МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики та інформатики

«До захисту допущено»

Завідувачка кафедри

_____Наталія МАСЛОВА

(підпис) (ім'я та прізвище)

«_____»_____2023 р.

Випускна кваліфікаційна робота

магістра

(освітній ступінь)

на тему «Дослідження та проєктування інтелектуальної системи оцінювання продуктивності розробників програмного забезпечення»/ «Research and design of an intelligent system for assessing the productivity of software developers»

Виконав: студент 2м курсу, групи ІПЗм-22

(шифр групи)

спеціальності (напрямку підготовки) 121 Інженерія програмного забезпечення

(шифр і назва напряму підготовки, спеціальності)

Криволап Віктор Васильович

(прізвище та ініціали) (підпис)

Керівник доц. кафедри ПМІ, к.т.н., Тетяна АЛТУХОВА

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доц. каф. ЕТ, к.т.н., доц. Віталій ШАМАЄВ _____

(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Рецензент зав. каф. АТ, к.т.н., доц. Валерій ПОЦЕПАЄВ _____

(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Луцьк – 2023р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики
Освітній ступінь магістр
Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувачка кафедри ПМІ
_____ Наталія МАСЛОВА
"____" _____2023 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Криволапу Віктору Васильовичу

1. Тема роботи «Дослідження та проєктування інтелектуальної системи оцінювання продуктивності розробників програмного забезпечення»/ «Research and design of an intelligent system for assessing the productivity of software developers»
Керівник роботи Алтухова Т.В., канд. техн. наук, доцент каф. ПМІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом від «22» вересня 2023 року № 478

2. Строк подання студентом роботи 05 грудня 2023 р.

3. Вихідні дані до роботи результати науково-дослідної роботи, матеріали переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області, формулювання завдань дослідження
2. Розробка моделі і реалізація системи оцінювання
3. Дослідження розробленої системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
робота містить 40 рисунків та інший матеріал, у кількості, достатній для демонстрації результатів кваліфікаційної роботи магістра

6. Консультанти розділів роботи

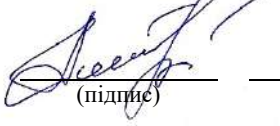
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Назарова І.А., доцент каф. ПМІ		

7. Дата видачі завдання 02 жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області, формулювання завдань дослідження, визначення актуальності майбутньої розробки, огляд літературних джерел	02.10.2023-15.10.2023	
2	Розробка моделі і реалізація системи оцінювання продуктивності розробників програмного забезпечення	16.10.2023-12.11.2023	
3	Дослідження розробленої системи оцінювання продуктивності розробників програмного забезпечення	13.11.2023-27.11.2023	
4	Оформлення супутніх матеріалів (розробка графічного матеріалу, тощо) та рецензування роботи	28.11.2023-05.12.2023	
5	Написання доповіді	06.12.2023-08.12.2023	
6	Захист роботи	20.12.2023	

Студент Віктор КРИВОЛАП
(підпис) (прізвище та ініціали)

Керівник роботи  Тетяна АЛТУХОВА
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Криволап В. В. Дослідження та проєктування інтелектуальної системи оцінювання продуктивності розробників програмного забезпечення / Випускна кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення». – ДВНЗ ДонНТУ, Луцьк, 2023.

Об’єктом дослідження є сучасні підходи до оцінки продуктивності розробників програмного забезпечення.

Предметом дослідження є системи що інтегрується в процес розробки програмного забезпечення, виконує збір показників продуктивності, виконує розрахунок продуктивності розробників програмного забезпечення.

Мета роботи полягає в розробці прототипу системи розрахунку продуктивності, отриманні результатів роботи системи на основі реалістичних вхідних даних, оцінці отриманих результатів та формуванні висновків.

Наукова новизна полягає у використанні комплексного підходу оцінки продуктивності розробників програмного забезпечення, результатом застосування якого має бути покращення умов праці та підвищення продуктивності роботи команд розробників програмного забезпечення.

Ключові слова: ПРОДУКТИВНІСТЬ РОЗРОБКИ, РОЗРОБНИК ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, СЕРЕДОВИЩЕ РОЗРОБКИ, ПОКАЗНИКИ ПРОДУКТИВНОСТІ, PYTHON, ООП, БАЗА ДАНИХ

Список публікацій:

Криволап В.В. Розробка системи комплексного оцінювання продуктивності розробників програмного забезпечення / В.В. Криволап, Т.В. Алтухова // «ТАК»: телекомунікації, автоматика, комп’ютерно-інтегровані технології: Всеукр. наук.-практ. конф. молодих вчених, 5-6 грудня 2023 р., м. Луцьк *(прийнята до публікації)*

ABSTRACT

Kryvolap V. V. Research and design of an intelligent system for evaluating the productivity of software developers / Graduation qualifying work for obtaining an educational degree «Master» in the specialty 121 Software engineering. – State Higher Educational Establishment «DonNTU», Lutsk, 2023.

The object of the work is modern approaches to assessing the productivity of software developers.

The subject of the work is a system that integrates into software development processes, collects performance indicators, and calculates the productivity of software developers.

This work aims to develop a prototype of a productivity calculation system, obtain the results of the system based on realistic input data, evaluate the results, and draw conclusions.

The scientific novelty lies in using a comprehensive approach to assessing the productivity of software developers, which should result in improved working conditions and increased productivity of software development teams.

Keywords: DEVELOPMENT PRODUCTIVITY, SOFTWARE DEVELOPER, DEVELOPMENT ENVIRONMENT, PERFORMANCE INDICATORS, PYTHON, OOP, DATABASE

The list of publications:

Kryvolap V.V. Development of a system for comprehensive assessment of software developers' productivity / V.V. Kryvolap, T.V. Altukhova // "TAK": telecommunications, automation, computer-integrated technologies: All-Ukrainian Scientific and Practical Conference of Young Scientists, 5-6 December 2023, Lutsk *(accepted for publication)*.

ЗМІСТ

ВСТУП 8

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ФОРМУЛЮВАННЯ ЗАВДАНЬ ДОСЛІДЖЕННЯ 10

1.1 Актуальність необхідності розробки системи оцінювання продуктивності розробників програмного забезпечення 10

1.2 Огляд літератури, праць на тему дослідження 11

1.3 Огляд існуючих рішень щодо оцінювання продуктивності розробників програмного забезпечення. 16

1.4 Формулювання мети і очікуваних результатів дослідження 20

1.5 Висновки за розділом 1 21

РОЗДІЛ 2 РОЗРОБКА МОДЕЛІ І РЕАЛІЗАЦІЯ СИСТЕМИ ОЦІНЮВАННЯ 23

2.1 Формулювання вимог 23

2.2 Математичне обґрунтування визначення продуктивності 24

2.3 Вибір і спосіб врахування метрик для розрахунку продуктивності 26

2.4 Розробка високорівневої архітектури системи 28

2.5 Обґрунтування вибору мови програмування 30

2.6 Вибір і налаштування середовища розробки 32

2.7 Розробка логіки сервісного додатку 32

2.8 Організація збереження даних 40

2.9 Варіанти використання функціоналу системи, клієнтські додатки 46

2.10 Висновки за розділом 2 47

РОЗДІЛ 3 ДОСЛІДЖЕННЯ РОЗРОБЛЕНОЇ СИСТЕМИ 48

3.1 Формалізація реалістичних тестових даних 48
3.2 Аналіз результатів роботи системи 51
3.3 Висновки за розділом 3 61
ВИСНОВКИ 63
ЛІТЕРАТУРА 65
ДОДАТОК А Зауваження нормоконтролера 70
ДОДАТОК Б Лістинг коду реалізації логіки роботи сервісу 71
ДОДАТОК В Лістинг коду модулів збереження даних 86
ДОДАТОК Г Лістинг коду використання програмного інтерфейсу сервісу для конфігурування, розрахунку продуктивності 92
ДОДАТОК Д Презентаційні матеріали 95

ВСТУП

Кількість програмного забезпечення що розробляється і використовується постійно збільшується. Розробка програмного забезпечення потребує витрат значних ресурсів. Продуктивність роботи розробників суттєво впливає на вартість розробки, терміни реалізації, успішність проєктів. Таким чином, оцінка продуктивності розробників програмного забезпечення є важливим елементом процесу розробки програмного забезпечення.

В загальному визначені продуктивність розробки програмного забезпечення це відношення кількості отриманого програмного забезпечення до кількості витрачених ресурсів [1]. Однак продуктивність розробки це комплексне поняття [2], існують різні методики розрахунку продуктивності [3], [4], [5] і з часом з розвитком технологій і методів розробки, методи оцінки продуктивності також змінюються. В сучасних умовах, для повноцінної оцінки продуктивності необхідно враховувати багато показників, як кількісних так і якісних.

Сучасні технології дозволяють розробникам програмного забезпечення взаємодіяти віддалено [6]. І при цьому інструменти що використовуються, такі як системи керування проєктами/завданнями, системи контрою версій вихідного коду, системи постійної інтеграції, середовища розробки, тощо, мають функціонал фіксації інформації про активність розробників [7]. Це створює можливості для збору такої інформації і побудови систем оцінювання продуктивності розробників програмного забезпечення.

Метою виконання цієї роботи є обґрунтування, дослідження та проєктування інтелектуальної системи оцінювання продуктивності розробників програмного забезпечення задля оптимізації та підвищення ефективності розробки проєктів.

В рамках виконання цієї магістерської роботи аналізуються розвиток підходів до оцінювання продуктивності розробників програмного забезпечення. В результаті виконаного аналізу обираються сучасні методи оцінки

продуктивності. Пропонується математична модель для розрахунку числового визначення продуктивності. Обираються методики і технології реалізації системи оцінювання продуктивності. Розробляється прототип системи.

Для тестування розробленої системи обираються реальні проєкти з відкритим вихідним кодом. Система конфігурується для збору інформації про активність розробників програмного забезпечення в цих проєктах. Отримані системою результати аналізуються і надаються висновки та пропозиції для подальшого розвитку.

Розроблена система може бути інтегрована в процес розробки програмного забезпечення і застосована для аналізу продуктивності програмістів. Результат роботи системи має застосовуватись для надання зворотного зв’язку, покращення процесів розробки, планування робіт.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ФОРМУЛЮВАННЯ ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Актуальність необхідності розробки системи оцінювання продуктивності розробників програмного забезпечення

Розробка програмного забезпечення це тривалий і затратний процес. Основною складовою цього процесу є розробники програмного забезпечення. Своєчасна і постійна оцінка продуктивності розробників є важливим механізмом забезпечення успішності проектів. Безперервне оцінювання продуктивності дозволяє ідентифікувати і запобігати втратам і помилкам на ранніх етапах.

Оцінка продуктивності розробників програмного забезпечення є відповідальністю різного рівня менеджерів проектів. Якісний менеджмент проекту розробки програмного забезпечення відповідно до дослідження [8] є одним з головних факторів мотивації розробників. Таким чином впровадження системи оцінювання продуктивності підвищує якість управління проектами і як наслідок через мотивацію розробників сприяє збереженню та підвищенню продуктивності.

Складність і розмір програмного забезпечення постійно зростають, змінюються технології та методи розробки. На початку становлення інженерії програмного забезпечення були спроби розрахунку оцінки продуктивності розробників і витрат на розробку програмного забезпечення по аналогії з методами промислового виробництва [9]. Але ці спроби вже на той час не були точними і вдалими. Системи оцінки продуктивності мають розвиватися одночасно з розвитком технологій і методів розробки програмного забезпечення.

Більшість сучасних проектів розробки програмного забезпечення потребує роботи команди. Системи оцінки продуктивності розробників мають враховувати

командний підхід і оцінювати як безпосередні показники ефективності – кількість коду, якість коду так і показники командної взаємодії – якість комунікації, своєчасність комунікації, змістовність, тощо.

1.2 Огляд літератури, праць на тему дослідження

Методи контроля якості програмного забезпечення, керування процесом розробки, продуктивності розробників почали розвиватися майже одночасно з самими методами розробки програмного забезпечення. Цій темі присвячено багато книг, статей та досліджень. Книги що використовувалися в даному дослідженні наведено в приблизно хронологічному порядку. Це дозволяє прослідкувати розвиток питання.

Книга [9] Фредеріка Брукса «The mythical man-month essays on software» вперше опублікована в 1975 році і витримала багато перевидань разом з доповненнями. Даний рукопис є класичною працею з продуктивності, комунікацій та управління проектами розробки програмного забезпечення. Фредерік Брукс в своїй науковій роботі стверджує що розробка програмного забезпечення є складним завданням, для виконання якого треба знати і враховувати його особливості. В книзі пропонуються рішення збереження продуктивності розробки при збільшені і ускладненні проектів. Не дивлячись на застарілість термінології, книга залишається актуальною і сьогодні.

Наступни рукопис [10] Тома ДеМарко «Controlling Software Projects: Management, Measurement & Estimation», 1982, є класичною працею з управління проектами розробки програмного забезпечення. В книзі розглядаються питання контролю, вимірювання та оцінки. Пропонуються методи засновані на кількісних оцінках. Пропонуються моделі управління програмними проектами. В книзі обговорюються також теми важливості оцінки продуктивності програмістів, методи оцінювання. Пропонується оцінювати кількість написаного коду,

кількість виявлених дефектів, витрати часу, тощо. Не дивлячись на те що книга написана давно, методи описані в ній можуть бути застосовані для часткової оцінки продуктивності програмістів.

В своїй роботі [11] Kurtel, Kaan «Quantifying Productivity of Individual Software Programmers: Practical Approach» взапропонував практичне застосування методів, які розглядаються в [10]. В статті пропонується використання методу розрахунку продуктивності розробників програмного забезпечення на основі кількості доданих рядків програмного коду, витраченого часу, важливості виправленого дефекту. На базі цих метрик формується зважена оцінка продуктивності. Розрахунок виконується для двох різних реальних проектів.

Книга [12] авторів Тома Демарко та Тимоті Лістера «Peopleware: Productive Projects and Teams» також є класичною науковою роботою з керування проектами розробки програмного забезпечення, в якій вони діляться своїм успішним досвідом. В рукописі звертається увага на те, що в багатьох випадках проблеми в розробці є соціальними, а не технічними. Однією основних ідей, які висуваються авторами, є те, що завдання керівника не в тому, щоб змусити людей працювати, а в тому щоб створити їм умови для ефективної роботи. Система моніторингу і оцінювання продуктивності може підвищити обізнаність, об’єктивність, оперативність реагування та якість роботи керівника, що, в свою чергу, створить можливість вчасно допомагати розробникам.

В наступній науковій статті [13] авторами Фентон, Норман І, та Мартін Ніл «Метрики програмного забезпечення: Успіхи, невдачі та нові напрямки» пропонується для оцінки використання метрик програмного забезпечення на момент 1999 року, пропонує більш складний комплексний метод використання метрик. В статті стверджується що на момент написання статті в індустрії в більшості застосовуються метрики розроблені в 1960х роках, такі як кількість строк коду, час, щільність помилок. Приводиться історія наукових досліджень та практик пов’язаних з метриками програмного коду. В своєму досліджені автори пропонують використання Баєсових мереж для аналізу метрик та прийняття

управлінських рішень, при цьому стверджуючи що використання базових метрик є обґрунтованим і на момент підготовки статті.

В статті [8] «Що ми знаємо про мотивацію розробників» авторів Холл Т., Шарп Х., Бічем С., Бадду Н. та Робінсон Х., 2008, автори описують результати систематичного аналізу 92 робіт з мотивації розробників програмного забезпечення. Головні мотивуючі фактори з кількістю згадувань більше 10 наведені в табл. 1.1.

Таблиця 1.1 – Основні фактори мотивації розробників програмного забезпечення

Фактор мотивації	Приклади мотивуючих практик	Кількість досліджених робіт
Визначеність завдань	Зрозумілі цілі, персональна зацікавленість, обмежені рамки робіт.	20
Якісний менеджмент	Наявність підтримки від керівництва, практики побудови команди, якісні комунікації.	16
Співпраця	Залученість до справ компанії, взаємодія з іншими.	16
Кар’єрні можливості	Можливості покращення навичок, зрозумілість кар’єрних перспектив	15
Різноманітність роботи	Застосування всіх навичок розробника, участь в різноманітних завданнях	14
Почуття приналежності	Підтримуючі відношення в команді	14
Винагорода та заохочення	Зростання платні за заохочення за високу продуктивність	14
Визнання	Акцент уваги на гарній продуктивності, добре зробленій роботі	12

Продовження таблиці 1.1

Фактор мотивації	Приклади мотивуючих практик	Кількість досліджених робіт
Технічна складність	Робота не є буденною, є технічно складні виклики.	11
Задоволення потреб розвитку	Наявність можливостей для навчання.	11
Зворотній зв’язок	Наявність зворотного зв’язку від керівництва та колег	10

Такі мотивуючі фактори як «Якісний менеджмент», «Визнання», «Зворотній зв’язок» можуть бути ефективно досягнуті шляхом впровадження систем оцінки продуктивності. В роботі стверджується що мотивація безпосередньо впливає на продуктивність. Автори виділяють внутрішню і зовнішню мотивацію. До зовнішніх мотивуючих факторів відносяться такі як якісний менеджмент, зворотній зв’язок.

Робота [14] «Показники розробки програмного забезпечення» автора Ніколетт Дейв присвячена метрикам які можуть бути використані в процесі розробки програмного забезпечення. Автор стверджує що метрики мають три функції – інформування, діагностика та мотивування. Метрики необхідно обирати в залежності від особливостей програмного проекту. В роботі метрики поділені на групи. Розглядаються метрики для керування проєктами. Розглядаються технічні метрики які отримуються з статичного аналізу коду, вимірювання продуктивності програм, історії змін коду, рівня покриття тестами, тощо. Розглядаються метрики людського самопочуття. Але автор пропонує проводити опитування членів команди для отримання таких метрик. Крім розгляду метрик автор застерігає від частих помилок – не треба розглядати людей як ресурс, не треба зациклюватися на самих цифрах, але приймати рішення. Автор пропонує практики застосування метрик для покращення показників.

В статті [15] The SPACE of Developer Productivity: There's more to it than you think 2021 розкривається найбільш сучасний підхід до оцінювання продуктивності розробника. Автори стверджують, що за десятки років не дивлячись на інтенсивне вивчення питання продуктивності розробників програмного забезпечення, накопичення знань і досвіду про вимірювання продуктивності, застосування вимірювань і навіть саме визначення продуктивності залишаються розмитими і не точними. В підготовці статті приймали участь велика кількість авторів – Ніколь Форсгрен з GitHub, Маргарет-Енн Сторі з Університету Вікторії, Чандра Мадділа, Томас Циммерман, Брайан Хоук та Дженна Батлер з Microsoft Research. Стаття базується на багатьох інших роботах, на які автори часто посилаються. В статті стверджується що задоволення від роботи та продуктивність значно пов’язані, розвінчуються міфи про продуктивність, сформовані за тривалий час, пропонується підхід оцінювання продуктивності з застосуванням багатовимірних метрик задоволення, активності, комунікацій, взаємодії. Приклад метрик для оцінки продуктивності розробників програмного забезпечення що пропонуються в статті наведено в табл. 1.2.

Таблиця 1.2 – Приклад метрик оцінки продуктивності відповідно до SPACE Framework

Задоволення і благополуччя	Продуктивність (результат процесу)	Активність (кількість дій або результатів)	Комунікація та співпраця	Ефективність та потік
1	2	3	4	5
Персональний рівень				
Задоволення процесом роботи. Задоволення оглядом коду.	Швидкість огляду коду.	Кількість оглядів коду. Робочій час. Кількість рядків.	Оцінка огляду коду. Якість обговорень. коду. Обмін знаннями.	Час огляду коду. Відсутність затримок у відповідях.

Продовження таблиці 1.2

Задоволення і благополуччя	Продуктивність (результат процесу)	Активність (кількість дій або результатів)	Комунікація та співпраця	Ефективність та потік
1	2	3	4	5
Рівень команди або групи				
Задоволення командою.	Швидкість огляду коду. Регулярність випусків.	Кількість завершених проміжних версій.	Якість обговорень. Доступ до знань	Час огляду коду. Поширення знань.
Рівень система або проекту				
Задоволення процесами.	Продуктивність огляду коду. Задоволення користувача.	Частота публікацій.	Поширення знань. Доступ до знань.	Час огляду коду. Продуктив-ність процесів.

1.3. Огляд існуючих рішень щодо оцінювання продуктивності розробників програмного забезпечення.

Існує велика кількість інструментів для вимірювання показників роботи програмістів. Більшість інструментів є вузькоспеціалізованими, розроблялися для потреб окремих організацій чи моделей діяльності, та пропонують врахування обмеженої кількості показників. Інструменти і методи комплексної оцінки почали розроблятися нещодавно.

Одним з перших типів інструментів для контролю активності розробників програмного забезпечення стали додатки обліку/відстеження робочого часу. Такі додатки зазвичай пропонують наступні функції. Автоматичне і ручне врахування робочого часу. Фіксація активності клавіатури та інших пристроїв. Періодичне збереження копій екрану для контролю відволікаючої активності. Відстеження звернень до інтернет ресурсів. Формування звітів про витрачений час. Цей тип додатків часто використовується при почасовій оплаті праці. Прикладами таких

додатків є Time Doctor, Clockify, TimeAnalytics. Використання таких підходів може мати як позитивний так і негативний вплив, підвищувати рівень стресу і в результаті призводити то падіння продуктивності [17].

Ще одними інструментами є професійні середовища розробки програмних продуктів. Вони мають вбудовані інструменти розрахунку метрик програмного коду. Частина з цих метрик можуть бути використані для розрахунку продуктивності розробників програмного забезпечення. Наприклад, популярне середовище розробки Microsoft Visual Studio дозволяє розраховувати наступні метрики коду:

а) індекс ремонтопридатності – обчислює значення яке може бути в диапазоні 0 .. 100 і відображає легкість обслуговування коду; більше значення відповідає кращій ремонтопридатності;

б) цикломатична складність – розраховується як кількість розгалужень в алгоритмі;

в) глибина успадкування – розраховується на основі підрахунку кількості класів що успадковуються один від одного в ієрархіях класів;

г) зв'язок класів – розраховується на основі різноманітних зв’язків між класами; при хорошому дизайні передбачається що класи між собою слабо зв’язані, на відміну від реалізації в середині класів;

д) кількість рядків вихідного коду – показує точну кількість рядків вихідного коду, присутніх у файлі, включно з порожніми рядками;

е) кількість рядків коду що виконується – показує приблизну кількість рядків або операцій коду що виконується; це підрахунок кількості операцій у виконуваному коді.

Зміни цих метрик між різними версіями коду можуть бути використані для розрахунку оцінки продуктивності розробника програмного забезпечення з точки зору об’єму і якості виконаної роботи.

Якість роботи як частина загальної оцінки продуктивності розробника програмного забезпечення більш детально може бути оцінена з використанням додатків для статичного аналізу коду. Одним з найбільш популярних додатків для

цих задач є SonarQube. SonarQube це інструмент для автоматичного оцінювання якості коду. Він аналізує код на наявність потенційних помилок, вразливостей та порушень рекомендацій щодо якості коду. SonarQube підтримує аналіз коду на відповідність рекомендаціям OWASP (Open Web Application Security Project), MISRA (Motor Industry Software Reliability Association), CWE (Common Weakness Enumeration). Кількість вразливостей та небезпечність вразливостей що з'явилася/зникла в результаті змін внесених розробником може використовуватись як метрика для розрахунку продуктивності.

