результат виконання оцінювання можна побачити на рисунку В.2.

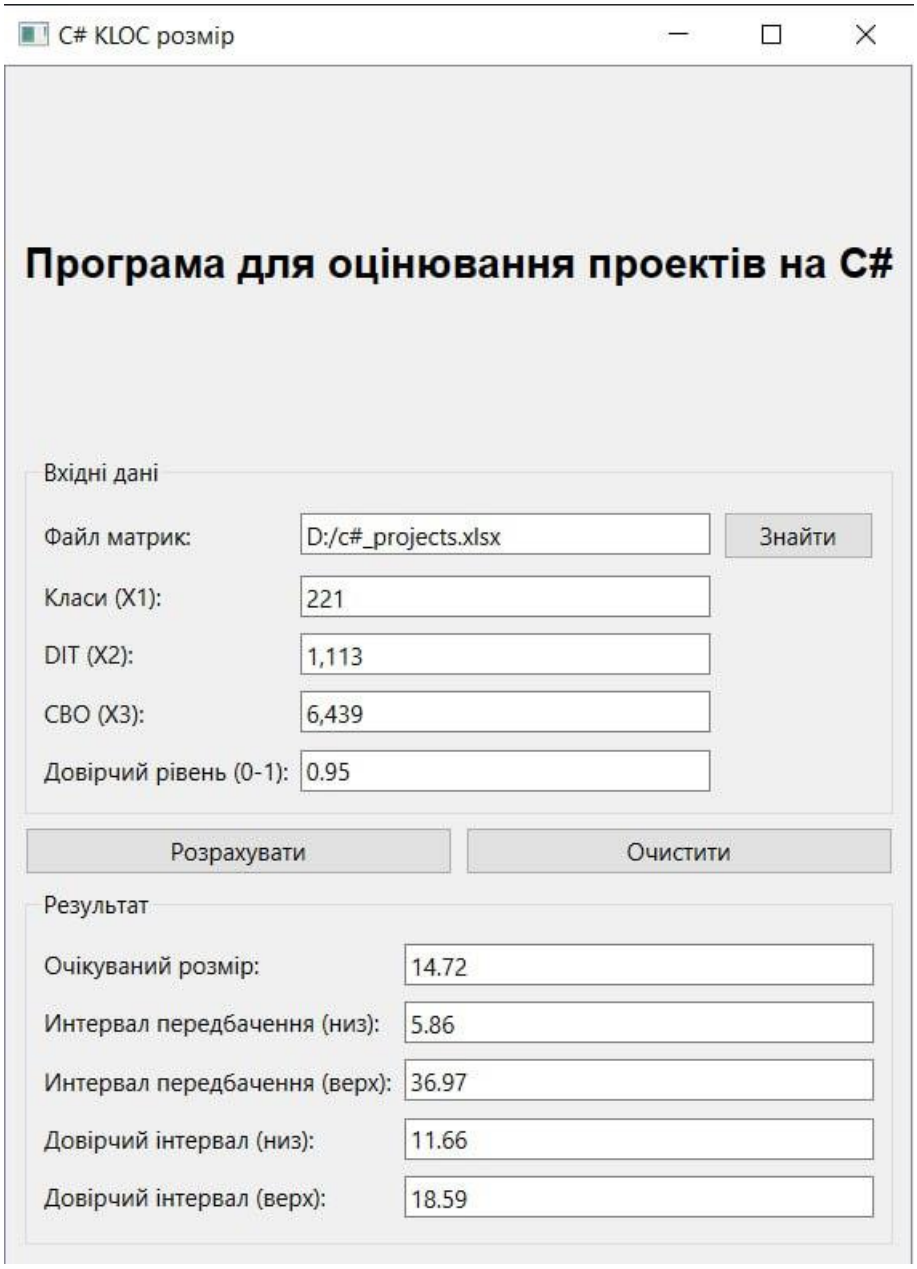

Рисунок В.2 – Графічний інтерфейс після оцінювання

ДОДАТОК Г – ТЕКСТ ПРОГРАМИ

```
import numpy as np
import pandas as pd
import math
from scipy.stats import norm
from scipy.stats import chi2
from scipy.stats import t
from scipy.stats import chisquare
from scipy.stats import normaltest
from scipy.stats import shapiro
from scipy.stats import chi2_contingency
from statistics import NormalDist
from sklearn import linear_model
import statsmodels.api as sm

class DataHandler:
    def __init__(self, data_input):
        self.data_frame = data_input.copy()
        self.total_columns = len(self.data_frame.columns)
        self.features_count = self.total_columns - 1

    def calculate_chi_squared(self, data_column, bins=6):
        total_elements = len(data_column)

        avg_value = np.mean(data_column)
        std_dev = np.std(data_column, ddof=1)

        min_value = data_column.min()
        max_value = data_column.max()
        bin_width = (max_value - min_value) / bins

        lower_limit = min_value
        upper_limit = lower_limit + bin_width

        chi_square_total = 0
        for i in range(bins):
            if i == bins - 1:
                selected_data = data_column[(data_column <= max_value) & (data_column
>= lower_limit)]
            else:
                selected_data = data_column[(data_column < upper_limit) &
(data_column >= lower_limit)]

            probability = norm(avg_value, std_dev).cdf(upper_limit) - norm(avg_value,
std_dev).cdf(lower_limit)
```

```
            chi_square_total += (len(selected_data) - total_elements * probability)
** 2 / (total_elements * probability)

            lower_limit = upper_limit
            upper_limit += bin_width

        return chi_square_total

    def mahalanobis_distance_calc(self, matrix_data):
        centered_matrix = matrix_data - np.mean(matrix_data, axis=0)

        cov_matrix = np.cov(matrix_data.values.T, bias=True)

        inverse_cov_matrix = np.linalg.inv(cov_matrix)

        dot_product = np.dot(centered_matrix, inverse_cov_matrix)

        mahalanobis_dist = np.dot(dot_product, centered_matrix.T)
        return mahalanobis_dist.diagonal()

    def apply_function(self, func, data_frame=None):
        if data_frame is None:
            data_frame = self.data_frame.copy()

        columns = data_frame.columns

        for index, column in enumerate(columns):
            new_column = 'Modified_' + column
            data_frame[new_column] = data_frame.iloc[:, index].map(lambda x: func(x))

        return data_frame

    def detect_outliers_based_on_mahalanobis(self, data_frame):
        while True:
            data_frame['mahalanobis_score'] = self.mahalanobis_distance_calc(
                matrix_data=data_frame.iloc[:,
self.features_count:self.features_count * 2]
            )
            critical_value = chi2.ppf(1 - 0.05, self.total_columns)
            max_index = data_frame['mahalanobis_score'].idxmax()
            max_distance = data_frame['mahalanobis_score'].max()

            if max_distance >= critical_value:
                data_frame = data_frame.drop(index=max_index)
            else:
                break

        return data_frame
```

```
    def compute_polynomial_model(self, data_frame, model_coefficients,
intercept_value):
        return (data_frame ** model_coefficients).prod(axis=1) * 10 **
intercept_value

    def compute_covariance_scalars(self, data_frame=None):
        column_names = data_frame.columns
        num_columns = len(column_names)

        self.column_means = np.mean(data_frame, axis=0)
        centered_data = data_frame - np.mean(data_frame, axis=0)

        for index in range(num_columns):
            squared_column = column_names[index] + '_Squared'
            centered_data[squared_column] = centered_data.iloc[:, index].map(lambda