Розвиток систем спільної роботи над створенням програмних продуктів дозволяє також розглядати їх в якості джерел інформації для оцінки продуктивності програмістів. На сьогоднішній день найбільш популярними системами спільної розробки є GitHub, GitLab, Bitbucket. Ці системи побудовані над системами контролю версій коду. Відповідно, вони дозволяють отримати наступні метрики для оцінки продуктивності розробників:

а) зміни в коді – кількість рядків коду що були додані, видалені, змінені; метрика дозволяє отримати уяву про об'єм виконаної роботи без урахування складності;

б) кількість збережень коду – кількість разів внесення змін в репозиторій коду; метрика дозволяє оцінити наскільки робота;

в) кількість запитів на об'єднання версій коду – кількість разів внесення змін в репозиторій коду які потребували огляду іншими членами команди;

г) кількість оглядів коду – метрика показує скільки разів розробник виконував огляд коду відповідно до запитів інших розробників;

д) час відгуку – через якій проміжок часу розробник відповідає на питання, створені завданні, запити на об'єднання коду;

е) час циклу розробки – час від створення гілки коду розробником до її закриття;

ж) кількість вирішених завдань.

Це не повний перелік метрик які доступні в системах спільної роботи над створенням програмних продуктів.

Наступні досліджені системи є комплексними системами моніторингу показників продуктивності. Вони не виконують розрахунку продуктивності у числовому визначенні. Ці системи з’явилися відносно недавно і розвиваються. Особливістю цих систем є можливість інтеграції з програмним забезпеченням яке використовується для побудови процесу створення програмних продуктів – системи обліку завдань, системи спільної розробки коду, системи автоматизованої збірки і публікації додатків, системи контролю якості коду, тощо.

Система Pluralsight Flow [18] це інструмент для вимірювання продуктивності розробників, який базується на даних з систем контролю версій. Flow підтримує інтеграцію з GitHub, GitLab, BitBucket, Jira, тощо. Цей інструмент допомагає командам отримати глибокий аналіз роботи їх розробників, виявити слабкі місця і підвищити загальну продуктивність. Слоганом системи є «Новий та вдосконалений Pluralsight Flow проливає світло на те, де і чому виникають вузькі місця в процесі доставки програмного забезпечення.» Система Pluralsight Flow дозволяє отримувати наступні показники:

а) продуктивність (Commit Volume) – система аналізує кількість комітів розробника за певний період часу;

б) робочий час (Work in Progress) – система відстежує, скільки завдань розробник виконує одночасно;

в) частота кодування (Coding Days) – показник того, скільки днів у тиждень розробник проводить за кодуванням;

г) співвідношення відновлення (Churn) – відсоток коду, який розробник написав і потім видалив чи змінив у той же період;

д) тривалість кодового рецензування (PR Review Time) – час, який потрібен для проведення огляду коду і його схвалення;

е) взаємодія з командою (Collaboration) – аналіз взаємодії розробника з іншими членами команди;

ж) помилки та спроби їх усунення (Rework Loops): частота та обсяг корекцій, які розробник робить на основі зворотнього зв'язку після рецензування

коду;

з) тренди та відхилення – відстеження зміни в показниках продуктивності з часом.

и) командна динаміка – визначення ефективності взаємодії всередині команди, що допомагає виявляти потенційні проблеми або зони для покращення.

Ще одним інструментом, який дозволяє виконувати комплексний аналіз продуктивності розробників програмного забезпечення є AnalyticsVerse [19]. AnalyticsVerse пропонує широкий спектр можливостей для моніторингу і аналізу продуктивності розробників програмного забезпечення. Ці можливості дозволяють менеджерам розробки отримувати глибоке розуміння того, як працюють їхні команди, і виявляти можливості для поліпшення продуктивності. Перелік основних функцій:

а) збір інформації з зовнішніх систем;

б) відстеження часу витраченого розробниками на різні завдання, такі як написання коду, тестування та зустрічі;

в) відстеження коду – кількість коду, кількість дефектів у коді;

г) відстеження активності – кількість коммітів в репозиторій, кількість відкритих і закритих задач.

1.4 Формулювання мети і очікуваних результатів дослідження

Метою цієї роботи є обґрунтування, дослідження та проєктування інтелектуальної системи оцінювання продуктивності розробників програмного забезпечення задля оптимізації та підвищення ефективності розробки проєктів.

В процесі виконання роботи необхідно проаналізувати існуючі методи, реалізовані програмні застосунки оцінювання продуктивності розробників програмного забезпечення. Вивчити наукові праці та джерела присвячені цій темі. Виявити проблеми та обмеження існуючих методів. Обґрунтувати

необхідність розробки нового метода і системи.

Запропонувати актуальний комплексний метод оцінювання продуктивності. Визначити ефективні метрики, методи їх поєднання. Взяти до уваги технічні метрики, якісні метрики, комунікативні метрики, тощо. Розробити математичну модель розрахунку продуктивності.

Обрати і обґрунтувати метод та інструменти розробки інтелектуальної системи оцінювання продуктивності. Описати вимоги до системи. Розробити систему з можливостями конфігурування, поєднання з системами і середовищами розробки, інтеграції в процес розробки і розгортання програмного забезпечення, застосування в умовах обмеженого доступу.

Розробити методику тестування системи. Дослідити ефективність та коректність роботи системи на реальних проектах з розробки програмних продуктів. Провести аналіз результатів та сформувати висновки по результатам виконаної роботи.

1.5 Висновки за розділом 1

Задача оцінки продуктивності розробників програмного забезпечення стала актуальною одночасно з задачами створення програмних продуктів. Різні методи оцінки продуктивності базувалися на розрахунку кількісних показників, таких як кількість рядків коду, витрачений час. Такий обмежений аналіз не дозволяв якісно оцінювати продуктивність розробників програмного забезпечення, планувати процес розробки, обирати шляхи покращення продуктивності.

Розвиток інструментів для створення програмного забезпечення, інструментів, інструментів командної роботи, розвиток мереж створили можливості для отримання більшої кількості метрик. Акценти почали зміщуватися на більш комплексні аспекти продуктивності, такі як якість коду,

співпраця в команді та відгук на зміни вимог. В наукових дослідженнях з’явився інтерес до вивчення взаємодії людини з комп'ютером, а також психологічних і культурних аспектів роботи в команді.

В теперішній час оцінка продуктивності розробників програмного забезпечення стає все більш багатогранною і зосередженою на соціальних навичках. Зокрема, навички спілкування, спроможність до навчання та адаптації, а також вміння працювати в команді набувають вирішального значення. В сучасній науковій літературі активно обговорюються питання створення ефективних команд, взаємодії між розробниками та іншими учасниками проекту, а також використання інструментів для автоматичної оцінки якості коду та продуктивності.

Інструменти для оцінки продуктивності розробників програмного забезпечення також пройшли значний шлях розвитку. Від простих додатків для моніторингу активності, підрахунку часу інструменти розвинулися до складних систем збору і аналізу інформації про продуктивність з багатьох джерел. Такі системи продовжують активно розвиватись і вдосконалюватися. А напрямок останніх досліджень [15] підтверджує що створення сучасних систем комплексного оцінювання продуктивності розробників програмного забезпечення є актуальним завданням.

РОЗДІЛ 2
РОЗРОБКА МОДЕЛІ І РЕАЛІЗАЦІЯ СИСТЕМИ ОЦІНЮВАННЯ

2.1 Формулювання вимог

Розробка програмного забезпечення має починатися з формулювання вимог. Вимоги до програмного продукту це опис того що має бути розроблено, як система що реалізується має поводитися, які в системи мають бути властивості [20]. Існують різні типи вимог: бізнес вимоги, вимоги відповідно до стандартів, функціональні вимоги, вимоги до якості, системні вимоги, вимоги до процесу, користувацькі вимоги, тощо. При розробці складного програмного забезпечення вимоги формулюються і уточнюються на протязі всіх етапів розробки. Для цього впроваджуються методики керування вимогами [21]. В даній роботі система пропонується і розробляється одноосібно, тому розробляються тільки функціональні та системні вимоги.

Функціональні вимоги до системи що розробляється в рамках цієї роботи:

а) інтеграція з сучасними інструментами створення програмних продуктів, інструментами керування процесами розробки;

б) можливість розширення, інтеграції з новими системами без переробки ядра системи;

в) збір і збереження інформації про різноманітні показники роботи розробників програмного забезпечення;

г) розрахунок комплексного значення продуктивності в числовому вираженні, в заданому діапазоні;

д) відображення по запиту інформації про продуктивність розробників в поточний момент і зміни продуктивності за вказаний період;

е) можливість інтеграції системи в процес розробки [22].

Системні вимоги до системи що розробляється в рамках цієї роботи:

а) наявність інтерфейсів для програмного доступу до функціональності системи для інтеграції з іншими системами для побудови процесів керування розробкою програмних проєктів;

б) можливість розгортання системи в поширених сучасних операційних системах – Linux, Windows, macOS;

в) можливість розгортання та використання в хмарному середовищі за допомогою контейнеризації [23].

2.2 Математичне обґрунтування визначення продуктивності

При реалізації системи оцінки продуктивності розробника програмного забезпечення необхідно розробити зручну математичну модель розрахунку. Отримане числове вираження має дозволяти однозначно трактувати продуктивність розробника, порівнювати значення за різні періоди, відстежувати динаміку.

Числове вираження продуктивності має знаходитись в відомому діапазоні значень $[0, .., P_{MAX}]$, де P_{MAX} – максимально можливе (нормоване) значення комплексної продуктивності. Наприклад, може бути прийняте, що комплексна продуктивність розробника має знаходитись в діапазоні [0, .., 100].

Відповідно до [14] продуктивність розробника програмного забезпечення може бути визначена як відношення отриманого результату роботи до затрачених зусиль:

$$P_1 = \frac{R}{E}, \tag{2.1}$$

де P_1 – продуктивність розрахована за одним критерієм оцінки,

R – отримані результати,

E – докладені зусилля.

В якості зусиль в прикладній системі можна використати тільки витрачений на роботу час, бо немає ефективних практичних методів оцінки інтелектуальних витрат, які можуть бути практичними і автоматизованими. Продуктивність має оцінюватись за якийсь визначений один проміжок часу і порівнюватись з продуктивністю за такі самі попередні проміжки часу. Можна сказати, що в цьому випадку $E = 1$ і $P_1 = R$.

Комплексна продуктивність розраховується як середнє значення продуктивностей розрахованих за окремими критеріями. Так як різні результати вимірюються в різних одиницях, то для розрахунку комплексної продуктивності, значення результатів мають бути нормованими. Деякі результати роботи можуть не мати числового вираження, тому якісні значення мають бути ранжовані.

В залежності від типу програмного проекту що розробляється, в залежності від пріоритетів, різні результати роботи можуть мати різну важливість. Тому для розрахунку комплексної продуктивності виникає необхідність застосування вагових коефіцієнтів.

З урахуванням наведених вище правил, значення комплексної продуктивності розробника програмного забезпечення може бути розраховане за формулою 2.2:

$$P_{\text{C}} = \frac{1}{N}\sum_{i=0}^{N-1} W_i \cdot R_{RN_i}, \qquad (2.2)$$

де P_C – числове вираження комплексної продуктивності,

N – кількість показників продуктивності що вимірюються,

W_i – ваговий коефіцієнт i-го показника продуктивності,

R_{RN_i} – ранжоване та нормоване значення i-го показника продуктивності.

Для того щоб числове значення комплексної продуктивності знаходилося в

діапазоні $[0, .., P_{MAX}]$, R_{RN_i} має приймати значення в такому ж діапазоні $[0, .., P_{MAX}]$ і для вагових коефіцієнтів W_i має виконуватися рівняння :

$$\sum_{i=0}^{N-1} W_i = 1. \quad (2.3)$$

2.3 Вибір і спосіб врахування метрик для розрахунку продуктивності

Відповідно до [15] при розрахунку комплексної продуктивності необхідно враховувати метрики різних типів, наприклад, кількісні, комунікативні, якісні. Наявність тих чи інших метрик залежить від обраних технологій та процесу розробки програмного забезпечення. В цій роботі для аналізу обираються метрики які мають бути присутніми в більшості сучасних підходів до розробки.

Базовою кількісною метрикою є кількість доданих рядків коду. Використання цієї метрики для оцінки продуктивності зазнає багато критики [24]. Кількість рядків коду що розробник програмного забезпечення може написати за день залежить від досвіду, розміру проекту, складності і особливостей завдання. Тому використання кількості написаних рядків коду безпосередньо для оцінки продуктивності без врахування цих факторів не є коректним. Але відповідно до [9], тривалі дослідження показують що середня кількість рядків коду за день за тривалий період складає 10. Відповідно до [26] в залежності від складності проекту середня кількість рядків коду за день може складати від 10 до 50. Таким чином показник кількість рядків коду за день можна врахувати виставивши нижню межу, при недосягнені якої система буде генерувати попередження і знижувати значення комплексної оцінки продуктивності. Сучасні системи контролю версій, спільної розробки мають

вбудовані можливості розрахунку кількості доданих, видалених, змінених рядків коду.

Для того щоб оцінити якість комунікацій можна використати таки метрики як регулярність збереження коду в систему управління версіями коду, наявність коментарів при збереженні коду, змістовність коментарів та відповідність коментарів погодженим шаблонам (наприклад, вказування ідентифікатору завдання в рамках якого зроблені зміни коду), своєчасність відправки звітів, тощо – все те що дозволяє команді та керівництву дізнатися про результати роботи і врахувати їх для своєї подальшої роботи. Кількість збережень коду (commit count) в систему управління версіями коду в одиницю часу є базовою комунікативною метрикою. Як і у випадку з кількістю рядків коду, неможливе пряме використання цієї метрики для розрахунку продуктивності. Відповідно до дослідження [27] середня кількість збережень коду складає 3 збереження на тиждень. Це значення може бути використане як порогове.

Третім типом метрик які потрібно врахувати для комплексної оцінки продуктивності розробника програмного забезпечення є оцінки якості коду. До цих метрик можна віднести результати статичного аналізу коду, результати виконання тестів, кількість повідомлень про помилки від користувачів. Якщо над проєктом працює команда розробників, то можливо встановити зв’язок між змінами які вносить кожен розробник, та результатами статичного аналізу коду і результатами виконання тестів. Встановлення безпосереднього зв’язку між кількістю повідомлень від користувачів про помилки і змінами внесеними окремим розробником в багатьох випадках складно або неможливо зробити автоматизованими методами, потрібен аналіз спеціалістом. Виконання тестів потребує налаштування середовища, тестових даних і потребує витрат ресурсів.

На теперішній час статичний аналіз коду є розвиненим інструментом, який дозволяє виявляти стилістичні, алгоритмічні недоліки коду, недоліки які можуть впливати на безпеку, тощо [28]. Для кожної розповсюдженої мови програмування є набір інструментів для статичного аналізу коду. Загально прийнятими назвами інструментів для статичного аналізу коду є static analysis too, code linter.

Деякі інструменти статичного аналізу виконують аналіз з прив’язкою до таких стандартів як OWASP (Open Web Application Security Project), MISRA (Motor Industry Software Reliability Association), CWE (Common Weakness Enumeration).

Результатом роботи статичних аналізаторів коду є надання переліку зауважень до коду що аналізується з вказівкою рівня важливості та рекомендаціями для виправлення. Зміні внесені розробником не мають призводити до збільшення зауважень.

Таким чином в даній роботі для розрахунку комплексної продуктивності розробників програмного забезпечення будуть використані такі показники роботи як кількість рядків коду (кількісний показник), кількість збережень коду в систему спільної роботи (комунікативний показник), кількість зауважень отриманих при виконанні статичного аналізу коду (якісний показник).

2.4 Розробка високорівневої архітектури системи

Відповідно до сформованих вимог розробляється архітектура системи. Ключовими поняттями архітектури програмного забезпечення є компоненти і комунікації [29]. Архітектура може бути представлена у вигляді формальних або неформальних діаграм та додаткових текстових специфікацій. Для створення формальних діаграм може бути використана уніфікована мова моделювання (UML) [30]. Графічне представлення архітектури системи наведене на рис. 2.1.

Як можна побачити з рисунку, система оцінювання продуктивності складається з системного додатку, бази даних та декількох клієнтських додатків. Можуть бути розроблені такі клієнтські додатки як графічний інтерфейс користувача та інтерфейс командного рядку. Клієнтські додатки системи приєднуються до сервісного додатку через «Клієнтський інтерфейс 1».

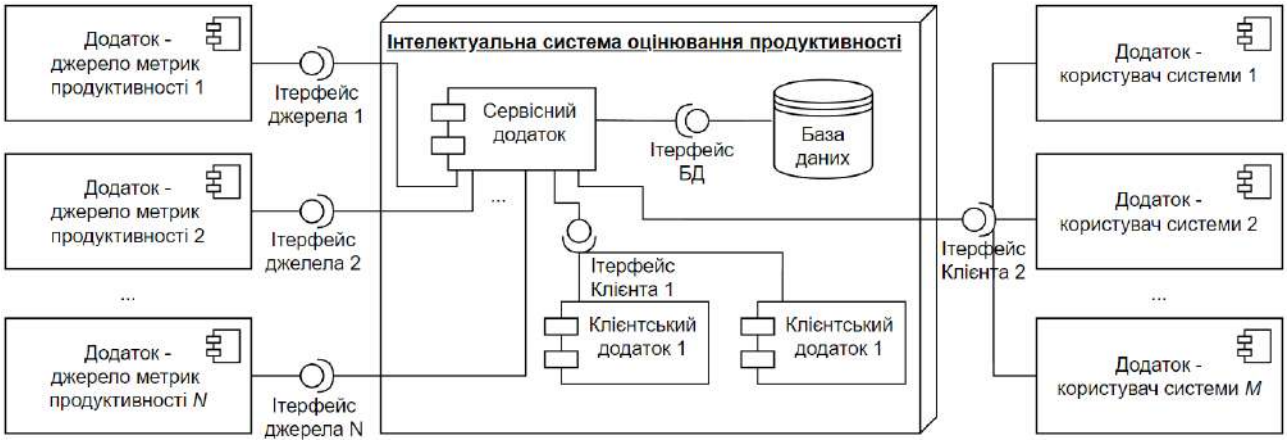


Рисунок 2.1 – Графічне представлення високорівневої архітектури інтелектуальної системи оцінювання продуктивності розробників програмного забезпечення

Для збору метрик продуктивності розробників програмного забезпечення системний додаток має мати можливість приєднуватися до зовнішніх додатків які є частиною інфраструктури розробки програмного забезпечення. Кожен з цих додатків має свій окремий інтерфейс. Таким чином, для відповідності вимогам, сервісний додаток системи має надавати можливість розширення функціоналу. Це може бути зроблено за рахунок розробки підсистеми розширень.

Ще однією частиною системи є база даних. Єдиним компонентом який взаємодіє з базою даних є сервісний додаток. В базі даних мають зберігатися конфігурації для під’єднання до додатків – джерел метрик, отримані значення метрик, розрахунки продуктивності, інформація про розробників, тощо.

Відповідно до вимог система має мати можливість інтегруватися в інфраструктуру середовища розробки програмного забезпечення. Це можливість реалізується шляхом розробки програмного інтерфейсу «Інтерфейс користувача 2».

2.5 Обґрунтування вибору мови програмування

Вибір мови програмування є важливим етапом розробки програмного забезпечення. Від вибору мови програмування будуть залежати як вартість розробки системи так і вартість експлуатації системи [31]. Мови програмування поділяються на мови програмування загального призначення, наприклад, C++, C#, Python, і спеціалізовані мови, наприклад, R, MATLAB. Для реалізації системи що розробляється в рамках цієї роботи, відповідно до сформованих вимог і запропонованої архітектури необхідно використання мови програмування загального призначення.

При виборі мови програмування необхідно врахувати наступні критерії:

а) тип додатку – в залежності від типу додатку – сервіс, веб додаток, настільний додаток, вбудований додаток, формулюються вимоги до мови програмування;

б) продуктивніть розробки – мови програмування мають різник синтаксис, стандартні бібліотеки, систему типів; кількість коду що необхідно написати для реалізації одного й того ж самого функціоналу на різних мовах програмування може сильно відрізнятися, наприклад, в Python об’єм коду в 6 разів менший ніж в C і в 2.4 рази менший ніж в Java [26];

в) досвід розробників, які будуть розробляти програмне забезпечення – програмісти які працюють з обраною мовою програмування три роки будуть на 30% ефективніші за програмістів з таким же досвідом але на інших мовах програмування [26];

г) продуктивність виконання – багато особливостей мов програмування впливають на швидкість виконання додатків що розробляються з їх використанням; найбільш важливими для швидкості виконання є такі характеристики, як спосіб виконання та вид типізації; наприклад, в мовах C, C++ [32] реалізовані компіляція в машинний код та статична, сильна типізація, це забезпечує найвищу можливу продуктивність виконання; в мовах C# [33], Java

реалізовані компіляція в проміжний код та статична сильна типізація; в мові програмування Python [34] реалізована безпосередня інтерпретація коду перед виконанням, за замовченням реалізована динамічна типізація, з версії 3.0 доступна статична типізація [35];

д) інтеграція – чи потрібно інтегрувати додаток що розробляється з уже існуючими в середовищі додатками;

е) переносимість – чи потрібно запускати додаток що розробляється в різних операційних системах;

ж) популярність – чи є розвинена спільнота, необхідні для розробки бібліотеки, документація, книги;

з) безпека;

и) доступність розробників.

Для виконання даної роботи важливими критеріями при виборі мови програмування є швидкість розробки, доступність для всіх сучасних операційних систем, розповсюдженість мови програмування. Менш важливим критерієм є швидкість виконання розробленого додатку, бо відповідно до специфіки додатку більшість часу буде витрачатися на комунікації між додатками.

Відповідно до вимог і особливостей проекту для реалізації проекту обирається мова програмування Python. Python – мова загального призначення, яка дозволяє розробляти код з використанням об’єктно-орієнтованого, імперативного, функціонального та процедурного підходів.

На момент розробки проекту вона за даними ресурсу Tiobe [36] є найбільш популярною мовою. Інформацію про популярність мов наведено на рис. 2.2. Python використовується для задач автоматизації, створення веб додатків, створення AI додатків, математичних обчислень, тощо. До переваг мови програмування Python можна віднести продуктивність розробки та розвиненість бібліотек та спільноти. До недоліків Python можна віднести низьку продуктивність виконання у деяких сценаріях, динамічна типізація.

Oct 2023	Oct 2022	Change	Programming Language	Ratings	Change
1	1		Python	14.82%	-2.25%
2	2		C	12.08%	-3.13%
3	4	↑	C++	10.67%	+0.74%
4	3	↓	Java	8.92%	-3.92%
5	5		C#	7.71%	+3.29%
6	7	↑	JavaScript	2.91%	+0.17%
7	6	↓	Visual Basic	2.13%	-1.82%
8	9	↑	PHP	1.90%	-0.14%

Рисунок 2.2 – Популярність мов програмування за даними ресурсу tiobe.com

2.6 Вибір і налаштування середовища розробки

Ефективність розробки і навіть якість вихідного коду багато в чому залежить від обраного середовища розробки. Для обраної мови програмування Python, існує декілька ефективних середовищ розробки [37]. Одним з найпопулярніших на теперішній момент є Microsoft Visual Code з розширеннями для мови програмування Python – Python, Pylint, Mypy Type Checker [38]. Така конфігурація дозволяє компенсувати недоліки мови програмування Python – відсутність контролю типів змінних, контролювати якість коду, ефективно виконувати налагодження коду, отримувати підказки, в тому числі з відображенням онлайн ресурсів. Саме така конфігурація середовища обрана для реалізації системи в рамках цієї роботи.

2.7 Розробка логіки сервісного додатку

Головною частиною системи оцінки продуктивності розробників

програмного забезпечення є сервісний додаток. Він має виконувати основні функції системи – збір інформації про метрики, розрахунок значення комплексної продуктивності, збереження та надання доступу до інформації.

Предметна область системи включає багато елементів які мають взаємодіяти між собою для отримання результатів. В таких випадках зручно застосовувати об’єктно орієнтовану методику програмування при якій елементам системи відповідають класи і структури класів в коді додатку [39]. Нижче наведено опис базових класів сервісного додатку.

Точкою входу в логіку роботи сервісного додатку є клас Service. Цей клас реалізує високорівневий програмний інтерфейс для роботи з системою. Реалізація відповідає шаблону проєктування «Фасад» [40]. Через цей клас можна отримати доступ до даних системи, запустити розрахунок продуктивності. В якості параметра конструктор класу отримує екземпляр класу Storage, для доступу до бази даних. Така реалізація відповідає шаблону проєктування «Інверсія залежностей» [41].

Клас Storage відповідає за роботу з базою даних. Клас є абстрактним для того щоб була можливість імплементувати нащадків для роботи як з справжньою базою даних для виробничого використання, так і для роботи з імітаціями баз даних в пам’яті для налагодження і перевірки коректності роботи.

Інформація про розробників програмного забезпечення представлена класом Developer. Кожен розробник характеризується іменем, контактними даними, тощо.

Для кожного розробника необхідна можливість налаштувати один чи декілька методів розрахунку показників продуктивності. Ця можливість реалізується шляхом імплементації базового абстрактного класу MetricCalculator. В даній роботі в якості джерела даних обраний GitHub [42], який надає можливість отримати інформацію про кількість рядків коду, кількість збережень через Web API, та отримати безпосередньо код шляхом клонування репозиторію коду. Для розрахунку показників на основі цих джерел добавлені класи-нащадки GitHubApiMetricCalculator, GitHubPylintMetricCalculator. При

конструюванні екземплярів цих класів необхідно передавати параметри підключення до джерела. Для можливості розширення функціональності додатку без зміни коду, для реалізації передачі параметрів додано базовий абстрактний клас MetricSource. Також реалізовано два похідних класа GitHubApiMetricSource, GitHubCloneMetricSource.

Калькулятор показників GitHubApiMetricCalculator реалізує отримання показників – кількість доданих рядків коду, кількість зроблених збережень коду шляхом використання API сервісу GitHub. Калькулятор показників GitHubPylintMetricCalculator реалізує клонування коду з сервера GitHub, аналіз коду з застосуванням інструменту Pylint [43]. Архітектура додатку дозволяє створювати додаткові калькулятори показників з використанням різноманітних технологій спільної розробки даних і для різних мов програмування. Але для обмеження об’єму роботи наведені приклади тільки для інструменту GitHub та мови програмування Python.

Розрахунок значень показників є важливою частиною системи, тому лістинги розрахунку обраних показників наведені нижче. Реалізація розрахунку кількісті рядків коду та кількості збережень коду з використанням GitHub API [44], метод класу GitHubApiMetricCalculator, наведена в лістингу на рис. 2.3.

Реалізація розрахунку показника якості коду Python з використанням клонування коду збереженого на GitHub та інструменту PyLint, метод класу GitHubPylintMetricCalculator, наведена в лістингу на рис. 2.4, 2.5.

Як можна побачити з наведених лістингів 2.1, 2.2, отримання показників продуктивності з різноманітних сучасних джерел не потребує значних зусиль. Такі інструменти як GitHub, git надають високорівневе API.

При необхідності отримання інформації про показники продуктивності програміста з інших джерел можна додавати більше похідних від MetricSource, MetricCalculator класів без необхідності зміни базового коду сервісного додатку. Відсутність необхідності змінювати базовий додаток для розширення функціоналу відповідає одному з SOLID принципів і є методом боротьби зі складністю [45].

```
def calculate(self, since: datetime, until: datetime) -> List[Metric]:
    base_url =
    f"https://api.github.com/repos/{self.metric_source.owner}/{self.metric_
    source.repo}/commits"
    headers = {
        "Authorization": f"token {CdpServiceConfig.GITHUB_TOKEN}",
        "Accept": "application/vnd.github.v3+json"
    }
    params:Dict[str, Any] = {
        "since": since,
        "until": until,
        "author": self.metric_source.login
    }
    resp = requests.get(base_url, headers=headers, params=params,
    timeout=5)
    resp.raise_for_status()
    commits = resp.json()
    loc = 0
    for commit in commits:
        commit_sha = commit["sha"]
        commit_resp = requests.get(f"{base_url}/{commit_sha}",
        headers=headers, timeout=10)
        json_data = commit_resp.json()
        commit_files = json_data.get("files")
        for file in commit_files:
            file_name = file.get("filename")
            if self.metric_source.include and \
                not any(re.match(p, file_name) for p in
                self.metric_source.include):
                continue
            if self.metric_source.exclude and \
                any(re.match(p, file_name) for p in
                self.metric_source.exclude):
                continue
            loc += file.get("additions")
    return [LocMetric(loc, source=self.metric_source),
            CommitMetric(len(commits), source=self.metric_source)]
```

Рисунок 2.3 – Лістинг коду методу розрахунку показників кількості рядків коду, та кількості збережень коду з використанням GitHub API

```
def calculate(self, since: datetime, until: datetime) -> List[Metric]:
 if not os.path.exists(CdpServiceConfig.WORKING_DIR):
        os.makedirs(CdpServiceConfig.WORKING_DIR)
    # Obtain the latest commit for specified period and user.
    headers = {
        "Authorization": f"token {CdpServiceConfig.GITHUB_TOKEN}",
        "Accept": "application/vnd.github.v3+json"
    }
    params:Dict[str, Any] = {
        "since": since,
        "until": until,
        "author": self.metric_source.login
    }
    resp = requests.get(
f"https://api.github.com/repos/{self.metric_source.owner}/{self.metric_so
urce.repo}/commits",
         headers=headers, params=params, timeout=5
    )
    resp.raise_for_status()
    commits = resp.json()
    # Obtain the main branch name
    resp = requests.get(
f"https://api.github.com/repos/{self.metric_source.owner}/{self.metric_so
urce.repo}",
         headers=headers, params=params, timeout=5
    )
    resp.raise_for_status()
    default_branch = resp.json().get("default_branch", None)
    if len(commits) == 0:
        lint_metric = LintMetric(self.__previous_metric.raw_value,
                                 self.__previous_metric.raw_value, True,
                                 source=self.metric_source)
        self.__previous_metric = lint_metric
        return [lint_metric]
    latest_commit_sha = commits[-1]["sha"]
    project_url =
    f"https://github.com/{self.metric_source.owner}/{self.metric_source.r
    epo}.git"
    project_dir = f"{self.metric_source.owner}_{self.metric_source.repo}"
    full_project_dir = os.path.join(CdpServiceConfig.WORKING_DIR,
    project_dir)
    auth_project_url = project_url.replace("https://",
    f"https://{CdpServiceConfig.GITHUB_TOKEN}@")
    # Clone the specified repository.
```

Рисунок 2.4 – Лістинг коду методу розрахунку якості коду Python з використанням клонування з GitHub та інструменту PyLint

```
if os.path.exists(full_project_dir):
    try:
        subprocess.run(["git", "checkout", default_branch],
                       check=True, text=True, capture_output=True,
                       cwd=full_project_dir)
        subprocess.run(["git", "pull"],
                       check=True, text=True, capture_output=True,
                       cwd=full_project_dir)
    except subprocess.CalledProcessError as er:
        print(er.output)
        shutil.rmtree(os.path.join(CdpServiceConfig.WORKING_DIR,
        project_dir))
        subprocess.run(["git", "clone", auth_project_url,
        project_dir],
                       check=True, text=True, capture_output=True,
                       cwd=CdpServiceConfig.WORKING_DIR)
else:
    subprocess.run(["git", "clone", auth_project_url, project_dir],
                   check=True, text=True, capture_output=True,
                   cwd=CdpServiceConfig.WORKING_DIR)
# Check out the latest commit.
subprocess.run(["git", "checkout", latest_commit_sha],
               check=True, text=True, capture_output=True,
               cwd=full_project_dir)
# Calculate pylint metrics
source_dir = full_project_dir
if self.metric_source.source_dir:
    source_dir = os.path.join(source_dir,
    self.metric_source.source_dir)
pylint_args = [source_dir, "--disable=C,R,E0401"]
linter = None
with open(os.devnull, "w", encoding="utf-8") as fnull:
    old_stdout = sys.stdout
    old_stderr = sys.stderr
    sys.stdout = fnull
    sys.stderr = fnull
    try:
        linter = pylint.lint.Run(pylint_args, exit=False)
    finally:
        sys.stdout = old_stdout
        sys.stderr = old_stderr
error_count = sum(linter.linter.stats.by_msg.values())
lint_metric = LintMetric(error_count,
self.__previous_metric.raw_value, source=self.metric_source)
self.__previous_metric = lint_metric
return [lint_metric]
```

Рисунок 2.5 – Продовження лістингу коду методу розрахунку якості коду Python з використанням клонування з GitHub та інструменту PyLint

Запропонована частина архітектури сервісного додатку для представлення інформації про розробника програмного забезпечення у вигляді UML діаграми має вигляд зображений на рис. 2.6.

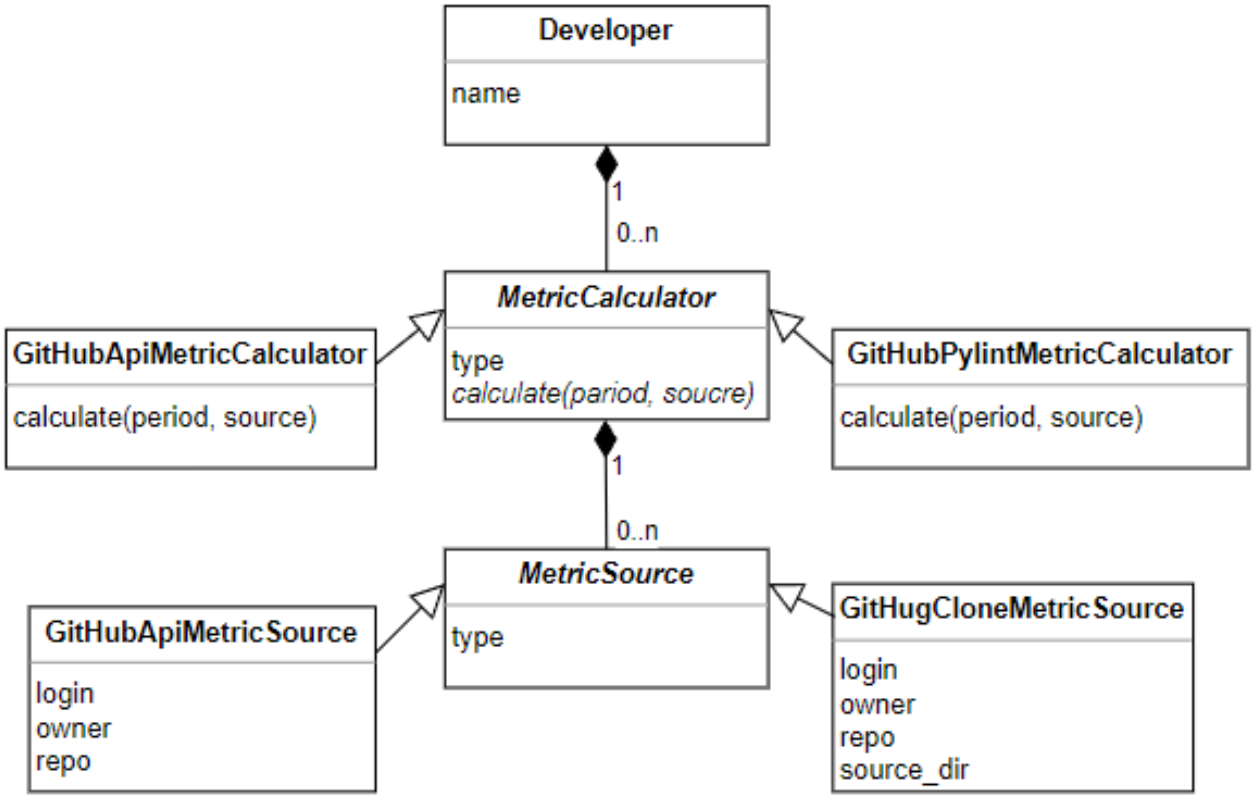


Рисунок 2.6 – UML-діаграма класів для представлення інформації про розробника програмного забезпечення

Запропонована архітектура дозволяє вказувати для розробника необмежену кількість методів розрахунку показників. Це необхідно коли різні типи показників збираються з різних типів джерел і коли розробник працює над декількома проектами одночасно. Ця особливість має бути врахована при реалізації логіки розрахунку значення комплексної продуктивності.

В результаті звернення сервісного додатку до джерел показників мають отримуватися відповідні показники. Для представлення показників в моделі класів доданий абстрактний базовий клас Metric та похідні класи LocMetric, CommitMetric, LintMetric для представлення показників кількості рядків,

кількості збережень та результату статичного аналізу відповідно. Крім збереження значення показника ці класи також реалізують розрахунок нормалізованого і зваженого значення показника. UML представлення класів показників наведено на рис. 2.7.

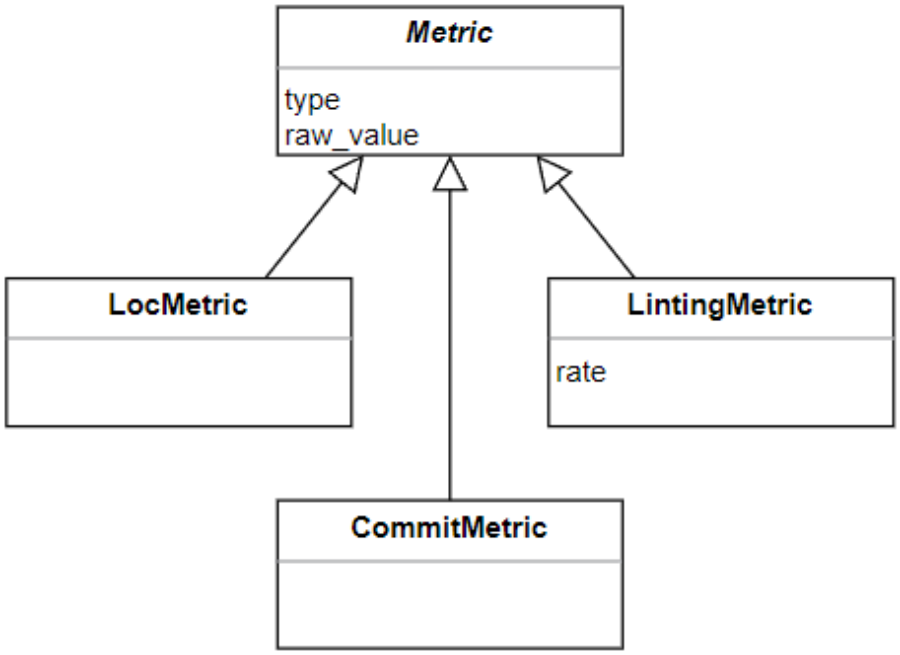


Рисунок 2.7 – UML-діаграма класів для представлення інформації про показники продуктивності розробника програмного забезпечення

На базі зібраних значень показників продуктивності для кожного розробника виконується розрахунок продуктивності. Для реалізації цього функціоналу розроблений клас ProductivityCalculator. Цей клас в якості вхідних даних отримує перелік показників продуктивності, групує показники по типам, розраховує середні значення, розраховує числове значення продуктивності. Метод калькуляції кінцевого значення продуктивності наведено в лістингу на рис. 2.8.

Для збереження результату роботи класу ProductivityCalculator реалізовано клас Productivity. Екземпляри цього класу зберігають числове значення

продуктивності і екземпляри класів показників продуктивності що використовувалися при розрахунках для одного розробника.

```
def calculate(self, metrics:List[Metric]) -> Tuple[float, List[Metric]]:
    grouped: Dict[type, Metric] = {}
    for metric in metrics:
        metric_type = type(metric)
        if not metric_type in grouped:
            grouped[metric_type] = metric
        else:
            grouped[metric_type] = metric.aggregate(grouped[metric_type])
    productivity:float = 0
    for metric in grouped.values():
        productivity += cast(Metric, metric).weighted_value
    productivity = productivity / len(grouped)
    return productivity, list(grouped.values())
```

Рисунок 2.8 – Лістинг методу розрахунку числового вираження продуктивності

Для представлення інформації про розрахунок продуктивності всіх розробників за вказаний період реалізовано клас Calculation. Екземпляри цього класу передаються в модуль збереження даних.

Загальна UML діаграма класів що відповідають за логіку роботи додатку наведено на рис. 2.9.

2.8 Організація збереження даних

Система що розробляється накопичує дані та надає доступ до даних за запитом. Необхідно зберігати дані про розробників, джерела показників, результати розрахунків. Це різноманітні сутності, вони структуровані, мають різні характеристики. З урахуванням характеристик даних що необхідно зберігати для збереження даних обираються реляційні бази даних [46].

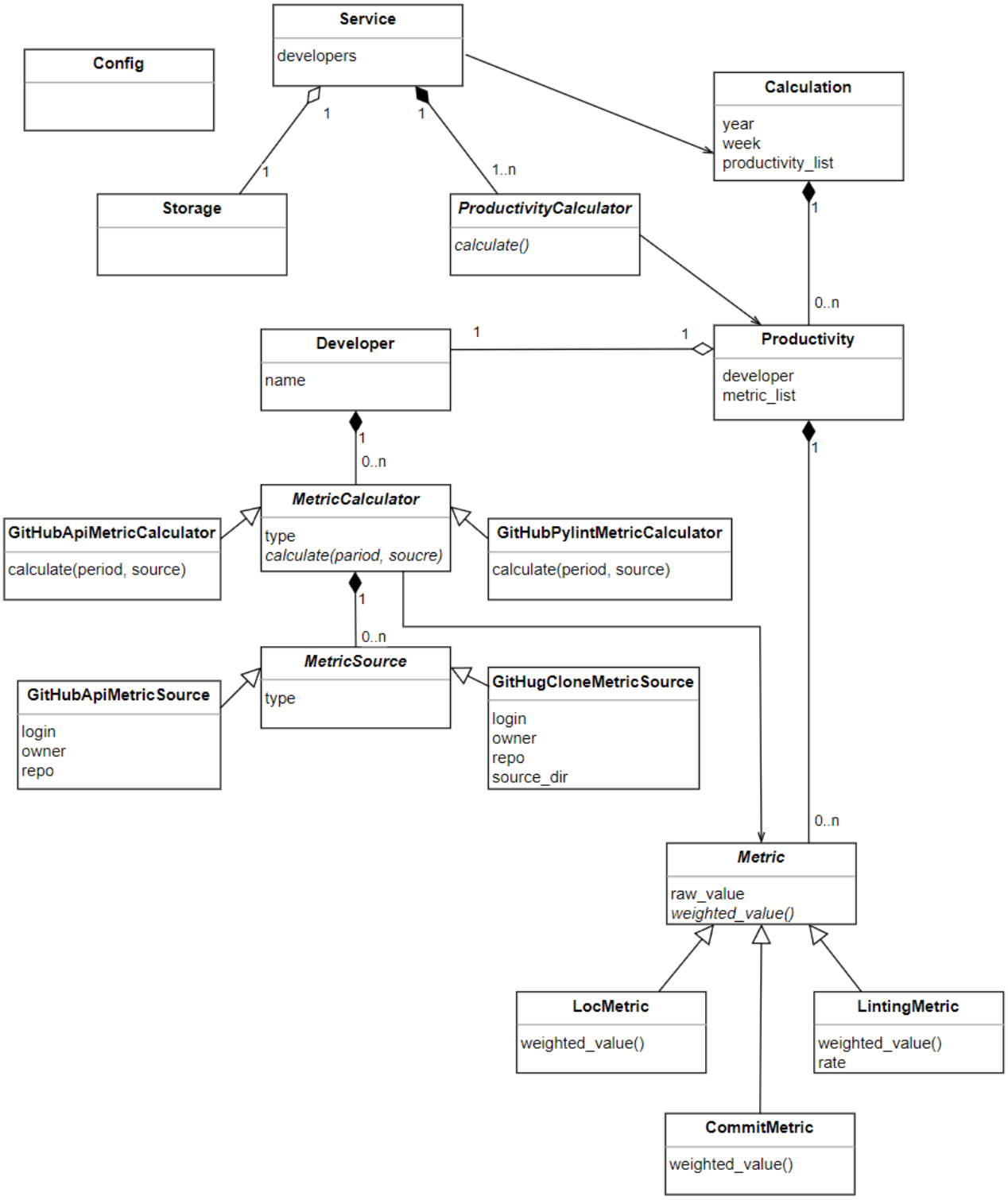


Рисунок 2.9 – UML-діаграма класів що реалізують логіку розрахунку продуктивності розробників програмного забезпечення

Одним з сучасних принципів розробки програмного забезпечення є принцип «спочатку код» (code first) [47]. Використання цього принципу дозволяє

досягти більш високої продуктивності розробки, можливість керування версіями додатків, можливість зміни параметрів. У випадку розробки функціональності збереження в базі даних підхід «спочатку код» дозволяє абстрагуватися від вибору конкретної системи управління реляційними базами даних (СУБД), використовувати різні СУБД при налагоджені і при промисловому використанні.

Для сучасних мов програмування розроблені бібліотеки які виконують функції зв'язування програмного коду і СУБД (Object-Ralation Mapping Library). Для мови програмування Python такою популярною бібліотекою є SQLAlchemy [48]. Бібліотека дозволяє в імперативному чи декларативному стилі описати структуру бази даних що розробляється, автоматично генерує SQL [49] код для створення бази даних [50]. Імперативний стиль дозволяє отримати більше контролю над технічними деталями. Декларативний стиль дозволяє використовувати описану модель даних в додатках доступу даних. Приклад коду опису сутностей бази даних в декларативному стилі та додавання записів з використанням цього коду наведено в лістингу на рис. 2.10.

На наведеному лістингу видно що для опису структури таблиці «Розробник» (Developer) і для додавання даних в цю таблицю створюється один спеціалізований ORM клас DbDeveloper.

Перелік сутностей що будуть зберігатися в базі даних з вказуванням імен класів та таблиць наведений в табл. 2.1.

Кожна таблиця має первинний ключ, дані в таблицях залежать тільки від первинного ключа і не залежать від інших даних в таблицях. Таким чином спроектована база даних відповідає третій нормальній формі [51]. Діаграма сутність-зв'язок для згенерованої бази даних наведена на рис. 2.11.