x: x * x)

        covariance_matrix = np.zeros((num_columns, num_columns))

        for i in range(num_columns):
            for j in range(num_columns):
                covariance_matrix[i][j] = (centered_data.iloc[:, i] *
centered_data.iloc[:, j]).sum()

        relevant_data = centered_data.iloc[:, 0:self.features_count]

        result_array = np.array([])

        inverse_cov_matrix = np.linalg.inv(covariance_matrix)
        self.inverse_cov_matrix = inverse_cov_matrix

        for _, row in relevant_data.iterrows():
            result_value = np.dot(np.dot(row, inverse_cov_matrix), row.T)
            result_array = np.append(result_array, result_value)

        return pd.DataFrame(result_array)

    def calculate_error(self, data_frame):
        total_samples = len(data_frame)

        squared_residuals_sum =
data_frame[['residuals_linear']].pow(2).sum()['residuals_linear']

        return math.sqrt(1.0 / (total_samples - self.features_count - 1) *
squared_residuals_sum)

    def project_data_metrics(self, metrics_data, confidence_level=0.05):
        metrics_data = self.apply_function(func=math.log10, data_frame=metrics_data)
```

```
        centered_metrics = metrics_data.iloc[:, self.features_count:] -
self.column_means

        metrics_data['prediction_regression'] = self.compute_polynomial_model(
            data_frame=metrics_data.iloc[:, 0:self.features_count],
            model_coefficients=self.regression_model.coef_,
            intercept_value=self.regression_model.intercept_
        )

        metrics_data['covariance_metrics'] = np.dot(np.dot(centered_metrics,
self.inverse_cov_matrix), centered_metrics.T)

        t_quantile = t.isf((1 - confidence_level) / 2, self.total_data_size -
self.features_count - 1)

        margin_error = t_quantile * self.error_standard * math.sqrt(1 /
self.total_data_size + float(metrics_data['covariance_metrics'][0]))
        prediction_margin = t_quantile * self.error_standard * math.sqrt(1 + (1 /
self.total_data_size) + float(metrics_data['covariance_metrics'][0]))

        metrics_data['lower_bound_confidence'] = 10 **
(math.log10(metrics_data['prediction_regression'][0]) - margin_error)
        metrics_data['upper_bound_confidence'] = 10 **
(math.log10(metrics_data['prediction_regression'][0]) + margin_error)
        metrics_data['lower_bound_prediction'] = 10 **
(math.log10(metrics_data['prediction_regression'][0]) - prediction_margin)
        metrics_data['upper_bound_prediction'] = 10 **
(math.log10(metrics_data['prediction_regression'][0]) + prediction_margin)

        return metrics_data

    def run_nonlinear_model(self):
        working_data = self.data_frame

        while True:
            log_transformed_data = self.apply_function(func=math.log10,
data_frame=working_data.iloc[:, :self.total_columns])

            log_transformed_data =
self.detect_outliers_based_on_mahalanobis(data_frame=log_transformed_data)

            log_transformed_data =
log_transformed_data.reset_index().drop(labels='index', axis=1)

            predictor_columns = log_transformed_data.iloc[:, self.total_columns + 1:2
* self.total_columns]
            target_column = log_transformed_data.iloc[:, self.total_columns]

            self.regression_model = linear_model.LinearRegression()
            self.regression_model.fit(predictor_columns, target_column)
```

```
            regression_summary = sm.OLS(target_column, predictor_columns).fit()
            predictions = self.regression_model.predict(predictor_columns)

            log_transformed_data['linear_predictions'] = pd.DataFrame(predictions)
            log_transformed_data['residuals_linear'] =
log_transformed_data['linear_predictions'] - log_transformed_data['Modified_Y']

            chi2_critical_value = chi2.ppf(1 - 0.05, 3)
            chi2_value =
self.calculate_chi_squared(data_column=log_transformed_data['residuals_linear'])