Як можна побачити з наведеної діаграми, таблиці metric_calculator «Метод розрахунку показників», metric_source «Джерело показників» мають поле json. З одного боку можна стверджувати що це порушення першої нормальної форми, бо в полі зберігається комплексне, а не атомарне значення. Але з іншого боку, це параметри однієї сутності, яка представлена сімейством класів і припускається що сімейство цих класів може суттєво розширюватися. В такому випадку,

рішення використання json формату [52] для зберігання серіалізованого представлення об’єктів методів розрахунку і джерел показників є виправданим. Також сучасні бази даних, такі як PostgreSQL [53] та SQLite [54] мають тип даних json і вбудовані функції для роботи з Json форматом, що дозволяє розглядати об’єкти json як атомарні дані.

```
# ...
class Base(DeclarativeBase):
    pass

class DbDeveloper(Base):
    __tablename__ = "developer"
    id:Mapped[int] = mapped_column(primary_key=True)
    name:Mapped[str] = mapped_column(unique=True, nullable=False)
    calculators:Mapped[List["DbMetricCalculator"]] =
    relationship(back_populates="developer")
    sources:Mapped[List["DbMetricSource"]] =
    relationship(back_populates="developer")

class DbMetricSourceType(Base):
    __tablename__ = "metric_source_type"
    id:Mapped[int] = mapped_column(primary_key=True)
    name:Mapped[str] = mapped_column(String(100), unique=True,
    nullable=False)

class DbMetricSource(Base):
    __tablename__ = "metric_source"
    id:Mapped[int] = mapped_column(primary_key=True)
    source_type_id:Mapped[int] =
    mapped_column(ForeignKey("metric_source_type.id",
    ondelete="RESTRICT"), nullable=False)
    developer_id:Mapped[int] = mapped_column(ForeignKey("developer.id",
    ondelete="CASCADE"), nullable=False)
    json:Mapped[str] = mapped_column(nullable=False)
    developer:Mapped["DbDeveloper"] =
    relationship(back_populates="sources")
    source_type:Mapped["DbMetricSourceType"] = relationship()
# ...
create_engine(engine_str)
Base.metadata.create_all(Storage.__engine)
# ...
storage = Storage()
session = storage.get_session()
developer1 = DbDeveloper(name="Developer Name")
session.add(developer1)
session.commit()
```

Рисунок 2.10 – Лістинг коду опису сутностей «Розробник», «Тип джерела показників», «Джерело показників» в декларативному стилі, виклик методу створення структури бази даних, додавання даних

Таблиця 2.1 – Перелік сутностей бази даних

Назва класу	Назва таблиці БД	Призначення
DbDeveloper	developer	Інформація про розробників програмного забезпечення
DbMetricSourceType	metric_source_type	Типи джерел показників продуктивності
DbMetricSource	metric_source	Параметри джерел показників продуктивності
DbMetricCalculatorType	metric_calculator_type	Типи методів розрахунку показників продуктивності
DbMetricCalculator	metric_calculator	Параметри методів розрахунку показників продуктивності
DbMetricType	metric_type	Типи показників продуктивності
DbCalculation	calculation	Параметри сеансів розрахунку продуктивності
DBMetric	calculation_metric	Значення показників продуктивності
DbGroupedMetric	calculation_grouped_metric	Значення показників продуктивності згрупованих за типом
DbProductivity	calculation_productivity	Значення продуктивності

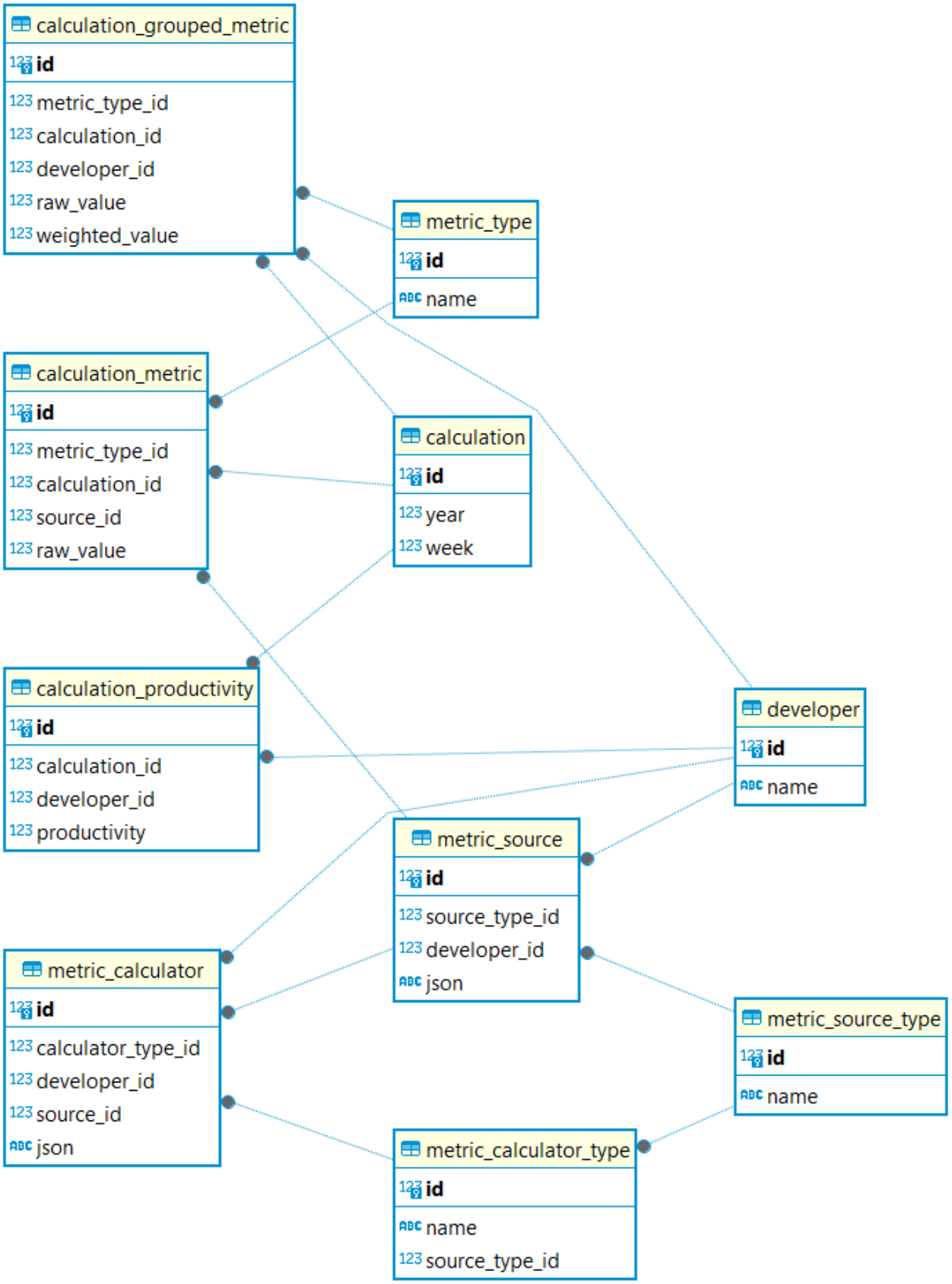


Рисунок 2.11 – Діаграма сутність-зв’язок для згенерованої бази даних

Обраний метод організації збереження даних в системі що розробляється дозволяє переходити на різні СУБД без зміни коду додатку. Достатньо тільки змінити адресу підключення до БД і в залежності від обраної СУБД встановити відповідну бібліотеку в середовищі виконання. Також це дає можливість використовувати в процесі розробки простішу СУБД, наприклад SQLite – поставляється з стандартною бібліотекою Python, а в процесі використання в виробництві – іншу, більш розвинену базу даних, наприклад, PostgreSQL.

2.9 Варіанти використання функціоналу системи, клієнтські додатки

Однією з вимог до системи що розробляється є можливість інтеграції в інфраструктуру та процес розробки програмного забезпечення. Реалізація системи що описана в попередніх двох розділах достатня для інтеграції у вигляди програмного модуля. Іншими шляхами автоматичного запуску функцій системи можуть бути додаток командного рядку та додаток що надає веб інтерфейс.

Стандартна бібліотека Python включає простий модуль argparse, який дозволяє з мінімальними зусиллями реалізувати додатки з інтерфейсом командного рядку. Приклад такого програмного додатку, який дозволяє запустити функцію розрахунку продуктивності для вказаного тижня наведений в додатку Г.

Для реалізації веб інтерфейсу необхідно застосувати додаткову бібліотеку. Невеликою, популярною бібліотекою для створення прототипів RESTful сервісів [55] є Flask [56]. Реалізація програмного додатку з використанням цієї бібліотеки наведена в додатку Г.

2.10 Висновки за розділом 2

В процесі виконання другого розділу роботи було сформульовано вимоги до системи оцінювання продуктивності розробників програмного забезпечення, запропоновано математичну модель розрахунку продуктивності і реалізовано необхідну функціональність на мові програмування Python.

В процесі розробки головного функціоналу системи було з’ясовано що існуючі інструменти спільної розробки, такі як GitHub, Git мають програмні інтерфейси які дозволяють отримувати інформації про різноманітні показники продуктивності. Необхідний для збору інформації про показники продуктивності Python код є компактним і порівняно простим в реалізації. При необхідності збору інформації про інші типи показників з інших джерел, можна швидко розширювати функціонал системи.

Python як мова програмування для реалізації систем збору і аналізу даних є продуктивним. Наявні всі необхідні бібліотеки для мережевої взаємодії, роботи з базами даних, аналізу даних, організації тестування функціоналу що розробляється. Це значно пришвидшує розробку.

Синтаксис Python дозволяє створювати компактний код що легко читати та аналізувати. Недоліки мови програмування Python, що можуть призводити до написання ненадійного коду, такі як відсутність контролю типу змінних, компенсуються наявністю таких інструментів контролю як PyLint, Муру, що інтегруються в сучасне середовище розробки.

При реалізації зберігання даних використовувався підхід «спочатку код» при якому структура бази даних для зберігання параметрів і результатів роботи системи описується безпосередньо в коді додатку.

Додатково якість і коректність коду перевірена розробленими модульними тестами.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

3.1 Формалізація реалістичних тестових даних

Для тестування системи оцінювання продуктивності розробників програмного забезпечення необхідно підготувати реалістичні тестові дані. Підготовка даних починається з пошуку джерел. Джерелом даних для системи що розробляється може бути налаштоване середовище командної розробки програмного забезпечення. Це середовище має реалізовувати інтерфейси для програмного отримання такої інформації як кількість доданих / видалених рядків коду, час і кількість збережень коду в систему за період часу, безпосередній доступ до різних версій коду, кількість вирішених завдань, тощо.

При відсутності можливості розгортання системи в середовищі закритих комерційних проєктів можна проаналізувати проєкти з відкритим вихідним кодом На теперішній момент існує декілька розвинених відкритих систем спільної розробки програмного забезпечення, які відповідають переліченим критеріям – GitHub [42], GitLab [57], BitBucket [58]. Найбільш популярним рішенням є GitHub. Треба зауважити що проєкти з відкритим вихідним кодом мають свою специфіку – розробники не працюють над ними на постійній основі, колектив розробників не є постійним. Це треба враховувати при виборі тестових проєктів.

GitHub пропонує розширене API, яке дозволяє автоматизувати збір інформації про репозиторії, збереження коду, огляд коду, запити на злиття коду, інциденти, і коментарі [59]. Це дозволяє організувати збір великих обсягів даних без необхідності ручного втручання. GitLab надає детальну інформацію за кожним збереженням коду, включаючи кількість рядків коду, що були додані або видалені.

Для тестування системи обираються проекти розробки програмного коду які відповідають наступним критеріям:

а) основна мова програмування Python – прототип системи має обмежений функціонал і реалізації статистичного аналізу коду тільки для мови програмування Python;

б) проект розробляється групою розробників – аналіз продуктивності розробників в рамках одного проекту / групи проектів дозволить оцінити результати роботи системи;

в) проекти мають мати періоди інтенсивної розробки – розробники не переключаються на роботи над іншими проектами.

GitHub має систему тегів і надає можливість пошуку проектів за мовою програмування та популярністю. В результаті пошуку за встановленими критеріями обрано групу проектів пов’язаних з продуктами Scrapy, Zyte. Scrapy – фреймворк з відкритим вихідним кодом для вилучення потрібних даних з веб-сайтів [60]. Zyte – комерційний сервіс, побудований на базі Scrapy [61]. Zyte також є продуктом з відкритим вихідним кодом. Перелік обраних проектів наведено в табл. 3.1.

Таблиця 3.1 – Проєкти обрані для аналізу

Назва	Посилання
Scrapy	https://github.com/scrapy/scrapy/
Scrapy Poet Extension	https://github.com/scrapinghub/scrapy-poet/
Scrapy Web Poet Extension	https://github.com/scrapinghub/web-poet/
Scrapy Zyte API	https://github.com/scrapy-plugins/scrapy-zyte-api/
Scrapy Spider Metadata	https://github.com/scrapy-plugins/scrapy-spider-metadata/
Scrapy Zyte SmartProxy	https://github.com/scrapy-plugins/scrapy-zyte-smartproxy/

Продовження таблиці 3.1

Назва	Посилання
Zyte Common Items	https://github.com/zytedata/zyte-common-items/
Zyte Spider Templates	https://github.com/zytedata/zyte-spider-templates/
Zyte Spider Templates Project	https://github.com/zytedata/zyte-spider-templates-project/
Python Zyte API	https://github.com/zytedata/python-zyte-api/
Zyte URL Matcher	https://github.com/zytedata/url-matcher/
Zyte Parsers	https://github.com/zytedata/zyte-parsers/

Для всіх наведених проєктів основною мовою розробки є Python. Таким чином розроблений функціонал збору і аналізу інформації для розрахунку продуктивності є достатнім.

Для аналізу обрано трьох розробників програмного забезпечення, які є постійними контриб'юторами обраних проєктів і для яких обрані проекти є основною діяльністю на платформі GitHub. Використання персональних даних не погоджувалось, тому надалі в роботі розробники згадуються як Розробник 1, Розробник 2 та Розробник 1.

Платформа GitHub також надає можливість візуально оцінити активність розробників за обраний період. Дані надаються у вигляді графічного поля. Графічне представлення активності Розробника 1 наведено на рис. 3.1, 3.2, 3.3.

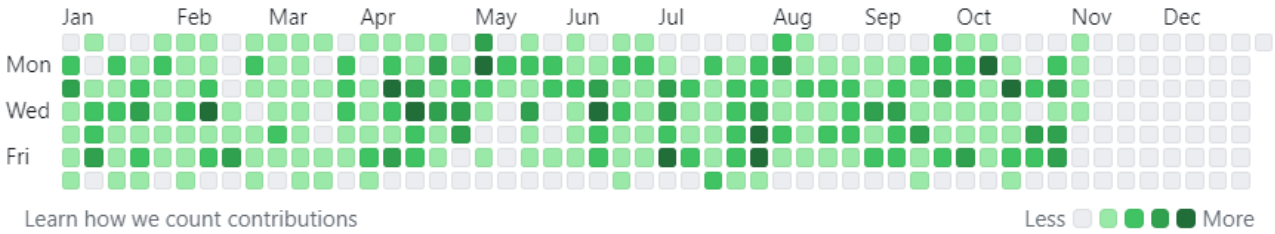


Рисунок 3.1 – Графічне представлення активності Розробника 1 за даними HitHub за 2023 рік

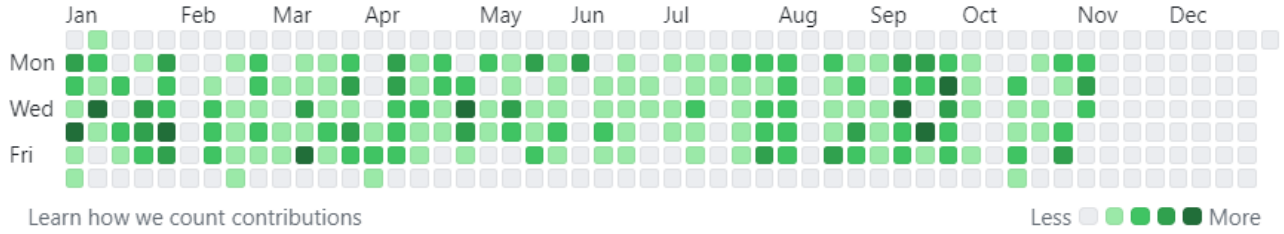


Рисунок 3.2 – Графічне представлення активності Розробника 2
за даними HitHub за 2023 рік

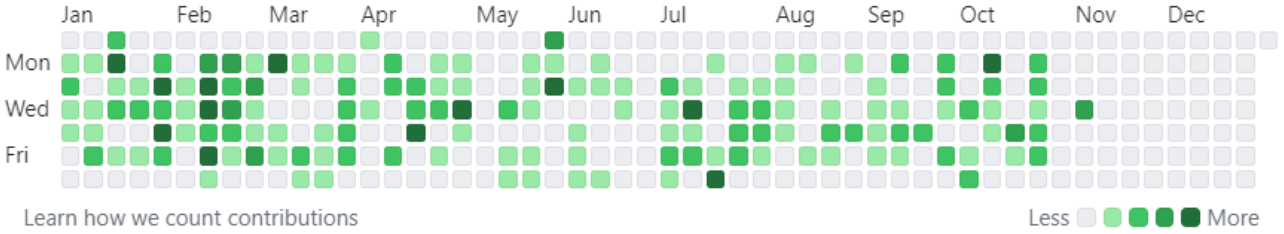


Рисунок 3.2 – Графічне представлення активності Розробника 3
за даними HitHub за 2023 рік

З наведених зображень видно що обрані розробники були активними на протязі 2023 року. Дані можуть бути використані для дослідження розробленої системи.

3.2 Аналіз результатів роботи системи

Логіка роботи системи оцінки продуктивності розробників програмного забезпечення складається з наступних етапів:

а) користувач системи конфігурує джерела збору інформації;

б) система збирає дані про показники і зберігає їх в базі даних;

в) система розраховує зважені, нормалізовані значення показників і значення числового вираження комплексної продуктивності;

г) система надає користувачу інформацію про розрахунки в табличному і

графічному вигляді;

д) користувач аналізує дані про продуктивність роботи розробників в порядку відповідному отриманим значенням продуктивності.

На першому кроці в базу даних системи вносяться параметри проектів перелічених в табл. 3.1. На другому кроці система збирає інформацію про показники продуктивності обраних розробників за вказаний період. Аналізуються дані за період з 1 тижня 01.01.2023 по 47 тиждень 26.11.2023.

Дані для розробників про показники, отримані системою в результаті виконання другого кроку, наведені в табл. 3.2, 3.3, 3.4.

Таблиця 3.2 – Інформація про показники продуктивності Розробника 1, зібрана системою за період з 01.01.2023 по 26.11.2023.

№ тижня	Рядки коду	Збере-ження	Заува-ження	№ тижня	Рядки коду	Збере-ження	Заува-ження
1	1057	34	435	25	13767	20	437
2	5838	19	435	26	1079	12	445
3	2084	11	435	27	20165	31	438
4	273	22	436	28	458	13	444
5	3384	4	435	29	230	13	444
6	3552	12	438	30	904	22	444
7	857	32	435	31	2203	43	444
8	235	12	437	32	898	17	444
9	101	7	436	33	668	12	444
10	270	9	437	34	100	11	444
11	296	9	437	35	911	13	444
12	0	0	437	36	28673	19	458
13	822	15	437	37	1195	24	458
14	294	10	437	38	290	13	458
15	1642	22	437	39	454	11	458
16	693	19	437	40	144	11	458
17	702	24	437	41	211	10	458
18	966	38	437	42	2547	13	458
19	310	15	437	43	336	20	458
20	11253	7	437	44	1479	28	455
21	22004	20	437	45	78	2	453
22	250	12	437	46	25	3	453
23	205	12	437	47	0	1	453
24	28967	38	437				

Таблиця 3.3 – Інформація про показники продуктивності Розробника 2, зібрана системою за період з 01.01.2023 по 26.11.2023.

№ тижня	Рядки коду	Збере-ження	Заува-ження	№ тижня	Рядки коду	Збере-ження	Заува-ження
1	430	39	435	25	3	1	437
2	210	9	435	26	0	0	437
3	1221	8	441	27	0	0	437
4	30548	19	436	28	59	4	437
5	15803	25	435	29	0	0	437
6	0	0	435	30	43	1	437
7	0	0	435	31	80	6	444
8	661	3	435	32	59	8	444
9	906	13	435	33	0	0	444
10	1	1	435	34	91	9	444
11	1042	13	437	35	54	5	444
12	35	4	437	36	28	3	444
13	1382	15	437	37	1848	29	444
14	0	0	437	38	591	35	458
15	176	15	437	39	683	18	458
16	0	0	437	40	3	1	458
17	519	4	437	41	0	0	458
18	520	23	437	42	607	6	455
19	86	2	437	43	0	0	455
20	889	15	437	44	655	4	455
21	987	9	437	45	1063	10	455
22	689	8	437	46	244	4	455
23	0	0	437	47	1678	10	455
24	138	5	437				

Таблиця 3.4 – Інформація про показники продуктивності Розробника 3, зібрана системою за період з 01.01.2023 по 26.11.2023.

№ тижня	Рядки коду	Збере-ження	Заува-ження	№ тижня	Рядки коду	Збере-ження	Заува-ження
1	16	5	0	25	6	1	435
2	145	4	435	26	0	0	435
3	1515	8	435	27	1338	4	435
4	709	2	435	28	304	17	435
5	2827	17	435	29	318	6	438
6	357	5	435	30	262	4	438
7	2009	29	435	31	327	4	438
8	365	17	435	32	27	3	438

Продовження таблиці 3.4

№ тижня	Рядки коду	Збере-ження	Заува-ження	№ тижня	Рядки коду	Збере-ження	Заува-ження
9	95	14	435	33	0	0	438
10	102	1	435	34	528	5	438
11	120	3	435	35	88	3	458
12	1387	2	435	36	461	3	458
13	308	4	435	37	251	3	458
14	7	1	435	38	30	2	458
15	380	6	435	39	1067	7	458
16	327	6	435	40	40	3	458
17	26	1	435	41	182	5	458
18	289	5	435	42	59	5	458
19	0	0	435	43	161	7	458
20	468	5	435	44	0	0	458
21	1023	9	435	45	40	5	458
22	49	6	435	46	111	1	458
23	270	2	435	47	131	2	458
24	37	1	435				

Для порівняння показників продуктивності розробників дані також представлені в графічному вигляді на рисунках 3.4, 3.5, 3.6.

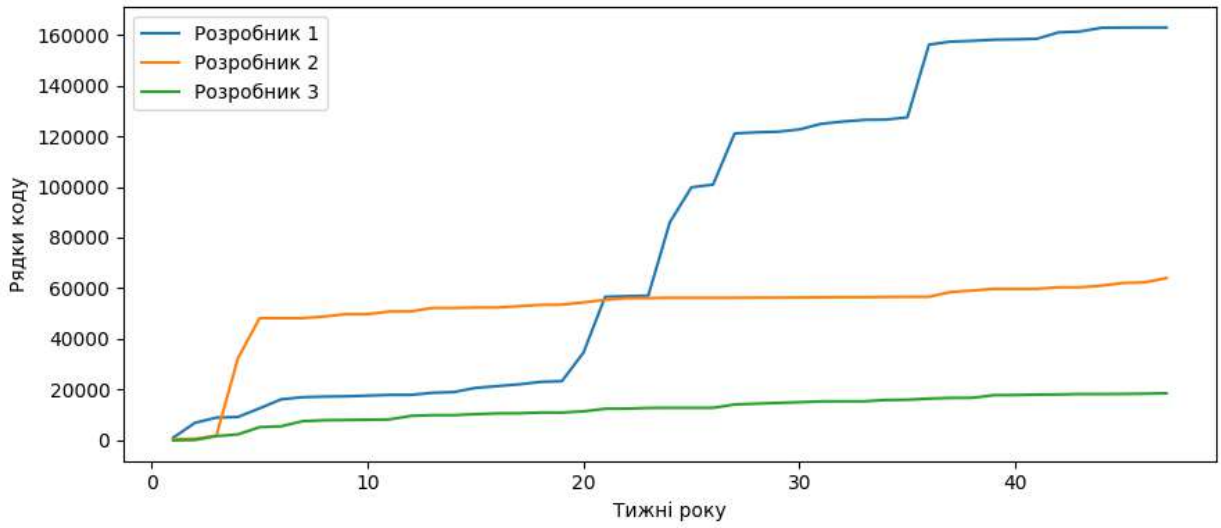


Рисунок 3.4 – Кумулятивна кількість рядків коду що додавалися розробниками за період з 01.01.2023 по 26.11.2023

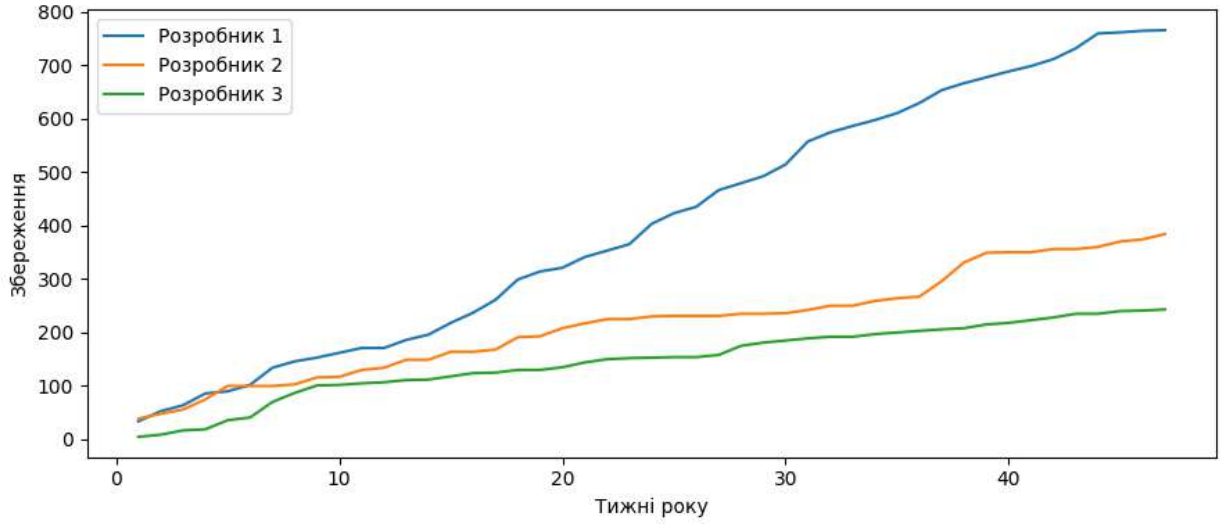


Рисунок 3.5 – Кумулятивна кількість збережень коду що здійснювалися розробниками за період з 01.01.2023 по 26.11.2023

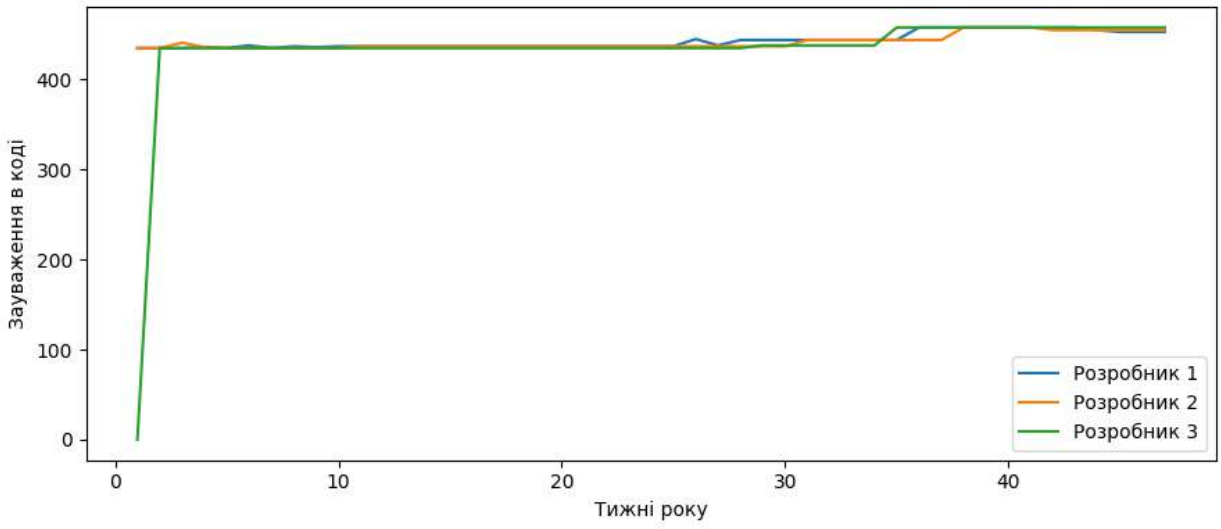


Рисунок 3.6 – Кількість зауважень в коді за період з 01.01.2023 по 26.11.2023

На етапі аналізу необроблених показників продуктивності можна побачити що розробники активно працювали на протязі всього періоду що розглядається. Виділяється кількість доданих рядків коду і збережень що робив Розробник 1. Якщо відштовхуватись від аналізу кількісних показників, то можна сказати що Розробник 1 є найбільш продуктивним, а Розробник 3 – найменш продуктивним. Негативним фактором є плавне зростання кількості зауважень в коді. Кількість зауважень розраховується для проєкту в цілому і відповідно майже співпадає у

всіх розробників. При розрахунку продуктивності розробників враховується не абсолютне значення цього показника, а його зміна. Абсолютне значення не дозволяє оцінити вплив показника на продуктивність.

На третьому кроці система, відповідно до завданих правил, розраховує зважені оцінки параметрів та значення комплексної продуктивності розробників. Дані про ці розрахунки для Розробник 1, Розробник 2, Розробник 3 наведені в табл. 3.5, 3.6, 3.7 відповідно.

Таблиця 3.5 – Зважені показники продуктивності і розраховане значення комплексної продуктивності Розробника 1

№ т-ня	Ряд. коду	Збер.	Заув.	Прод.	№ т-ня	Ряд. коду	Збер.	Заув.	Прод.
1	11.43	100	100	70.48	25	4.29	50	50	34.76
2	100	100	50	83.33	26	0.00	0.00	50	16.67
3	100	100	50	83.33	27	100	100	50	83.33
4	100	100	50	83.33	28	100	100	50	83.33
5	100	100	50	83.33	29	100	100	50	83.33
6	100	100	50	83.33	30	100	100	50	83.33
7	100	100	50	83.33	31	100	100	50	83.33
8	100	100	50	83.33	32	19.29	100	50	56.43
9	67.86	100	50	72.62	33	0.00	0.00	50	16.67
10	72.86	50	50	57.62	34	100	100	50	83.33
11	85.71	100	50	78.57	35	62.86	100	25.00	62.62
12	100	100	50	83.33	36	100	100	50	83.33
13	100	100	50	83.33	37	100	100	50	83.33
14	5.00	50	50	35.00	38	21.43	100	50	57.14
15	100	100	50	83.33	39	100	100	50	83.33
16	100	100	50	83.33	40	28.57	100	50	59.52
17	18.57	50	50	39.52	41	100	100	50	83.33
18	100	100	50	83.33	42	42.14	100	50	64.05
19	0.00	0.00	50	16.67	43	100	100	50	83.33
20	100	100	50	83.33	44	0.00	0.00	50	16.67
21	100	100	50	83.33	45	28.57	100	50	59.52
22	35.00	100	50	61.67	46	79.29	50	50	59.76
23	100	100	50	83.33	47	93.57	100	50	81.19
24	26.43	50	50	42.14					

Таблиця 3.6 – Зважені показники продуктивності і розраховане значення комплексної продуктивності Розробника 2

№ т-ня	Ряд. коду	Збер.	Заув.	Прод.	№ т-ня	Ряд. коду	Збер.	Заув.	Прод.
1	100	100	50	83.33	25	2.14	50	50	34.05
2	100	100	50	83.33	26	0	0	50	16.67
3	100	100	25.00	75.00	27	0	0	50	16.67
4	100	100	75.00	91.67	28	42.14	100	50	64.05
5	100	100	50	83.33	29	0	0	50	16.67
6	0	0	50	16.67	30	30.71	50	50	43.57
7	0	0	50	16.67	31	57.14	100	25.00	60.71
8	100	100	50	83.33	32	42.14	100	50	64.05
9	100	100	50	83.33	33	0	0	50	16.67
10	0.71	50	50	33.57	34	65.00	100	50	71.67
11	100	100	50	83.33	35	38.57	100	50	62.86
12	25.00	100	50	58.33	36	20.00	100	50	56.67
13	100	100	50	83.33	37	100	100	50	83.33
14	0	0	50	16.67	38	100	100	25.00	75.00
15	100	100	50	83.33	39	100	100	50	83.33
16	0	0	50	16.67	40	2.14	50	50	34.05
17	100	100	50	83.33	41	0	0	50	16.67
18	100	100	50	83.33	42	100	100	50	83.33
19	61.43	100	50	70.48	43	0	0	50	16.67
20	100	100	50	83.33	44	100	100	50	83.33
21	100	100	50	83.33	45	100	100	50	83.33
22	100	100	50	83.33	46	100	100	50	83.33
23	0	0	50	16.67	47	100	100	50	83.33
24	98.57	100	50	82.86					

Таблиця 3.7 – Зважені показники продуктивності і розраховане значення комплексної продуктивності Розробника 3

№ т-ня	Ряд. коду	Збер.	Заув.	Прод.	№ т-ня	Ряд. коду	Збер.	Заув.	Прод.
1	11.43	100	100	70.48	25	4.29	50	50	34.76
2	100	100	50	83.33	26	0	0	50	16.67
3	100	100	50	83.33	27	100	100	50	83.33
4	100	100	50	83.33	28	100	100	50	83.33
5	100	100	50	83.33	29	100	100	50	83.33
6	100	100	50	83.33	30	100	100	50	83.33
7	100	100	50	83.33	31	100	100	50	83.33

Продовження таблиці 3.7

№ т-ня	Ряд. коду	Збер.	Заув.	Прод.	№ т-ня	Ряд. коду	Збер.	Заув.	Прод.
8	100	100	50	83.33	32	19.29	100	50	56.43
9	67.86	100	50	72.62	33	0	0	50	16.67
10	72.86	50	50	57.62	34	100	100	50	83.33
11	85.71	100	50	78.57	35	62.86	100	25.00	62.62
12	100	100	50	83.33	36	100	100	50	83.33
13	100	100	50	83.33	37	100	100	50	83.33
14	5.00	50	50	35.00	38	21.43	100	50	57.14
15	100	100	50	83.33	39	100	100	50	83.33
16	100	100	50	83.33	40	28.57	100	50	59.52
17	18.57	50	50	39.52	41	100	100	50	83.33
18	100	100	50	83.33	42	42.14	100	50	64.05
19	0	0	50	16.67	43	100	100	50	83.33
20	100	100	50	83.33	44	0	0	50	16.67
21	100	100	50	83.33	45	28.57	100	50	59.52
22	35.00	100	50	61.67	46	79.29	50	50	59.76
23	100	100	50	83.33	47	93.57	100	50	81.19
24	26.43	50	50	42.14					

Для наочності результати розрахунку зважених показників і продуктивності наведено на рис. 3.7, 3.8, 3.9, 3.10.

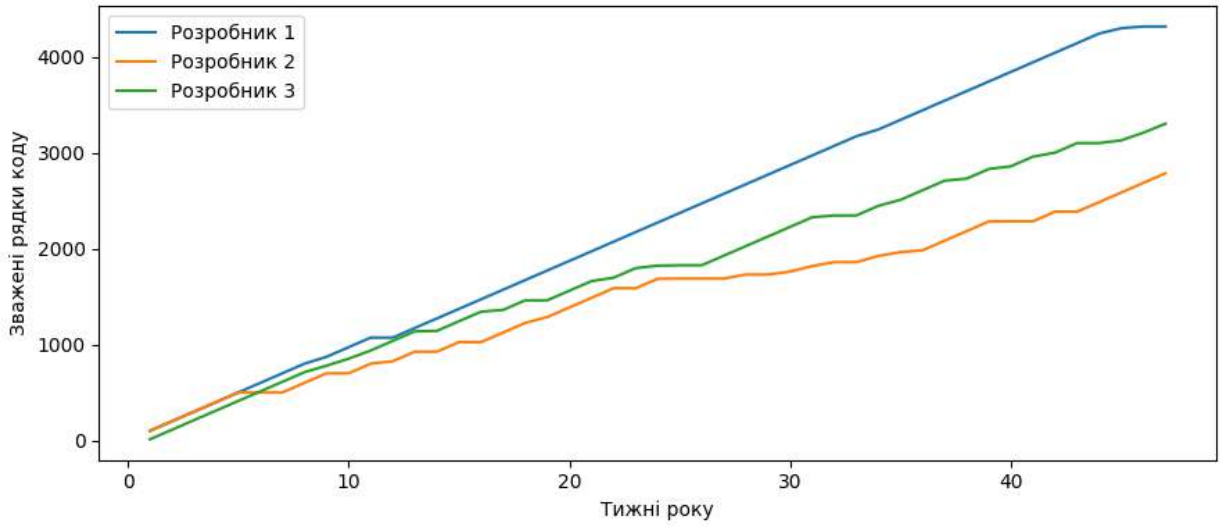


Рисунок 3.7 – Кумулятивна зважена кількість рядків коду що додавалися розробниками за період з 01.01.2023 по 26.11.2023

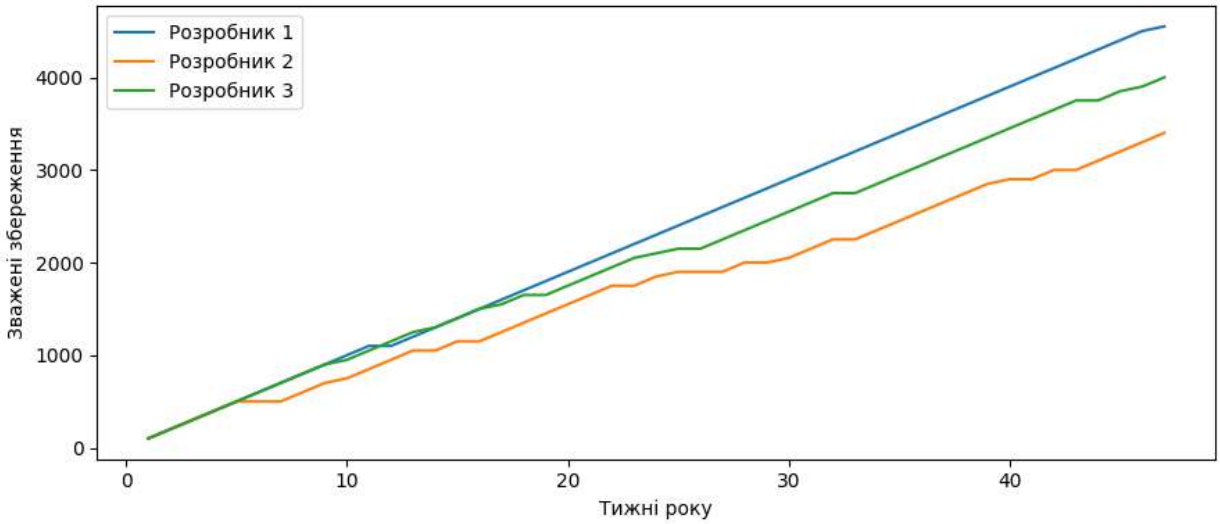


Рисунок 3.8 – Кумулятивна кількість збережень коду що здійснювалися розробниками за період з 01.01.2023 по 26.11.2023

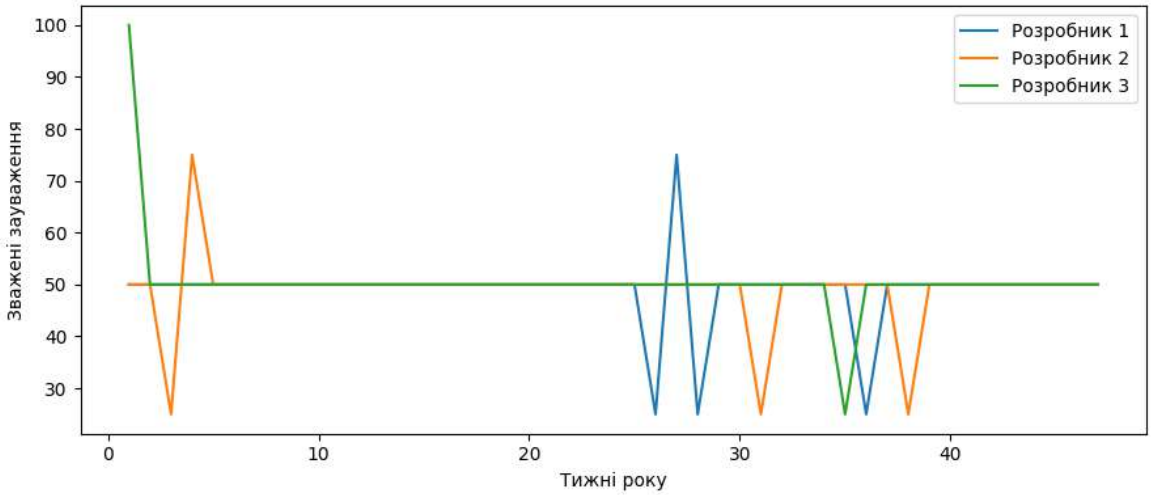


Рисунок 3.9 – Зважена кількість зауважень в коді за період з 01.01.2023 по 26.11.2023

На рис. 3.9 наглядно видно логіку роботи системи з показником якості коду. При наявності зауважень до коду, при відсутності змін в кількості зауважень зважене значення показника дорівнює 50 одиницям, при появі нових помилок, в залежності від кількості значення показника знижується. При наявності зауважень до коду і виправлені деяких з них, значення показника збільшується в залежності від кількості виправлень. Максимальне значення 100, може бути

досягнуто тільки при відсутності помилок в коді. Зміни в коді зроблені розробниками Розробник 1 і Розробник 2 частіше призводили до появи нових помилок ніж зміни в коді зроблені Розробником 3.

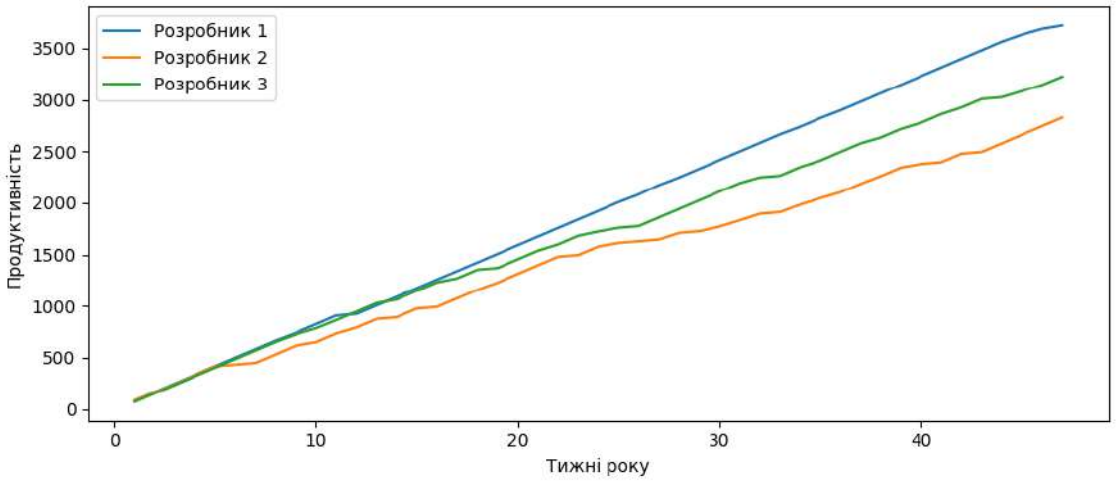


Рисунок 3.10 – Кумулятивна комплексна продуктивність розробників за період з 01.01.2023 по 26.11.2023

Наявні дані також дозволяють оцінити середню загальну продуктивність команди розробки. Ця оцінка наочно продемонструє які показники найбільш негативно впливають на продуктивність. Графіки загальної продуктивності і показників наведено на рис. 3.11, 3.12 в звичайному і кумулятивному вигляді відповідно.

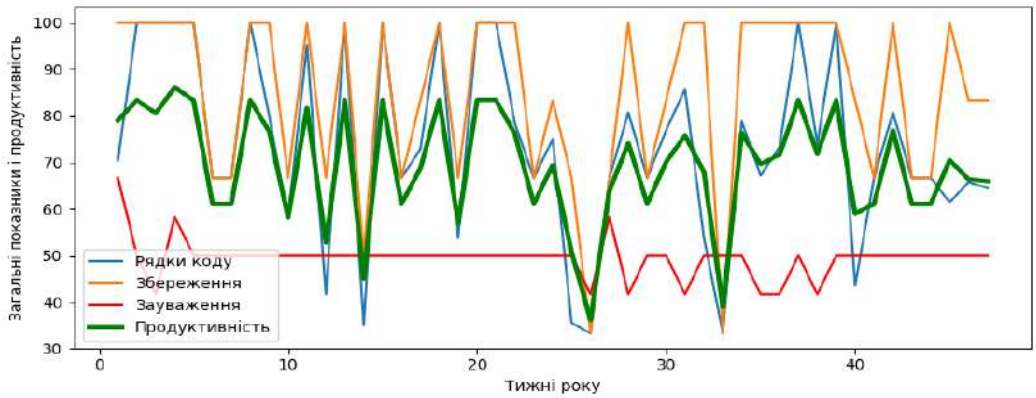


Рисунок 3.11 – Загальні показники і продуктивність роботи команди розробників

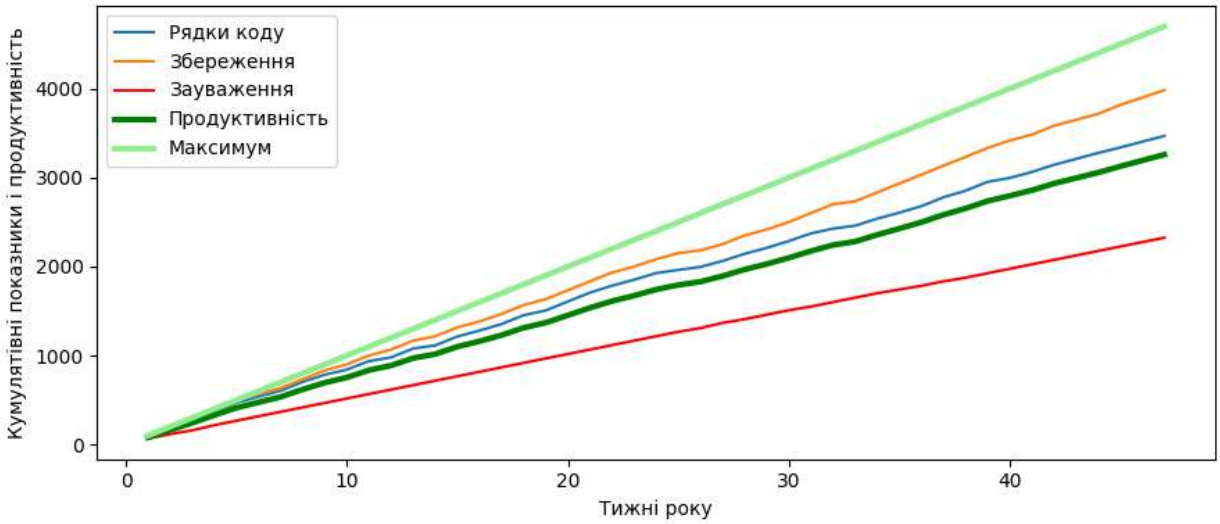


Рисунок 3.12 – Загальні показники і продуктивність роботи команди розробників

Графік 3.11 дозволяє побачити, що найбільш негативно на загальну оцінку продуктивності впливає наявність помилок в коду і що найбільше зниження продуктивності відбувалося з 26-го по 34-й тижні року. Графік 3.12 дозволяє графічно побачити різницю між фактичною і максимально можливою продуктивністю. Фактична продуктивність на кінець періоду що аналізується складала 69.4% від максимально можливої продуктивності.

3.3 Висновки за розділом 3

В ході дослідження розробленої системи оцінювання продуктивності розробників програмного забезпечення були обрані проєкти з відкритим вихідним кодом і розробники що працюють над цими проєктами. Система виконала збір інформації про показники роботи розробників та виконала розрахунок продуктивності.

Отримані результати продемонстрували що пряма трактовка чисельних показників продуктивності може призводити до висновків, які можуть

відрізнятися від таких що отримуються при використання комплексної оцінки. За кількісними показниками Розробник 2 має кращі результати ніж Розробник 3. Але стабільність і якість роботи Розробника 3 дозволили йому отримати більш високе значення комплексної продуктивності. Ця відмінність демонструє що система дозволяє зменшити значення людського фактору, суб’єктивності при оцінці продуктивності, врахувати більше факторів і коректніше визначити пріоритети при плануванні роботи з розробниками над покращенням процесів і продуктивності.

Наявність можливості аналізу не тільки фінального значення продуктивності, але й результатів проміжних розрахунків дозволяє після визначення пріоритетів зрозуміти над якими саме недоліками треба працювати для покращення процесів і продуктивності.

ВИСНОВКИ

Задача оцінки продуктивності розробників програмного забезпечення є актуальною, що підтверджується наявністю нових програмних продуктів що розвиваються і нещодавніх наукових робіт. Більше того, актуальність буде зростати через збільшення об’єму цифровізації та кількості розробників що працюють віддалено.