            log_transformed_data['nonlinear_predictions'] =
self.compute_polynomial_model(
                data_frame=log_transformed_data.iloc[:, 1:self.total_columns],
                model_coefficients=self.regression_model.coef_,
                intercept_value=self.regression_model.intercept_
            )

            log_transformed_data['covariance_values'] =
self.compute_covariance_scalars(data_frame=log_transformed_data.iloc[:,
self.total_columns + 1:self.total_columns * 2])

            self.total_data_size = len(log_transformed_data)

            self.error_standard =
self.calculate_error(data_frame=log_transformed_data)

            t_quantile = t.isf(0.025, self.total_data_size - self.features_count - 1)

            log_transformed_data['confidence_margin'] =
log_transformed_data[['covariance_values']].map(lambda x: t_quantile *
self.error_standard * math.sqrt(1 / self.total_data_size + x))
            log_transformed_data['prediction_margin'] =
log_transformed_data[['covariance_values']].map(lambda x: t_quantile *
self.error_standard * math.sqrt(1 + (1 / self.total_data_size) + x))

            log_transformed_data['lower_confidence_bound'] = 10 **
(log_transformed_data['linear_predictions'] -
log_transformed_data['confidence_margin'])
            log_transformed_data['upper_confidence_bound'] = 10 **
(log_transformed_data['linear_predictions'] +
log_transformed_data['confidence_margin'])
            log_transformed_data['lower_prediction_bound'] = 10 **
(log_transformed_data['linear_predictions'] -
log_transformed_data['prediction_margin'])
            log_transformed_data['upper_prediction_bound'] = 10 **
(log_transformed_data['linear_predictions'] +
log_transformed_data['prediction_margin'])
```

```
            outliers = log_transformed_data[(log_transformed_data.Y <=
log_transformed_data.lower_prediction_bound) | (log_transformed_data.Y >=
log_transformed_data.upper_prediction_bound)]

            if chi2_critical_value > chi2_value and outliers.empty:
                break
            elif chi2_critical_value < chi2_value:
                if abs(log_transformed_data['residuals_linear'].max()) >
abs(log_transformed_data['residuals_linear'].min()):
                    residual_to_remove =
log_transformed_data['residuals_linear'].idxmax()
                else:
                    residual_to_remove =
log_transformed_data['residuals_linear'].idxmin()

                log_transformed_data =
log_transformed_data.drop(index=residual_to_remove)
                working_data = log_transformed_data
            elif not outliers.empty:
                log_transformed_data = log_transformed_data.drop(outliers.index)
                working_data = log_transformed_data


import sys
from PyQt6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QGridLayout,
    QLabel, QLineEdit, QPushButton, QFileDialog, QGroupBox, QScrollArea, QComboBox
)
from PyQt6.QtCore import Qt
from PyQt6.QtGui import QFont, QDoubleValidator, QIntValidator
import pandas as pd
from pathlib import Path
from DataHandler import DataHandler

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("C# KLOC розмір")
        self.resize(900, 600)

        self.setupUi()

    def setupUi(self):
        # Scrollable central widget
        centralWidget = QWidget()
        scrollArea = QScrollArea()
        scrollArea.setWidgetResizable(True)
        scrollArea.setWidget(centralWidget)
        self.setCentralWidget(scrollArea)
```

```
        # Main Layout
        mainLayout = QVBoxLayout()
        centralWidget.setLayout(mainLayout)

        # Header
        headerLabel = QLabel("Програма для оцінювання проектів на C#")
        headerLabel.setFont(QFont("Arial", 16, QFont.Weight.Bold))
        headerLabel.setAlignment(Qt.AlignmentFlag.AlignCenter)
        mainLayout.addWidget(headerLabel)

        # Input Section
        inputGroupBox = QGroupBox("Вхідні дані")