Аналіз продуктивності розробників програмного забезпечення вимагає такої побудови процесу розробки, при якій використовуються інструменти, які дозволяють сумісну роботу і накопичення даних про активність і результати роботи. На поточний момент є всі необхідні інструменти для побудови таких середовищ. Фіксація інформації про виконану роботу, якісна комунікація мають бути частиною культури розробки і заохочуватись.

Сучасні методи розробки, мови програмування, середовища розробки дозволяють ефективно розробляти системи подібні до системи оцінювання продуктивності розробників програмного забезпечення що створювалася в рамках цієї роботи. Використана мова програмування Python надзвичайно ефективна для створення пілотних і дослідницьких систем. Сучасні методи розробки і бібліотеки дозволяють компенсувати такі недоліки Python як відсутність строгої типізації і уникнути потенційних помилок. Використане середовище розробки Microsoft Visual Studio Code є ефективним модульним інструментом.

В якості джерела даних для аналізу роботи системи були використані проєкти з відкритим вихідним кодом що розташовані на платформі GitHub. При підборі проєктів і розробників виникли труднощі пов’язані з тим що розробники не працюють над проектами з відкритим вихідним кодом систематично, як наймані працівники працюють над комерційними проектами. Пошук проєктів, що відповідають необхідним критеріям – систематичність роботи, стабільний колектив розробників, зайняв суттєвий час.

Побудована в рамках даної роботи система оцінювання продуктивності розробників програмного забезпечення використовує базові кількісні, комунікативні і якісні показники. Система спроєктована таким чином щоб надавати можливість розширювати функціонал, додавати підтримку показників продуктивності, мов програмування, середовищ розробки і взаємодії.

Аналіз результатів, отриманих в процесі дослідження розробленої системи, показав що врахування окремих показників роботи не дає можливості вірно оцінити продуктивність. Розрахунок комплексного показника продуктивності дозволяє зменшити вплив суб'єктивності. Числове вираження комплексного показника продуктивності, нормалізація значень показників продуктивності дають можливість виконати аналіз і наочно продемонструвати результати роботи розробника програмного забезпечення і можливі шляхи покращення продуктивності. Фіксований діапазон можливих значень продуктивності дозволяє однозначно трактувати результат і оцінювати наскільки можна покращити продуктивність.

Слід зауважити, що при ранжуванні і нормалізації значень показників продуктивності використовуються коефіцієнти визначені користувачем/адміністратором системи оцінювання продуктивності. Це впливає на кінцевий результат. Таким чином, методи нормалізації і ранжування кожного показника, вага кожного показника мають визначатися в залежності від пріоритетів та особливостей проєктів розробки програмного забезпечення.

В межах подальшого вивчення застосування системи оцінювання продуктивності розробників програмного забезпечення може бути виконаний тривалий експеримент, який має включати створення контрольованої інфраструктури розробки програмного забезпечення, налаштування системи для аналізу даних проекту, ітеративний аналіз результатів роботи системи та впровадження змін в налаштуванні системи та в процеси проєкту. Також необхідне дослідження більшої кількості типів показників продуктивності.

ЛІТЕРАТУРА

1. Programming Productivity [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Programming_productivity

2. Tangen S. «Demystifying productivity and performance» / S. Tangen // International Journal of Productivity and Performance Management – 2005, 54. – С. 34–46.

3. Ramírez Y. W. Measuring knowledge worker productivity: A taxonomy / Y. W. Ramírez, D. A. Nembhard // Journal of Intellectual Capital. – 2004, 5. – С. 602-628.

4. Chew B. W. No-Nonsense Guide to Measuring Productivity / B. W. Chew // Harvard Business Review. 1988, 66. – С. 110-115.

5. Drucker P. F. Knowledge-Worker Productivity: The Biggest Challenge / P. F. Drucker // California Management Review. 1999, 41. – С. 79-94.

6. Jason F. Remote: Office Not Required / F. Jason, D. Heinemeier Hansson – New York: Crown Business, 2013. – 250 с.

7. Kutner J. Remote Pairing: Collaborative Tools for Distributed Development / J. Kutner – The Pragmatic Bookshelf, Dallas, 2014. – 90 с.

8. Hall T. What Do We Know about Developer Motivation? / T Hall, H. Sharp, S. Beecham, N. Baddoo, H. Robinson. // IEEE Software. 2008, 25, 4. – С. 92–94.

9. Brooks F. P. The Mythical Man-Month Essays on Software Engineering / F. P. Brooks – Addison-Wesley, 2013. – 336 с.

10. DeMarco T. Controlling Software Projects: Management Measurement & Estimation / T. DeMarco, Boehm B.W. – Yourdon Press, 1982. – 284 с.

11. Kurtel K. Quantifying Productivity of Individual Software Programmers: Practical Approach / K. Kurtel // Computing and Informatics. 2015, 34. – С. 959–972.

12. DeMarco T. Peopleware: Productive Projects and Teams / T. DeMarco, T. R. Lister – Addison-Wesley Professional, 2013. – 272с.

13. Fenton, N. E. Software Metrics: Successes, Failures and New Directions / N. E. Fenton, M. Neil // Journal of Systems and Software. 1999, 47. – С. 149–157.

14. Nicolette D. Software Development Metrics / D. Nicolette – Manning, 2015. – 192 с.

15. Forsgren N. The SPACE of Developer Productivity: There's more to it than you think / Forsgren N. Storey M.-A., Maddila C., Zimmermann T., Houck B, Butler J. // Queue. 2021, 19, 1. – С. 20–48.

16. McConnell S. Software Estimation : Demystifying the Black Art / S. McConnel. – New York: O'Reilly Media, 2009. – 308 с.

17. Chase Thiel, Julena M. Bonner, John Bush, David Welsh, Niharika Garud. 2022. Monitoring Employees Makes Them More Likely to Break Rules. Harvard Business Review.

18. Pluarlsight Flow [Електронний ресурс]. – Режим доступу: https://www.pluralsight.com/product/flow

19. AnalyticsVerse [Електронний ресурс]. – Режим доступу: https://analyticsverse.com/

20. Wiegers K. E. Software Requirements / K.E. Wiegers, J. Beatty – Redmond, Washington: Microsoft, 2013. – 637 с.

21. Leffingwell D. Managing Software Requirements : A Use Case Approach / D. Leffingwell, D. Widrig – Boston: Addison-Wesley, 2003. – 502с.

22. Duvall P. M. Continuous Integration Improving Software Quality and Reducing Risk / P. M. Duvall, S. Matyas, A. Glover – Upper Saddle River NJ: Addison-Wesley, 2013. – 283 с.

23. Poulton N. Docker Deep Dive: Zero to Docker in a Single Book! / N. Poulton – Birmingham UK: Packt Publishing, 2023. – 266 с.

24. Kaushal B. Analysis Of Source Lines Of Code(SLOC) Metric / B. Kaushal, T. Vinit, P. Pushpraj // International Journal of Emerging Technology and Advanced Engineering. – 2012, 2, 5.

25. Pressman R. S. Software Engineering: A Practitioner's Approach / R.S. Pressman – McGraw-Hill Higher Education, 2010. – 895 с.

26. McConnell, S. Code Complete: A Practical Handbook of Software Construction / S. McConnel – Microsoft Press, 2004. – 960 с.

27. Carsten K. The Empirical Commit Frequency Distribution of Open Source Projects / K. Carsten, D. Riehle, M. A. Salim // Proceedings of the 9th International Symposium on Open Collaboration, 2013 – 8 с.

28. Møller A. Static Program Analysis / A. Møller, M. I. Schwartzbach – Department of Computer Science, Aarhus University, Denmark, 2023. – 210 с.

29. Lano K. Introduction to Software Architecture: Innovative Design Using Clean Architecture and Model-Driven Engineering / K. Lano, Y. T. Sobhan – Springer, 2023. – 264 с.

30. Fowler M. Uml Distilled: A Brief Guide to the Standard Object Modeling Language / M. Fowler – Boston: Addison-Wesley, 2015. – 175 с.

31. Farshidi S. A decision model for programming language ecosystem selection: Seven industry case studies / S. Farshidi, S. Jansen, M. Deldar // Information and Software Technology. – 2021.

32. Stroustrup B. Programming Principles and Practice Using C / B. Stroustrup – Upper Saddle River NJ: Addison-Wesley, 2018. – 1274 с.

33. Richter J. CLR Via C# / J. Richter – Redmond WA: Microsoft, 2013. – 863 с.

34. Matthes E. Python Crash Course: A Hands-On Project-Based Introduction to Programming / E. Matthes – San Francisco: No Starch Press, 2023. – 514 с.

35. Alomari Z. Comparative Studies of Six Programming Languages / Z. Alomari, O. E. Halimi, K. Sivaprasad, C. Pandit – ArXiv, 2015. – 71 с.

36. TIOBE Index for November 2023 [Електронний ресурс] – Режим доступу: https://www.tiobe.com/tiobe-index/

37. Most Used Python IDEs and Code Editors [Електронний ресурс] – Режим доступу: https://www.codium.ai/blog/most-used-python-ides-and-code-editors/

38. Speight A. Visual Studio Code for Python Programmers / A. Speight – Hoboken NJ: Wiley, 2021. – 234 с.

39. Booch G. Object-Oriented Analysis and Design with Applications / G. Booch – Addison-Wesley Professional, 2011. – 720 с.

40. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software / Gamma E., R. Helm, R. Johnson, J. Vlissides – Addison-Wesley Professional, 1994. – 416 с.

41. Van Deursen S. Mark Seemann Dependency Injection Principles Practices and Patterns / S.Van Deursen, M. Seemann – Manning Publications, 2019. – 552 с.

42. GitHub [Електронний ресурс]. – Режим доступу: https://github.com/

43. Anaya M. Clean Code in Python Develop Maintainable and Efficient Code / M. Anaya – Birmingham: Packt Publishing Limited, 2021. – 423 с.

44. GitHub REST API Documentation [Електронний ресурс]. – Режим доступу: https://docs.github.com/en/rest

45. Martin R. C. Clean Architecture / R. C. Martin – Harlow England: Prentice Hall, 2017. – 432 с.

46. Khan W. SQL and NoSQL Database Software Architecture Performance Analysis and Assessments – A Systematic Literature Review / W. Khan, K. Teerath та ін. – arXiv, 2022. – 57 с.

47. Lerman J. Programming Entity Framework: Code First / J. Lerman, R. Miller – Beijing Sebastopol CA: O'Reilly Media: Lightning Source UK, 2013. – 176 с.

48. Myers J. Essential Sqlalchemy / J. Myers, R. Copeland – Sebastopol CA: O'Reilly Media, 2016. – 230 с.

49. Beaulieu A. Learning SQL / A. Beaulieu – O'Reilly Media, 2020. – 380 с.

50. Ramm M. SQLAlchemy: Database Access Using Python / M. Ramm, M. Bayer – Boston Mass. London: Addison-Wesley ; Pearson Education distributor, 2009. – 504 с.

51. Captain F. A. Six-Step Relational Database Design: A Step by Step Approach to Relational Database Design and Development / F. A. Captain – CreateSpace Independent Publishing Platform, 2013. – 232 с.

52. Smith B. Beginning Json / B. Smith – New York NY: Apress, 2015. – 300 с.

53. Schönig H. J. Mastering PostgreSQL 15: Advanced Techniques to Build and Manage Scalable Reliable and Fault-Tolerant Database Applications / H. J. Schönig – Birmingham UK: Packt Publishing, 2023. – 522 с.

54. Grant A. The Definitive Guide to SQLite / A. Grant, M. Owens – New York: Apress, 2010. – 347 с.

55. Peralta J. H. Microservice APIs: Using Python Flask FastAPI OpenAPI and More / J. H. Peralta – Shelter Island NY: Manning Publications, 2023. – 412 с.

56. Copperwaite M. Learning Flask Framework: Build Dynamic Data-Driven Websites and Modern Web Applications with Flask / M. Copperwaite, C. Leifer – Birmingham UK: Packt Publishing, 2015. – 412 с.

57. GitLab DevSecOps Platform [Електронний ресурс]. – Режим доступу: https://about.gitlab.com/

58. BitBucket Code & CI/CD [Електронний ресурс]. – Режим доступу: https://bitbucket.org/product

59. Sharma R. U. Github Essentials: Unleash the Power of Collaborative Development Workflows Using GitHub / R. U. Sharma, A. Pipinellis Birmingham: Packt Publishing, 2018. – 170 с.

60. Scrapy [Електронний ресурс]. – Режим доступу: https://github.com/scrapy/scrapy

61. Zyte [Електронний ресурс]. – Режим доступу: https://www.zyte.com/

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
ПЗ		Нещільне заповнення сторінок, перед підзаголовком пункту або підпункту по 2 порожніх рядка
С. 29, 35, 46		Перехід до наступної сторінки не потрібен.

Додаток Б

Лістинг коду реалізації логіки роботи сервісу

Лістинг Б.1 – Файл calculation.py

```
from typing import List
from cdpservice.entity import Entity

from cdpservice.productivity import Productivity


class Calculation(Entity):
    def __init__(self, year:int, week:int,
productivity_list:List[Productivity]):
        super().__init__()
        self.__year = year
        self.__week = week
        self.__productivity_list = productivity_list

    @property
    def year(self):
        return self.__year

    @property
    def week(self):
        return self.__week

    @property
    def productivity_list(self):
        return self.__productivity_list
```

Лістинг Б.2 – Файл config.py

```
class CdpServiceConfig:
    """Global configuration values"""

    WORKING_DIR = "./workdir"

    GITHUB_TOKEN = "ghp_xJoLslZtkI1OXurtABpyW9QjOsY9vo7OPUgU"

    METRIC_WEIGHTED = 100

    DB_CONN = "sqlite:///db.sqlite"
```

Лістинг Б.3 – Файл developer.py

```
from typing import List
from cdpservice.entity import Entity
from cdpservice.metric_calculator import MetricCalculator

class Developer(Entity):

    def __init__(self, name:str, calculators: List[MetricCalculator]):
        super().__init__()
        self.__name = name
```

```
        self.__calculators = calculators
```

Продовження лістингу Б.3

```
    @property
    def name(self):
        return self.__name

    @property
    def calculators(self):
        return self.__calculators
```

Лістинг Б.4 – Файл entity.py

```
from abc import ABC

class Entity(ABC):

    def __init__(self):
        self.__dbid = None

    @property
    def dbid(self) -> int:
        if not self.__dbid:
            raise RuntimeError("dbid is not set.")
        return self.__dbid

    @dbid.setter
    def dbid(self, new_value:int):
        if self.__dbid:
            raise RuntimeError("dbid has already been set.")
        self.__dbid = new_value
```

Лістинг Б.5 – Файл metric_calculator_github_api.py

```
from datetime import datetime
from typing import Any, Dict, List

import re
import requests

from cdpservice.config import CdpServiceConfig
from cdpservice.metric import Metric
from cdpservice.metric_calculator import MetricCalculator
from cdpservice.metric_commit import CommitMetric
from cdpservice.metric_loc import LocMetric
from cdpservice.metric_source_github_api import GitHubApiMetricSource

class GitHubApiMetricCalculator(MetricCalculator):
    """A metrics calculator that uses the GitHub API to calculate LOC and Commit
count metrics.
    """

    def __init__(self, metric_source:GitHubApiMetricSource):
        super().__init__()
        self.__metric_source = metric_source

    @property
    def metric_source(self) -> GitHubApiMetricSource:
```

```
        return self.__metric_source
```

Продовження лістингу Б.5

```
    def calculate(self, since: datetime, until: datetime) -> List[Metric]:
        # https://docs.github.com/en/rest/commits/commits?apiVersion=2022-11-28
        base_url =
f"https://api.github.com/repos/{self.metric_source.owner}/{self.metric_source.re
po}/commits"
        headers = {
            "Authorization": f"token {CdpServiceConfig.GITHUB_TOKEN}",
            "Accept": "application/vnd.github.v3+json"
        }
        params:Dict[str, Any] = {
            "since": since,
            "until": until,
            "author": self.metric_source.login
        }

        resp = requests.get(base_url, headers=headers, params=params, timeout=5)
        resp.raise_for_status()

        commits = resp.json()

        loc = 0
        for commit in commits:
            commit_sha = commit["sha"]
            commit_resp = requests.get(f"{base_url}/{commit_sha}",
headers=headers, timeout=10)
            json_data = commit_resp.json()
            commit_files = json_data.get("files")
            for file in commit_files:
                file_name = file.get("filename")
                if self.metric_source.include and \
                    not any(re.match(p, file_name) for p in
self.metric_source.include):
                    continue
                if self.metric_source.exclude and \
                    any(re.match(p, file_name) for p in
self.metric_source.exclude):
                    continue
                loc += file.get("additions")

        return [LocMetric(loc, source=self.metric_source),
CommitMetric(len(commits), source=self.metric_source)]
```

Лістинг Б.6 – Файл metric_calculator_github_api.py

```
from datetime import datetime
from typing import Any, Dict, List

import os
import shutil
import subprocess
import sys

import pylint.lint
import requests

from cdpservice.config import CdpServiceConfig
from cdpservice.metric import Metric
```

```
from cdpservice.metric_calculator import MetricCalculator
from cdpservice.metric_lint import LintMetric
```

Продовження лістингу Б.6

```
from cdpservice.metric_source_github_clone import GitHubCloneMetricSource

class GitHubPylintMetricCalculator(MetricCalculator):

    def __init__(self, metric_source:GitHubCloneMetricSource):
        super().__init__()
        self.__metric_source = metric_source

    __previous_metric:LintMetric = LintMetric(0, 0)

    @property
    def metric_source(self) -> GitHubCloneMetricSource:
        return self.__metric_source

    def calculate(self, since: datetime, until: datetime) -> List[Metric]:

        if not os.path.exists(CdpServiceConfig.WORKING_DIR):
            os.makedirs(CdpServiceConfig.WORKING_DIR)

        # Obtain the latest commit for specified period and user.
        headers = {
            "Authorization": f"token {CdpServiceConfig.GITHUB_TOKEN}",
            "Accept": "application/vnd.github.v3+json"
        }
        params:Dict[str, Any] = {
            "since": since,
            "until": until,
            "author": self.metric_source.login
        }
        resp = requests.get(

f"https://api.github.com/repos/{self.metric_source.owner}/{self.metric_source.re
po}/commits",
             headers=headers, params=params, timeout=5
        )
        resp.raise_for_status()
        commits = resp.json()

        # Obtain the main branch name
        resp = requests.get(

f"https://api.github.com/repos/{self.metric_source.owner}/{self.metric_source.re
po}",
             headers=headers, params=params, timeout=5
        )
        resp.raise_for_status()
        default_branch = resp.json().get("default_branch", None)

        if len(commits) == 0:
            lint_metric = LintMetric(self.__previous_metric.raw_value,
                                     self.__previous_metric.raw_value, True,
                                     source=self.metric_source)
            self.__previous_metric = lint_metric
            return [lint_metric]


        latest_commit_sha = commits[-1]["sha"]
```

```
        project_url =
f"https://github.com/{self.metric_source.owner}/{self.metric_source.repo}.git"
        project_dir = f"{self.metric_source.owner}_{self.metric_source.repo}"
```

Продовження лістингу Б.6

```
        full_project_dir = os.path.join(CdpServiceConfig.WORKING_DIR,
project_dir)
        auth_project_url = project_url.replace("https://",
f"https://{CdpServiceConfig.GITHUB_TOKEN}@")

        # Clone the specified repository.
        if os.path.exists(full_project_dir):
            try:
                subprocess.run(["git", "checkout", default_branch],
                               check=True, text=True, capture_output=True,
                               cwd=full_project_dir)
                subprocess.run(["git", "pull"],
                               check=True, text=True, capture_output=True,
                               cwd=full_project_dir)
            except subprocess.CalledProcessError as er:
                print(er.output)
                shutil.rmtree(os.path.join(CdpServiceConfig.WORKING_DIR,
project_dir))
                subprocess.run(["git", "clone", auth_project_url, project_dir],
                               check=True, text=True, capture_output=True,
                               cwd=CdpServiceConfig.WORKING_DIR)
        else:
            subprocess.run(["git", "clone", auth_project_url, project_dir],
                           check=True, text=True, capture_output=True,
                           cwd=CdpServiceConfig.WORKING_DIR)

        # Check out the latest commit.
        subprocess.run(["git", "checkout", latest_commit_sha],
                       check=True, text=True, capture_output=True,
                       cwd=full_project_dir)

        # Calculate pylint metrics
        source_dir = full_project_dir
        if self.metric_source.source_dir:
            source_dir = os.path.join(source_dir, self.metric_source.source_dir)
        pylint_args = [source_dir, "--disable=C,R,E0401"]
        linter = None
        with open(os.devnull, "w", encoding="utf-8") as fnull:
            old_stdout = sys.stdout
            old_stderr = sys.stderr
            sys.stdout = fnull
            sys.stderr = fnull
            try:
                linter = pylint.lint.Run(pylint_args, exit=False)
            finally:
                sys.stdout = old_stdout
                sys.stderr = old_stderr

        error_count = sum(linter.linter.stats.by_msg.values())
        lint_metric = LintMetric(error_count, self.__previous_metric.raw_value,
source=self.metric_source)
        self.__previous_metric = lint_metric

        return [lint_metric]
```

Лістинг Б.7 – Файл metric_calculator.py

```
from abc import ABC, abstractmethod
from typing import List
from datetime import datetime
from cdpservice.entity import Entity

from cdpservice.metric import Metric


class MetricCalculator(Entity):

    def __init__(self):
        super().__init__()

    @abstractmethod
    def calculate(self, since:datetime, until:datetime) -> List[Metric]:
        pass
```

Лістинг Б.8 – Файл metric_commit.py

```
from typing import Optional, cast
from cdpservice.config import CdpServiceConfig
from cdpservice.metric import Metric
from cdpservice.metric_source import MetricSource

class CommitMetric(Metric):

    MIN_WEEK_COMMIT = 2

    def __init__(self, raw_value:float, days:int=7,
source:Optional[MetricSource]=None):
        super().__init__(raw_value, source)
        self.__days = days

    @property
    def days(self) -> int:
        return self.__days

    @property
    def weighted_value(self) -> float:
        weighted_value = self.raw_value * \
            CdpServiceConfig.METRIC_WEIGHTED / \
            (CommitMetric.MIN_WEEK_COMMIT)
        if weighted_value > CdpServiceConfig.METRIC_WEIGHTED:
            weighted_value = CdpServiceConfig.METRIC_WEIGHTED
        return weighted_value

    def aggregate(self, other:Metric) -> 'CommitMetric':
        if not isinstance(other, CommitMetric):
            raise RuntimeError("CommitMetric object is expected")
        other = cast(CommitMetric, other)
        return CommitMetric(self.raw_value + other.raw_value, max(self.days,
other.days))
```

Лістинг Б.9 – Файл metric_lint.py

```
from typing import Optional, cast
from cdpservice.config import CdpServiceConfig
from cdpservice.metric import Metric
```

Продовження лістингу Б.9

```
from cdpservice.metric_source import MetricSource

class LintMetric(Metric):

    ERRORS_HIGH_DECREASED = 75

    ERRORS_DECREASED = 99

    ERRORS_EXIST = 50

    ERRORS_INCREASED = 101

    ERRORS_HIGH_INCREASED = 125

    WEIGHT_NONE = 100

    WEIGHT_HIGH_DECREASED = 90

    WEIGHT_DECREASED = 75

    WEIGHT_EXIST = 50

    WEIGHT_INCREASED = 25

    WEIGHT_HIGH_INCREASED = 0

    def __init__(self, raw_value:float, prev_raw_value:float=0,
changes:bool=True, source:Optional[MetricSource]=None):
        super().__init__(raw_value, source)
        self.__changes = changes
        self.__prev_raw_value = prev_raw_value
        if not changes and raw_value:
            raise RuntimeError("raw_value must be zero when changes is False")

    def aggregate(self, other:Metric) -> 'LintMetric':
        if not isinstance(other, LintMetric):
            raise RuntimeError("LintMetric object is expected")
        other = cast(LintMetric, other)
        return LintMetric(
            self.raw_value + other.raw_value,
            self.prev_raw_value + other.prev_raw_value,
            self.changes or other.changes
        )

    @property
    def changes(self):
        return self.__changes

    @property
    def prev_raw_value(self):
        return self.__prev_raw_value

    @property
    def weighted_value(self) -> float:
        if not self.changes:
            return 0
        norm_value = 0
        if self.prev_raw_value:
            change = self.raw_value * 100 / self.prev_raw_value
            if change <= LintMetric.ERRORS_HIGH_DECREASED:
                norm_value = LintMetric.WEIGHT_HIGH_DECREASED
```

Продовження лістингу Б.9

```
            elif change <= LintMetric.ERRORS_DECREASED:
                norm_value = LintMetric.WEIGHT_DECREASED
            elif change >= LintMetric.ERRORS_HIGH_INCREASED:
                norm_value = LintMetric.WEIGHT_HIGH_INCREASED
            elif change >= LintMetric.ERRORS_INCREASED:
                norm_value = LintMetric.WEIGHT_INCREASED
            else:
                norm_value = LintMetric.WEIGHT_EXIST
        else:
            if self.raw_value == 0:
                norm_value = LintMetric.WEIGHT_NONE
            else:
                norm_value = LintMetric.WEIGHT_EXIST

        weighted_value = CdpServiceConfig.METRIC_WEIGHTED * norm_value /
LintMetric.WEIGHT_NONE
        if weighted_value > CdpServiceConfig.METRIC_WEIGHTED:
            weighted_value = CdpServiceConfig.METRIC_WEIGHTED

        return weighted_value

    def __str__(self):
        return f"{self.__class__.__name__}, raw:{self.raw_value},
rate:{self.prev_raw_value})"
```

Лістинг Б.10 – Файл metric_lint.py

```
from typing import Optional, cast
from cdpservice.metric import Metric
from cdpservice.config import CdpServiceConfig
from cdpservice.metric_source import MetricSource

class LocMetric(Metric):
    """Line of Code Metric

    This metric uses raw_value property as the number of added lines of code.
    """

    MIN_DAY_LOC = 20

    def __init__(self, raw_value:float, days:int=7,
source:Optional[MetricSource]=None):
        super().__init__(raw_value, source)
        self.__days = days

    @property
    def days(self) -> int:
        return self.__days

    @property
    def weighted_value(self) -> float:
        weighted_value = self.raw_value * \
            CdpServiceConfig.METRIC_WEIGHTED / \
            (LocMetric.MIN_DAY_LOC * self.__days)
        if weighted_value > CdpServiceConfig.METRIC_WEIGHTED:
            weighted_value = CdpServiceConfig.METRIC_WEIGHTED
        return weighted_value

    def aggregate(self, other:Metric) -> 'LocMetric':
```

Продовження лістингу Б.10

```
        if not isinstance(other, LocMetric):
            raise RuntimeError("LocMetric object is expected")
        other = cast(LocMetric, other)
        return LocMetric(self.raw_value + other.raw_value, max(self.days,
other.days))
```

Лістинг Б.11 – Файл metric_source_github_api.py

```
from typing import List, Optional
from cdpservice.metric_source import MetricSource

class GitHubApiMetricSource(MetricSource):

    def __init__(self, login:str, owner:str, repo:str,
                 include:Optional[List[str]]=None,
exclude:Optional[List[str]]=None):
        super().__init__()
        self.__login = login
        self.__owner = owner
        self.__repo = repo
        self.__include_regx = include
        self.__exclude_regx = exclude

    @property
    def login(self) -> str:
        return self.__login

    @property
    def owner(self) -> str:
        return self.__owner

    @property
    def repo(self) -> str:
        return self.__repo

    @property
    def include(self) -> Optional[List[str]]:
        return self.__include_regx

    @property
    def exclude(self) -> Optional[List[str]]:
        return self.__exclude_regx
```

Лістинг Б.12 – Файл metric_source_github_api.py

```
from cdpservice.metric_source_github_api import GitHubApiMetricSource

class GitHubCloneMetricSource(GitHubApiMetricSource):

    def __init__(self, login, owner:str, repo:str, source_dir:str):
        super().__init__(login, owner, repo)
        self.__source_dir = source_dir

    @property
    def source_dir(self) -> str:
        return self.__source_dir
```

Лістинг Б.13 – Файл metric_source.py

```
from cdpservice.entity import Entity

class MetricSource(Entity):
    def __init__(self):
        super().__init__()
```

Лістинг Б.14 – Файл metric.py

```
from abc import ABC, abstractmethod
from typing import Optional
from cdpservice.entity import Entity

from cdpservice.metric_source import MetricSource

class Metric(Entity):

    def __init__(self, raw_value:float, source:Optional[MetricSource]=None):
        super().__init__()
        self.__source = source
        self.__raw_value = raw_value

    @abstractmethod
    def aggregate(self, other:'Metric') -> 'Metric':
        pass

    @property
    def source(self) -> Optional[MetricSource]:
        return self.__source

    @property
    def raw_value(self) -> float:
        return self.__raw_value

    @property
    @abstractmethod
    def weighted_value(self) -> float:
        pass

    def __str__(self):
        return f"{self.__class__.__name__}, raw:{self.raw_value})"
```

Лістинг Б.15 – Файл productivity_calculator.py

```
from typing import Dict, Tuple, cast
from typing import List
from typing import TypedDict

from cdpservice.metric import Metric

class ProductivityCalculator:

    def calculate(self, metrics:List[Metric]) -> Tuple[float, List[Metric]]:

        grouped: Dict[type, Metric] = {}
        for metric in metrics:
            metric_type = type(metric)
            if not metric_type in grouped:
                grouped[metric_type] = metric
```

Продовження лістингу Б.15

```
            else:
                grouped[metric_type] = metric.aggregate(grouped[metric_type])

        productivity:float = 0
        for metric in grouped.values():
            productivity += cast(Metric, metric).weighted_value
        productivity = productivity / len(grouped)

        return productivity, list(grouped.values())
```

Лістинг Б.16 – Файл productivity.py

```
from typing import List
from cdpservice.developer import Developer

from cdpservice.metric import Metric


class Productivity:
    def __init__(self, developer:Developer, value:float,
                 metrics:List[Metric], grouped_metrics:List[Metric]):
        self.__developer = developer
        self.__value = value
        self.__metrics = metrics
        self.__grouped_metrics = grouped_metrics

    @property
    def developer(self):
        return self.__developer

    @property
    def value(self):
        return self.__value

    @property
    def metrics(self):
        return self.__metrics

    @property
    def grouped_metrics(self):
        return self.__grouped_metrics
```

Лістинг Б.17 – Файл service.py

```
from datetime import datetime
from typing import List, Optional
from cdpservice.calculation import Calculation

from cdpservice.developer import Developer
from cdpservice.productivity import Productivity
from cdpservice.productivity_calculator import ProductivityCalculator
from cdpservice.storage import Storage

import cdpservice.tools.datetime_tools as datetime_tools

class Service:
    """The entry-point class for all Complex Developer Productivity service
functionality."""
```

Продовження лістингу Б.17

```
    def __init__(self):
        self.__storage = Storage()

    def get_developers(self) -> List[Developer]:
        return self.__storage.get_developers()

    def save_calculation(self, calculation:Calculation):
        self.__storage.save_calculation(calculation)

    def calculate(self, year:int, week:int, developers:List[Developer]) ->
Calculation:

        prod_list:List[Productivity] = []
        prod_calc = ProductivityCalculator()
        since, until = datetime_tools.get_week_datetimes(year, week)

        for developer in developers:
            metrics = []
            for calc in developer.calculators:
                metrics += calc.calculate(since, until)
            prod_value, grouped_metrics = prod_calc.calculate(metrics)
            prod_list.append(Productivity(
                developer, prod_value, metrics, grouped_metrics))

        calculation = Calculation(year, week, prod_list)
        return calculation
```

Лістинг Б.18 – Файл cdpservice_tests/test_developer_credentials.py

```
import sys
from unittest import TestCase

if sys.gettrace() is not None:
    sys.path.append(".")

from cdpservice.metric_source_github_api import GitHubApiMetricSource

class DeveloperCredentialsTests(TestCase):

    def test_developer_github_credentials(self):
        cred = GitHubApiMetricSource("login1", "owner1", "repo1")
        self.assertEqual("login1", cred.login)
```

Лістинг Б.19 – Файл cdpservice_tests/test_developer.py

```
import sys
import unittest

if sys.gettrace() is not None:
    sys.path.append(".")

from cdpservice.developer import Developer

class DeveloperTests(unittest.TestCase):

    def test_creation(self):
        name = "Developer Name"
```

Продовження лістингу Б.19

```
        dev = Developer(name, [])
        self.assertEqual(name, dev.name)
        with self.assertRaises(AttributeError):
            dev.name = "New Name"


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

Лістинг Б.20 – Файл cdpservice_tests/test_metric_calculator.py

```
import sys
import unittest

if sys.gettrace() is not None:
following string:
    sys.path.append(".")

from cdpservice.metric_calculator_github_pylint import
GitHubPylintMetricCalculator
from cdpservice.metric_calculator_github_api import GitHubApiMetricCalculator
from cdpservice.metric_source_github_api import GitHubApiMetricSource
from cdpservice.metric_source_github_clone import GitHubCloneMetricSource


class MetricSourceTests(unittest.TestCase):

    def test_github_api_metric_calculator_constructor(self):

        source = GitHubApiMetricSource("Gillibald", "AvaloniaUI", "Avalonia")
        calc = GitHubApiMetricCalculator(source)
        since = "2023-10-09T00:00:00Z"
        until = "2023-10-15T23:59:59Z"
        calc.calculate(since, until)

    def test_github_api_metric_calculator(self):
        source = GitHubApiMetricSource("Gillibald", "AvaloniaUI", "Avalonia",
[r".*\.cs"])
        calculator = GitHubApiMetricCalculator(source)
        since = "2023-10-09T00:00:00Z"
        until = "2023-10-15T23:59:59Z"
        metrics = calculator.calculate(since, until)
        self.assertEqual(637, metrics[0].raw_value)
        self.assertEqual(10, metrics[1].raw_value)

    def test_github_pylint_metric_source(self):
        metric_source = GitHubCloneMetricSource("tiangolo", "tiangolo",
"fastapi", "fastapi")
        calculator = GitHubPylintMetricCalculator(metric_source)
        since = "2022-01-03T00:00:00Z"
        until = "2022-01-09T23:59:59Z"
        metrics = calculator.calculate(since, until)
        self.assertNotEqual(0, metrics[0].raw_value)

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

Лістинг Б.21 – Файл cdpservice_tests/test_metric.py

```
import sys
import unittest

from cdpservice.metric_lint import LintMetric
from cdpservice.metric_loc import LocMetric

if sys.gettrace() is not None:
    sys.path.append(".")

class MetricTests(unittest.TestCase):

    def test_metric_creation(self):
        pass

    def test_loc_metric_half(self):

        metric = LocMetric(10, 1)
        self.assertAlmostEqual(50, metric.weighted_value)

    def test_loc_metric_high(self):

        metric = LocMetric(160)
        self.assertAlmostEqual(100, metric.weighted_value)

    def test_loc_metric_zero(self):

        metric = LocMetric(0)
        self.assertAlmostEqual(0, metric.weighted_value)

    def test_lint_metric(self):

        metric = LintMetric(91, 100)
        self.assertAlmostEqual(metric.WEIGHT_DECREASED, metric.weighted_value)

    def test_dbid(self):

        metric = LocMetric(20)
        metric.dbid = 1

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

Лістинг Б.22 – Файл cdpservice_tests/test_productivity_calculator.py

```
import unittest

from cdpservice.metric_commit import CommitMetric
from cdpservice.metric_loc import LocMetric
from cdpservice.metric_lint import LintMetric
from cdpservice.productivity_calculator import ProductivityCalculator

class ProductivityCalculatorTest(unittest.TestCase):

    def test_productivity(self):
        metrics = [
            LocMetric(10),
            LocMetric(10),
            CommitMetric(1),
            CommitMetric(1),
```

Продовження лістингу Б.22

```
            LintMetric(8, 10),
            LintMetric(10, 9),
        ]
        productivity_calculator = ProductivityCalculator()
        productivity, _ = productivity_calculator.calculate(metrics)
        self.assertAlmostEqual(63.1, round(productivity,2))
```

Лістинг Б.23 – Файл cdpservice_tests/test_storage.py

```
import os
import unittest

from sqlalchemy import MetaData

from cdpservice.storage import DbDeveloper, Storage

TEST_DB_FILE = "db-test.sqlite"
TEST_DB_CONN = f"sqlite:///{TEST_DB_FILE}"

class StorageTest(unittest.TestCase):

    def test_storage_creation(self):
        storage = Storage(TEST_DB_CONN)
        self.assertEqual(True, os.path.exists(TEST_DB_FILE))
        metadata = MetaData()
        metadata.reflect(bind=storage.get_engine())
        metadata.drop_all(bind=storage.get_engine())

    def test_add_delete_developer(self):
        storage = Storage(TEST_DB_CONN)
        session = storage.get_session()
        db_developer = DbDeveloper(name="Developer 1", calculators=[])
        self.assertFalse(db_developer.id)
        session.add(db_developer)
        session.commit()
        self.assertTrue(dbid := db_developer.id)
        session.delete(db_developer)
        session.commit()
        self.assertFalse(
            session.query(DbDeveloper)
            .filter(DbDeveloper.id == dbid).first())
        metadata = MetaData()
        metadata.reflect(bind=storage.get_engine())
        metadata.drop_all(bind=storage.get_engine())
```

Додаток В

Лістинг коду модулів збереження даних

Лістинг В.1 – Файл storage.py

```
import json
from typing import List, Optional

from datetime import datetime
from sqlalchemy import Engine, ForeignKey, String, UniqueConstraint,
create_engine
from sqlalchemy.orm import DeclarativeBase, Mapped, Session, mapped_column,
relationship
from cdpservice.calculation import Calculation
from cdpservice.config import CdpServiceConfig

from cdpservice.developer import Developer
from cdpservice.metric import Metric
from cdpservice.metric_calculator import MetricCalculator
from cdpservice.metric_calculator_github_api import GitHubApiMetricCalculator
from cdpservice.metric_calculator_github_pylint import
GitHubPylintMetricCalculator
from cdpservice.metric_commit import CommitMetric
from cdpservice.metric_lint import LintMetric
from cdpservice.metric_loc import LocMetric
from cdpservice.metric_source import MetricSource
from cdpservice.metric_source_github_api import GitHubApiMetricSource
from cdpservice.metric_source_github_clone import GitHubCloneMetricSource

# https://docs.sqlalchemy.org/en/20/orm/quickstart.html#declare-models

class Base(DeclarativeBase):
    pass

class DbDeveloper(Base):
    __tablename__ = "developer"
    id:Mapped[int] = mapped_column(primary_key=True)
    name:Mapped[str] = mapped_column(unique=True, nullable=False)
    calculators:Mapped[List["DbMetricCalculator"]] =
relationship(back_populates="developer")
    sources:Mapped[List["DbMetricSource"]] =
relationship(back_populates="developer")

class DbMetricSourceType(Base):
    __tablename__ = "metric_source_type"
    id:Mapped[int] = mapped_column(primary_key=True)
    name:Mapped[str] = mapped_column(String(100), unique=True, nullable=False)

class DbMetricSource(Base):
    __tablename__ = "metric_source"
    id:Mapped[int] = mapped_column(primary_key=True)
    source_type_id:Mapped[int] =
mapped_column(ForeignKey("metric_source_type.id", ondelete="RESTRICT"),
nullable=False)
    developer_id:Mapped[int] = mapped_column(ForeignKey("developer.id",
ondelete="CASCADE"), nullable=False)
    json:Mapped[str] = mapped_column(nullable=False)

    developer:Mapped["DbDeveloper"] = relationship(back_populates="sources")
```

Продовження лістингу В.1

```
    source_type:Mapped["DbMetricSourceType"] = relationship()

class DbMetricCalculatorType(Base):
    __tablename__ = "metric_calculator_type"
    id:Mapped[int] = mapped_column(primary_key=True)
    name:Mapped[str] = mapped_column(String(100), unique=True, nullable=False)
    source_type_id:Mapped[int] =
mapped_column(ForeignKey("metric_source_type.id", ondelete="RESTRICT"),
nullable=False)

    source_type:Mapped[DbMetricSourceType] = relationship()

class DbMetricCalculator(Base):
    __tablename__ = "metric_calculator"
    id:Mapped[int] = mapped_column(primary_key=True)
    calculator_type_id:Mapped[int] =
mapped_column(ForeignKey("metric_calculator_type.id", ondelete="RESTRICT"),
nullable=False)
    developer_id:Mapped[int] = mapped_column(ForeignKey("developer.id",
ondelete="CASCADE"), nullable=False)
    source_id:Mapped[int] = mapped_column(ForeignKey("metric_source.id",
ondelete="CASCADE"), nullable=False)
    json:Mapped[str] = mapped_column(nullable=False)

    developer:Mapped["DbDeveloper"] = relationship(back_populates="calculators")
    source:Mapped["DbMetricSource"] = relationship()

class DbMetricType(Base):
    __tablename__ = "metric_type"
    id:Mapped[int] = mapped_column(primary_key=True)
    name:Mapped[str] = mapped_column(String(100))

class DbCalculation(Base):
    __tablename__ = "calculation"
    id:Mapped[int] = mapped_column(primary_key=True)
    year:Mapped[int] = mapped_column(nullable=False)
    week:Mapped[int] = mapped_column(nullable=False)
    __table_args__ = (
        UniqueConstraint(year, week, name="uc_year_week"),
    )

class DBMetric(Base):
    __tablename__ = "calculation_metric"
    id:Mapped[int] = mapped_column(primary_key=True)
    metric_type_id:Mapped[int] = mapped_column(ForeignKey("metric_type.id",
ondelete="RESTRICT"), nullable=False)
    calculation_id:Mapped[int] = mapped_column(ForeignKey("calculation.id",
ondelete="CASCADE"), nullable=False)
    source_id:Mapped[int] = mapped_column(ForeignKey("metric_source.id",
ondelete="RESTRICT"), nullable=False)
    raw_value:Mapped[float] = mapped_column(nullable=False)

    metric_type:Mapped[DbMetricType] = relationship()
    calculation:Mapped[DbCalculation] = relationship()
    source:Mapped[DbMetricSource] = relationship()

class DbGroupedMetric(Base):
    __tablename__ = "calculation_grouped_metric"
    id:Mapped[int] = mapped_column(primary_key=True)
    metric_type_id:Mapped[int] = mapped_column(ForeignKey("metric_type.id",
ondelete="RESTRICT"), nullable=False)
```

Продовження лістингу В.1

```
    calculation_id:Mapped[int] = mapped_column(ForeignKey("calculation.id",
ondelete="CASCADE"), nullable=False)
    developer_id:Mapped[int] = mapped_column(ForeignKey("developer.id",
ondelete="CASCADE"))
    raw_value:Mapped[float] = mapped_column(nullable=False)
    weighted_value:Mapped[float] = mapped_column(nullable=False)


    metric_type:Mapped[DbMetricType] = relationship()
    calculation:Mapped[DbCalculation] = relationship()
    developer:Mapped[DbDeveloper] = relationship()

class DbProductivity(Base):
    __tablename__ = "calculation_productivity"
    id:Mapped[int] = mapped_column(primary_key=True)
    calculation_id:Mapped[int] = mapped_column(ForeignKey("calculation.id",
ondelete="CASCADE"), nullable=False)
    developer_id:Mapped[int] = mapped_column(ForeignKey("developer.id",
ondelete="CASCADE"), nullable=False)
    productivity:Mapped[float] = mapped_column(nullable=False)

    calculation:Mapped[DbCalculation] = relationship()
    developer:Mapped[DbDeveloper] = relationship()

class Storage():

    METRIC_TYPE_ID__LOC = 1
    METRIC_TYPE_ID__COMMIT = 2
    METRIC_TYPE_ID__LINT = 3

    METRIC_SOURCE_ID__GITHUB_API = 1
    METRIC_SOURCE_ID__GITHUB_CLONE = 2

    METRIC_CALCULATOR_ID__GITHUB_API = 1
    METRIC_CALCULATOR_ID__GITHUB_PYLINT = 2

    def __init__(self, engine_str:Optional[str]=None):
        if not Storage.__engine:
            if engine_str:
                Storage.__engine = create_engine(engine_str)
            else:
                Storage.__engine = create_engine(CdpServiceConfig.DB_CONN)
        # https://docs.sqlalchemy.org/en/20/tutorial/metadata.html#emitting-ddl-
to-the-database-from-an-orm-mapping
        Base.metadata.create_all(Storage.__engine)
        Storage.__initialize()

    def get_engine(self) -> Engine:
        if not Storage.__engine:
            raise RuntimeError((
                "Storage.__engine can't be None "
                "when it is used via a class instance."))
        return Storage.__engine

    def get_session(self) -> Session:
        return Session(Storage.__engine)

    def get_developers(self, session:Optional[Session]=None) -> List[Developer]:
        if not session:
            session = Session(Storage.__engine)
        db_developers = session.query(DbDeveloper).all()
```

Продовження лістингу В.1

```
        developers:List[Developer] = []
        for db_developer in db_developers:
            calculators:List[MetricCalculator] = []
            for db_calculator in db_developer.calculators:
                calculator = Storage.db_2_calculator(db_calculator)
                calculators.append(calculator)
            developer = Developer(db_developer.name, calculators)
            developer.dbid = db_developer.id
            developers.append(developer)
        return developers

    def save_calculation(self, calculation:Calculation,
session:Optional[Session]=None):

        if not session:
            session = self.get_session()

        db_calculation = DbCalculation(year=calculation.year,
week=calculation.week)
        session.add(db_calculation)

        for productivity in calculation.productivity_list:

            db_productivity = DbProductivity(
                calculation=db_calculation,
                developer_id=productivity.developer.dbid,
                productivity = productivity.value)
            session.add(db_productivity)

            for metric in productivity.metrics:
                db_metric = DBMetric(
                    metric_type_id=self.__get_db_metric_type(metric),
                    calculation=db_calculation,
                    source_id=metric.source.dbid,
                    raw_value=metric.raw_value
                )
                session.add(db_metric)

            for metric in productivity.grouped_metrics:
                db_grouped_metric = DbGroupedMetric(
                    metric_type_id=self.__get_db_metric_type(metric),
                    calculation=db_calculation,
                    developer_id=productivity.developer.dbid,
                    raw_value=metric.raw_value,
                    weighted_value=metric.weighted_value
                )
                session.add(db_grouped_metric)

        session.commit()


    __engine:Optional[Engine] = None


    @classmethod
    def db_2_source(cls, db_source:DbMetricSource) -> MetricSource:
        data = json.loads(db_source.json)
        if Storage.METRIC_SOURCE_ID__GITHUB_API == db_source.source_type_id:
            github_api_metric_source = GitHubApiMetricSource(
                data["login"], data["owner"], data["repo"],
                data["include"], data["exclude"])
```

Продовження лістингу В.1

```
            github_api_metric_source.dbid = db_source.id
            return github_api_metric_source
        elif Storage.METRIC_SOURCE_ID__GITHUB_CLONE == db_source.source_type_id:
            github_clone_metric_source = GitHubCloneMetricSource(
                data["login"], data["owner"],
                data["repo"],data["source_dir"])
            github_clone_metric_source.dbid = db_source.id
            return github_clone_metric_source
        else:
            raise NotImplementedError(
                (f"Metric source type {db_source.source_type_id} is unknown. "
                 f"Override the db_2_source method if the support of new "
                 f"metric source types is necessary."))