        inputGridLayout = QGridLayout()
        inputGroupBox.setLayout(inputGridLayout)
        mainLayout.addWidget(inputGroupBox)

        self.dataFileInput = self.createFileSelectionField("Файл матрик:",
inputGridLayout, 0)
        self.classCountInput = self.createTextInput("Класи (X1):", QIntValidator(),
inputGridLayout, 1)
        self.ditValueInput = self.createTextInput("DIT (X2):", QDoubleValidator(),
inputGridLayout, 2)
        self.cboValueInput = self.createTextInput("CBO (X3):", QDoubleValidator(),
inputGridLayout, 3)
        self.confidenceInput = self.createTextInput("Довірчий рівень (0-1):",
QDoubleValidator(0.0, 1.0, 2), inputGridLayout, 4, "0.95")

        # Buttons
        buttonLayout = QHBoxLayout()
        self.calculateBtn = QPushButton("Розрахувати")
        self.calculateBtn.clicked.connect(self.performCalculation)
        self.clearBtn = QPushButton("Очистити")
        self.clearBtn.clicked.connect(self.resetFields)
        buttonLayout.addWidget(self.calculateBtn)
        buttonLayout.addWidget(self.clearBtn)
        mainLayout.addLayout(buttonLayout)

        # Output Section
        outputGroupBox = QGroupBox("Результат")
        outputGridLayout = QGridLayout()
        outputGroupBox.setLayout(outputGridLayout)
        mainLayout.addWidget(outputGroupBox)

        self.projectEstimateOutput = self.createOutputField("Очікуваний розмір:",
outputGridLayout, 0)
        self.lowerPredOutput = self.createOutputField("Интервал передбачення (низ):",
outputGridLayout, 1)
        self.upperPredOutput = self.createOutputField("Интервал передбачення
(верх):", outputGridLayout, 2)
```

```
        self.lowerConfOutput = self.createOutputField("Довірчий інтервал (низ):",
outputGridLayout, 3)
        self.upperConfOutput = self.createOutputField("Довірчий інтервал (верх):",
outputGridLayout, 4)

    def createFileSelectionField(self, label, layout, row):
        labelWidget = QLabel(label)
        filePathInput = QLineEdit()
        browseBtn = QPushButton("Знайти")
        browseBtn.clicked.connect(lambda: self.selectFile(filePathInput))
        layout.addWidget(labelWidget, row, 0)
        layout.addWidget(filePathInput, row, 1)
        layout.addWidget(browseBtn, row, 2)
        return filePathInput

    def selectFile(self, inputField):
        filename, _ = QFileDialog.getOpenFileName(self, "Обрати файл", "", "Файли
(*.xlsx *.csv *.txt)")
        if filename:
            inputField.setText(filename)

    def createTextInput(self, label, validator, layout, row, defaultValue=""):
        labelWidget = QLabel(label)
        textInput = QLineEdit()
        textInput.setValidator(validator)
        textInput.setText(defaultValue)
        layout.addWidget(labelWidget, row, 0)
        layout.addWidget(textInput, row, 1)
        return textInput

    def createOutputField(self, label, layout, row):
        labelWidget = QLabel(label)
        outputField = QLineEdit()
        outputField.setReadOnly(True)
        layout.addWidget(labelWidget, row, 0)
        layout.addWidget(outputField, row, 1)
        return outputField

    def resetFields(self):
        self.dataFileInput.clear()
        self.classCountInput.clear()
        self.ditValueInput.clear()
        self.cboValueInput.clear()
        self.confidenceInput.setText("0.95")
        self.projectEstimateOutput.clear()
        self.lowerPredOutput.clear()
        self.upperPredOutput.clear()
        self.lowerConfOutput.clear()
        self.upperConfOutput.clear()
```

```
    def performCalculation(self):
        filePath = self.dataFileInput.text()
        classCount = float(self.classCountInput.text())
        ditValue = float(self.ditValueInput.text())
        cboValue = float(self.cboValueInput.text())
        confidenceLevel = float(self.confidenceInput.text())

        if not filePath:
            raise ValueError("Файл не знайдено.")

        data = pd.read_excel(filePath)
        handler = DataHandler(data)
        handler.run_nonlinear_model()

        result = handler.project_data_metrics(
            metrics_data=pd.DataFrame({"X1": [classCount], "X2": [ditValue], "X3":
[cboValue]}),
            confidence_level=confidenceLevel
        )

        self.projectEstimateOutput.setText('%.2f' %
result['prediction_regression'][0])
        self.lowerPredOutput.setText( f"{result['lower_bound_prediction'][0]:.2f}")
        self.upperPredOutput.setText( f"{result['upper_bound_prediction'][0]:.2f}")
        self.lowerConfOutput.setText( f"{result['lower_bound_confidence'][0]:.2f}")
        self.upperConfOutput.setText( f"{result['upper_bound_confidence'][0]:.2f}")


if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

ДОДАТОК Д – ОПИС ПРОГРАМИ

1 Загальні відомості

Найменування: «C# LOC розмір».

1.2 Програмне забезпечення, необхідне для функціонування програми

Для запуску та подальшої роботи програмного забезпечення необхідна операційна система Windows 7/8/10/11, Ubuntu 17/18/19/20.

1.3 Мови програмування

Програмне забезпечення для оцінювання розміру програмних застосунків на C#, розроблене з використанням мови програмування Python та використанням бібліотек Pandas, NumPy та SciPy для виконання математичних обчислень. Для створення графічного інтерфейсу користувача було використано сумісну з Python бібліотеку Qt.

2 Функціональне призначення

Функціональним призначенням програми є надання користувачу можливості оцінити програмні застосунки на C#.

Функції програмного застосунку, які доступні користувачу:

– Оцінювання розміру програмних застосунків на C# у тисячах строк коду.

– Оцінювання довірчого інтервалу значення розміру програмних застосунків на C#.

– Оцінювання інтервалу передбачення значення розміру програмних застосунків на C#.

2.1 Функціональні обмеження

Програмне забезпечення не вимагає використання зовнішніх ресурсів, тому функціональні обмеження відсутні.

3 Технічні засоби, що використовуються

Програма має експлуатуватись на обладнанні з параметрами не нижчими за наступні: 64-бітній процесор на архітектурі x86 або ARM, оперативна пам’ять 2Гб (1 Гб мінімум), простір на жорсткому диску 100 Мб мінімум.

4 Виклик та завантаження

Для запуску та подальшої роботи програмного забезпечення необхідна операційна система Windows 7/8/10/11, Ubuntu 17/18/19/20.

Запуск програмного застосунку відбувається за рахунок натискання на ярлик програмного застосунку.

5 Вхідні дані

Вхідними даними для побудови нелінійної регресійної моделі є файл з метриками з розширенням .txt/.csv/.xlsx в якому в першій колонці знаходиться залежна величина – кількість рядків коду у тисячах, а у другому-четвертому кількість класів, кількість методів на кожен клас та глибина дерева наслідування відповідно.

Вхідними даними для оцінювання розміру програмних застосунків на C# є кількість класів, середня кількість методів на кожен класів, глибина дерева наслідування, довірча ймовірність.

Кількість класів застосунку на C# – це ціле число, яке більше за одиницю.

Значення середньої кількості методів, глибини дерева наслідування застосунку на C# мають бути дійсними додатними числами.

Довірча ймовірність вказує на відсотки та є дійсним числом від нуля до одиниці включно.

6 Вихідні дані

Вихідними даними є отримані оцінки розміру у тисячах рядків коду, оцінки довірчого інтервалу та інтервалу передбачення. Отримані значення буде відображено у відповідних полях графічного інтерфейсу програми.