    @classmethod
    def db_2_calculator(cls, db_calculator:DbMetricCalculator) ->
MetricCalculator:
        source = Storage.db_2_source(db_calculator.source)
        if Storage.METRIC_CALCULATOR_ID__GITHUB_API ==
db_calculator.calculator_type_id:
            if not isinstance(source, GitHubApiMetricSource):
                raise ValueError("The source variable must be
GitHubApiMetricSource")
            calculator_github_api = GitHubApiMetricCalculator(source)
            calculator_github_api.dbid = db_calculator.id
            return calculator_github_api
        elif Storage.METRIC_CALCULATOR_ID__GITHUB_PYLINT ==
db_calculator.calculator_type_id:
            if not isinstance(source, GitHubCloneMetricSource):
                raise ValueError("The source variable must be
GitHubPylintMetricCalculator")
            calculator_github_clone = GitHubPylintMetricCalculator(source)
            calculator_github_clone.dbid = db_calculator.id
            return calculator_github_clone
        else:
            raise NotImplementedError(
                (f"Metric calculator type {db_calculator.calculator_type_id} is
unknown. "
                 f"Override the db_2_calculator method if the support of new "
                 f"metric source types is necessary."))

    @classmethod
    def __initialize(cls):
        session = Session(cls.__engine)
        if session.query(DbMetricSourceType).first():
            return
        # Metric Type
        metric_type_loc = DbMetricType(
            id=Storage.METRIC_TYPE_ID__LOC, name="LOC"
        )
        metric_type_commit = DbMetricType(
            id=Storage.METRIC_TYPE_ID__COMMIT, name="Commit"
        )
        metric_type_lint = DbMetricType(
            id=Storage.METRIC_TYPE_ID__LINT, name="Linting"
        )
        session.add_all([metric_type_loc, metric_type_commit, metric_type_lint])
        # Metric Source Type
        github_api_metric_source_type = DbMetricSourceType(
            id=Storage.METRIC_SOURCE_ID__GITHUB_API,
            name="GitHub API Metric Source")
```

Продовження лістингу В.1

```
        github_clone_metric_source_type = DbMetricSourceType(
            id=Storage.METRIC_SOURCE_ID__GITHUB_CLONE,
            name="GitHub Clone Metric Source"
        )
        github_api_metric_calculator_type = DbMetricCalculatorType(
            id=Storage.METRIC_CALCULATOR_ID__GITHUB_API,
            name="GitHub API Metric Calculator",
            source_type=github_api_metric_source_type
        )
        session.add_all([github_api_metric_source_type,
github_clone_metric_source_type])
        #Metric Calculator Type
        github_pylint_metric_calculator_type = DbMetricCalculatorType(
            id=Storage.METRIC_CALCULATOR_ID__GITHUB_PYLINT,
            name="GitHub Pylint Metric Calculator",
            source_type=github_clone_metric_source_type
        )
        session.add_all([github_api_metric_calculator_type,
github_pylint_metric_calculator_type])
        session.commit()

    @classmethod
    def __get_db_metric_type(cls, metric:Metric):
        if isinstance(metric, LocMetric):
            return Storage.METRIC_TYPE_ID__LOC
        elif isinstance(metric, CommitMetric):
            return Storage.METRIC_TYPE_ID__COMMIT
        elif isinstance(metric, LintMetric):
            return Storage.METRIC_TYPE_ID__LINT
        else:
            raise RuntimeError(f"Undefined metric type {type(metric).__name__}")
```

Додаток Г

Лістинг коду використання програмного інтерфейсу сервісу для конфігурування, розрахунку продуктивності

Лістинг Г.1 – Файл usage_data_addition.py – заповнення бази даних системи вхідними даними через програмний інтерфейс

```
import json

from collections import namedtuple

from cdpservice.storage import Storage, DbDeveloper
from cdpservice.storage import DbMetricSource, DbMetricCalculator

PY_RE = r".*\.py"

SourceData = namedtuple("SourceData", ["owner", "repo", "source_dir"])

sources = [
    SourceData(owner="scrapy", repo="scrapy", source_dir="scrapy"),
    SourceData(owner="scrapinghub", repo="scrapy-poet",
source_dir="scrapy_poet"),
    SourceData(owner="scrapinghub", repo="web-poet", source_dir="web_poet"),
    SourceData(owner="scrapy-plugins", repo="scrapy-zyte-api",
source_dir="scrapy_zyte_api"),
    SourceData(owner="scrapy-plugins", repo="scrapy-spider-metadata",
source_dir="scrapy_spider_metadata"),
    SourceData(owner="scrapy-plugins", repo="scrapy-zyte-smartproxy",
source_dir="scrapy_zyte_smartproxy"),
    SourceData(owner="zytedata", repo="zyte-common-items", source_dir=
"zyte_common_items"),
    SourceData(owner="zytedata", repo="zyte-spider-templates", source_dir=
"zyte_spider_templates"),
    SourceData(owner="zytedata", repo="zyte-spider-templates-project",
source_dir= "zyte_spider_templates_project"),
    SourceData(owner="zytedata", repo="python-zyte-api", source_dir=
"zyte_api"),
    SourceData(owner="zytedata", repo="url-matcher", source_dir= "url_matcher"),
    SourceData(owner="zytedata", repo="zyte-parsers", source_dir=
"zyte_parsers"),
]

storage = Storage()
session = storage.get_session()

developer1 = DbDeveloper(name="Andrey Rakhmatullin") # @wRAR
https://github.com/wRAR
developer2 = DbDeveloper(name="Adrián Chaves") # @Gallaecio
https://github.com/Gallaecio
developer3 = DbDeveloper(name="Mikhail Korobov") # kmike
https://github.com/kmike
session.add(developer1)
session.add(developer2)
session.add(developer3)

developers = [developer1, developer2, developer3]
developer_logins = ["wRAR", "Gallaecio", "kmike"]
```

Продовження лістингу Г.1

```
for i in range(3):
    for source in sources:
        developer_source_api =
DbMetricSource(source_type_id=storage.METRIC_SOURCE_ID__GITHUB_API,
            developer=developers[i],
json=json.dumps({"login":developer_logins[i], "owner":source.owner,
"repo":source.repo, "include":PY_RE, "exclude":""}))
        developer_source_clone =
DbMetricSource(source_type_id=storage.METRIC_SOURCE_ID__GITHUB_CLONE,
            developer=developers[i],
json=json.dumps({"login":developer_logins[i], "owner":"scrapy", "repo":"scrapy",
"source_dir":source.source_dir}))
        session.add(developer_source_api)
        session.add(developer_source_clone)
        developer_calculator_api = DbMetricCalculator(
            calculator_type_id=storage.METRIC_CALCULATOR_ID__GITHUB_API,
            developer=developers[i], source=developer_source_api, json="")
        developer_calculator_pylint = DbMetricCalculator(
            calculator_type_id=storage.METRIC_CALCULATOR_ID__GITHUB_PYLINT,
            developer=developers[i], source=developer_source_clone, json="")
        session.add(developer_calculator_api)
        session.add(developer_calculator_pylint)

session.commit()
```

Лістинг Г.2 – Файл usage_data_calculate_cmd.py – Консольний додаток для запуску розрахунку продуктивності у вказаний тиждень

```
import argparse

from cdpservice.service import Service
from cdpservice.storage import Storage

if __name__ == "__main__":

    argparser = argparse.ArgumentParser()
    argparser.add_argument("-y", "--year", type=int, required=True)
    argparser.add_argument("-w", "--week", type=int, required=True)
    args = argparser.parse_args()

    year = args.year
    week = args.week

    storage = Storage()
    service = Service()
    developers = storage.get_developers()

    calculation = service.calculate(year, week, developers)
    storage.save_calculation(calculation)

    print("The calculation has been done successfully.")
```

Лістинг Г.3 – Файл usage_data_calculation_service.py – Веб-сервіс для запуску розрахунку продуктивності через мережу

```
from flask import Flask
from flask_restful import Resource, Api

import threading
import uuid

from cdpservice.service import Service
from cdpservice.storage import Storage

app = Flask(__name__)
api = Api(app)

calculations = {}

def long_running_calculate(year, week, calc_id):
    calculations[calc_id] = 'In Progress'
    try:
        storage = Storage()
        service = Service()
        developers = storage.get_developers()
        calculation = service.calculate(year, week, developers)
        storage.save_calculation(calculation)
        calculations[calc_id] = 'Completed'
    except Exception as e:
        calculations[calc_id] = f'Failed: {str(e)}'

class CalculateResource(Resource):
    def get(self, year, week):
        calc_id = str(uuid.uuid4())
        threading.Thread(target=long_running_calculate, args=(year, week,
calc_id)).start()
        return {"calc_id": calc_id}, 202

class StatusResource(Resource):
    def get(self, calc_id):
        status = calculations.get(calc_id, "Not Found")
        return {"status": status}

api.add_resource(CalculateResource, '/calculate/<int:year>/<int:week>')
api.add_resource(StatusResource, '/status/<calc_id>')

if __name__ == '__main__':
    app.run(debug=True)
```

Додаток Д

Презентаційні матеріали

Рисунок Д.1 – Титульний слайд презентації

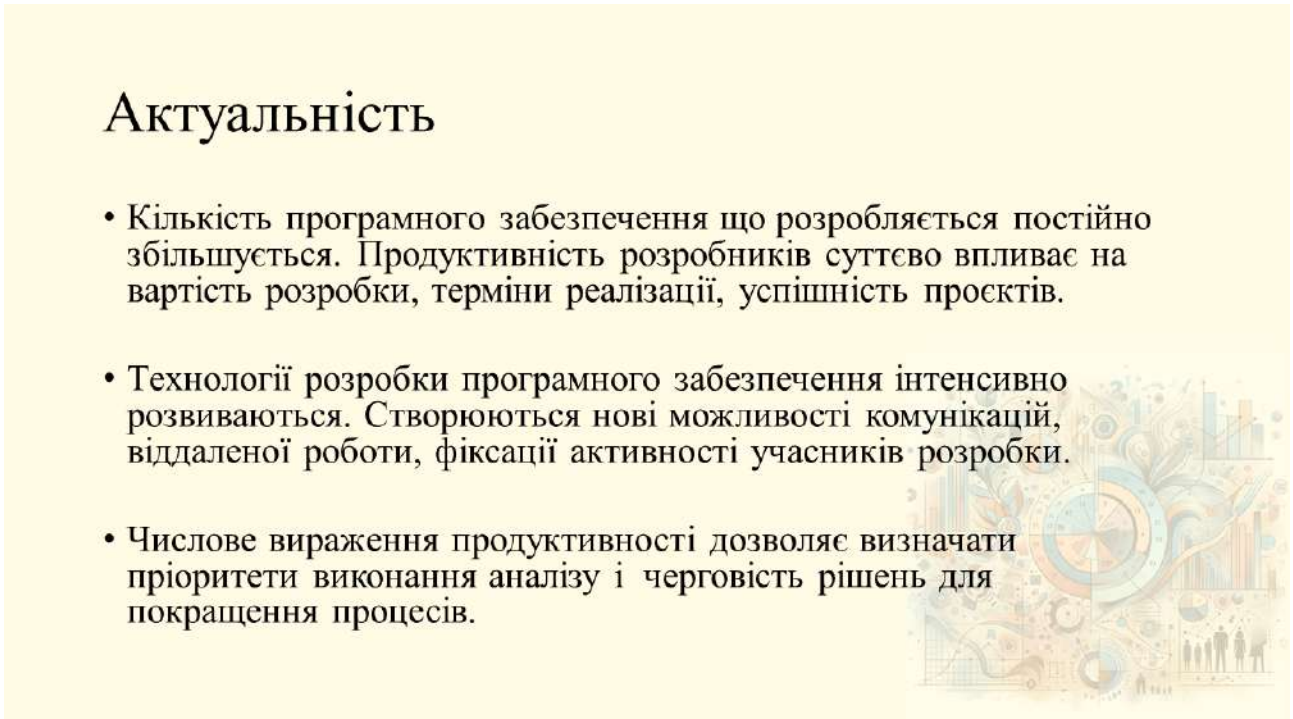


Рисунок Д.2 – Слайд «Актуальність»

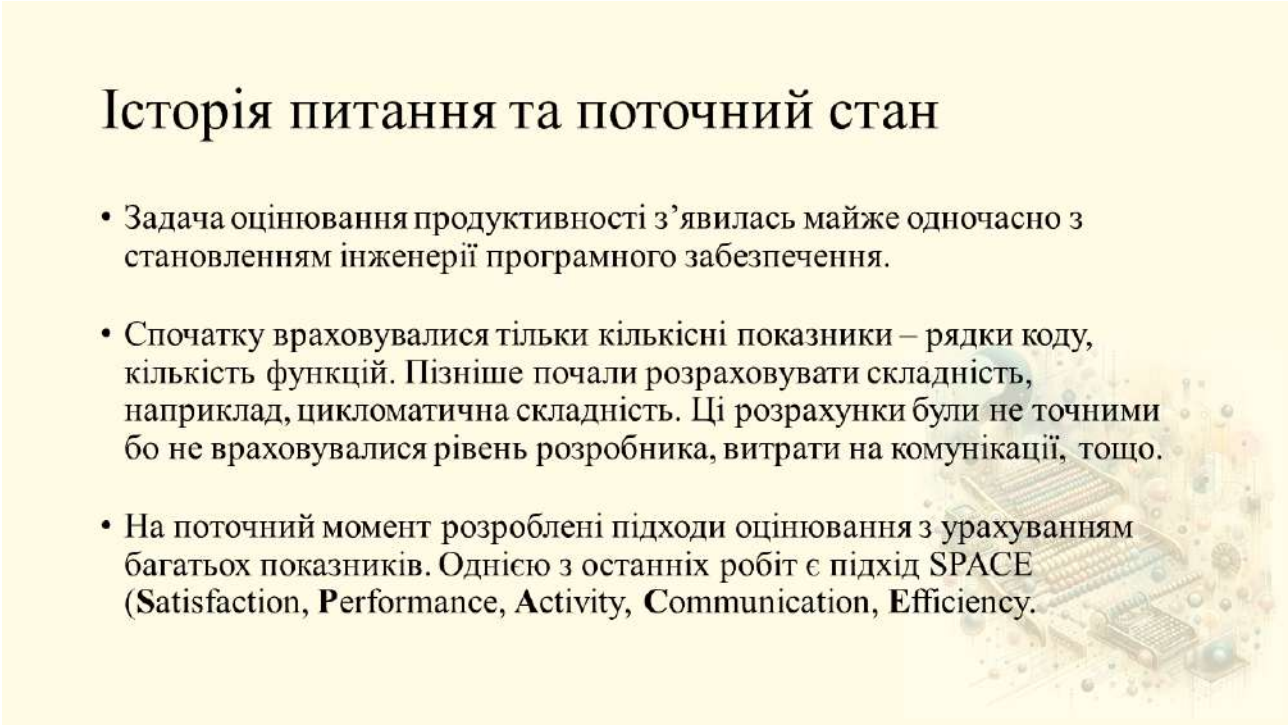


Рисунок Д.3 – Слайд «Історія питання та поточний стан»

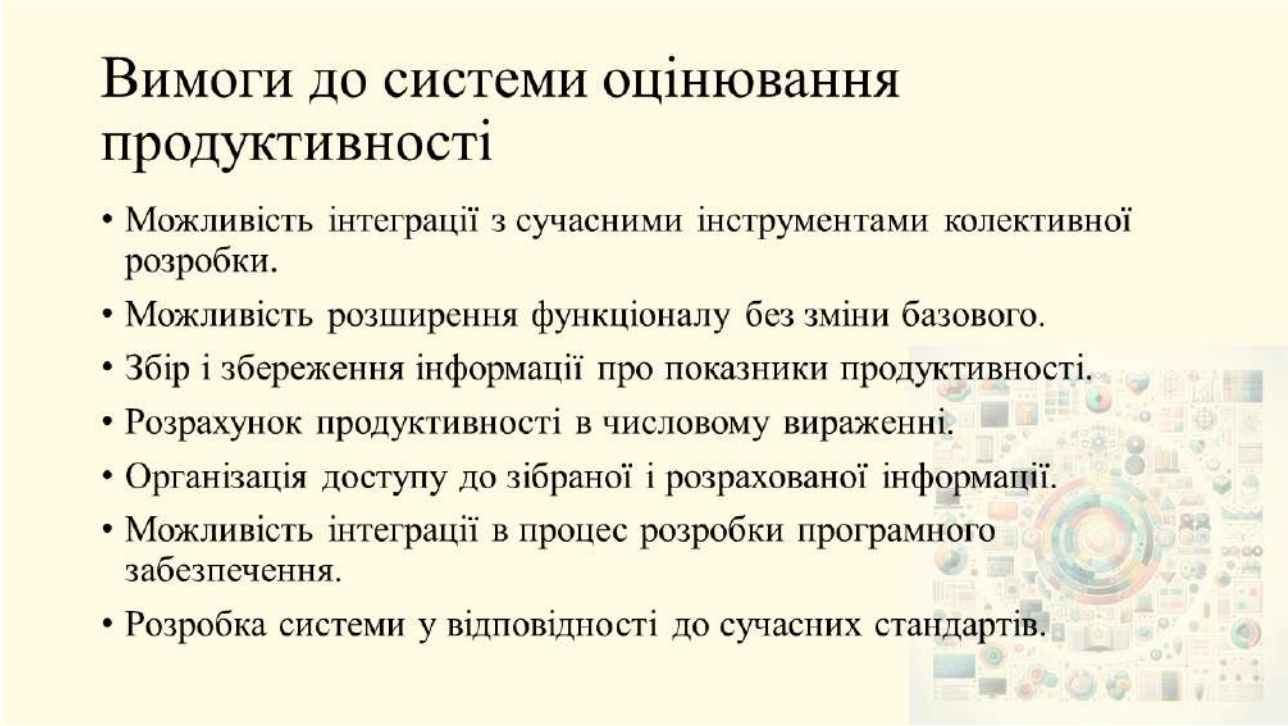


Рисунок Д.4 – Слайд «Вимоги до системи оцінювання продуктивності»

Математичне обґрунтування

Продуктивність, розрахована за одним критеріем: $P_1 = \frac{R}{E}$

де R – отримані результати, E – докладені зусилля, E це час і завжди дорівнює 1.

Продуктивність, розрахована за одним критеріем:

$$P_C = \frac{1}{N}\sum_{i=0}^{N-1} W_i \cdot R_{RN_i}$$

де P_C – числове вираження комплексної продуктивності, N – кількість показників продуктивності що вимірюються, W_i – ваговий коефіцієнт i-го показника продуктивності, R_{RN_i} – ранжоване та нормоване значення i-го показника продуктивності

Вимога до вагових коефіцієнтів:

$$\sum_{i=0}^{N-1} W_i = 1$$

Рисунок Д.5 – Слайд «Математичне обґрунтування»

Метрики які будуть збиратись для проведення дослідження

- LOC – кількість рядків коду (кількісний показник). Це базова метрика і завжди може використовуватись як один з показників. Значення отримане з досвіду – 10-50 рядків змістовного коду в день. Враховується абсолютне значення.
- COMMIT – кількість збережень в систему контролю версій (комунікативний показник). Значення отримане з досвіду – 2-3 збереження коду в тиждень. Враховується абсолютне значення.
- LINT – оцінка якості коду інструментом Lint (PyLint) (якісний показник). Враховується зміна значення у порівнянні з попередніми розрахунками.

Рисунок Д.6 – Слайд «Метрики які будуть збиратися для проведення дослідження»

Рисунок Д.7 – Слайд «Високорівнева архітектура системи»

Рисунок Д.8 – Слайд «Вибір мови програмування»

Рисунок Д.9 – Слайд «Логіка розрахунку продуктивності»

Рисунок Д.10 – Слайд «Збереження параметрів та результатів»

Варіанти використання системи

- Програмний інтерфейс – дозволяє автоматичне виконання шляхом інтеграції програмного коду з іншими системами.
- Інтерфейс командного рядку – дозволяє автоматичне виконання шляхом виконання команди в командному рядку в одному середовищі.
- Веб RESTful інтерфейс – дозволяє автоматизувати виконання шляхом виконання мережевого запиту.

Рисунок Д.11 – Слайд «Варіанти використання системи»

Реалістичні тестові дані

Розробник 1, Розробник 2, Розробник 3

НАЗВА ПРОЄКТУ	ПОСИЛАННЯ
Scrapy	https://github.com/scrapy/scrapy/
Scrapy Poet Extension	https://github.com/scrapinghub/scrapy-poet/
Scrapy Web Poet Extension	https://github.com/scrapinghub/web-poet/
Scrapy Zyte API	https://github.com/scrapy-plugins/scrapy-zyte-api/
Scrapy Spider Metadata	https://github.com/scrapy-plugins/scrapy-spider-metadata/
Scrapy Zyte SmartProxy	https://github.com/scrapy-plugins/scrapy-zyte-smartproxy/
Zyte Common Items	https://github.com/zytedata/zyte-common-items/
Zyte Spider Templates	https://github.com/zytedata/zyte-spider-templates/
Zyte Spider Templates Project	https://github.com/zytedata/zyte-spider-templates-project/
Python Zyte API	https://github.com/zytedata/python-zyte-api/
Zyte URL Matcher	https://github.com/zytedata/url-matcher/
Zyte Parsers	https://github.com/zytedata/zyte-parsers/

Рисунок Д.12 – Слайд «Реалістичні тестові дані»

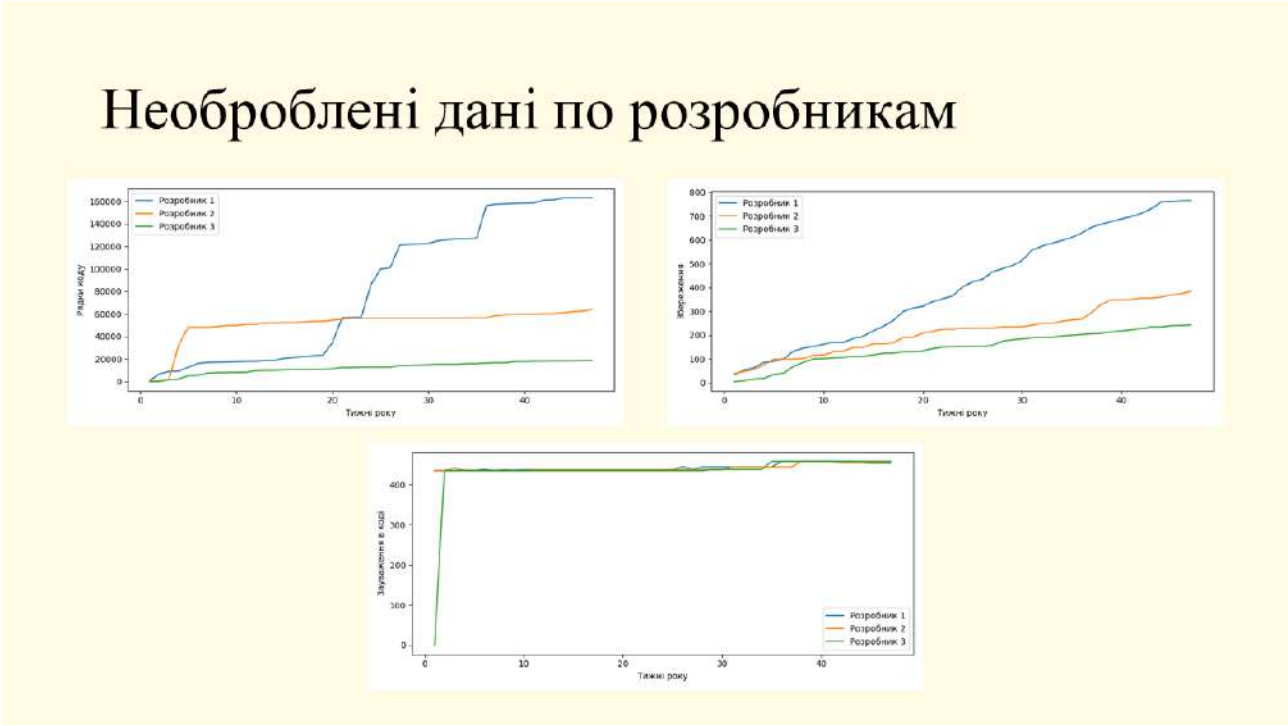


Рисунок Д.13 – Слайд «Необроблені дані по розробникам»

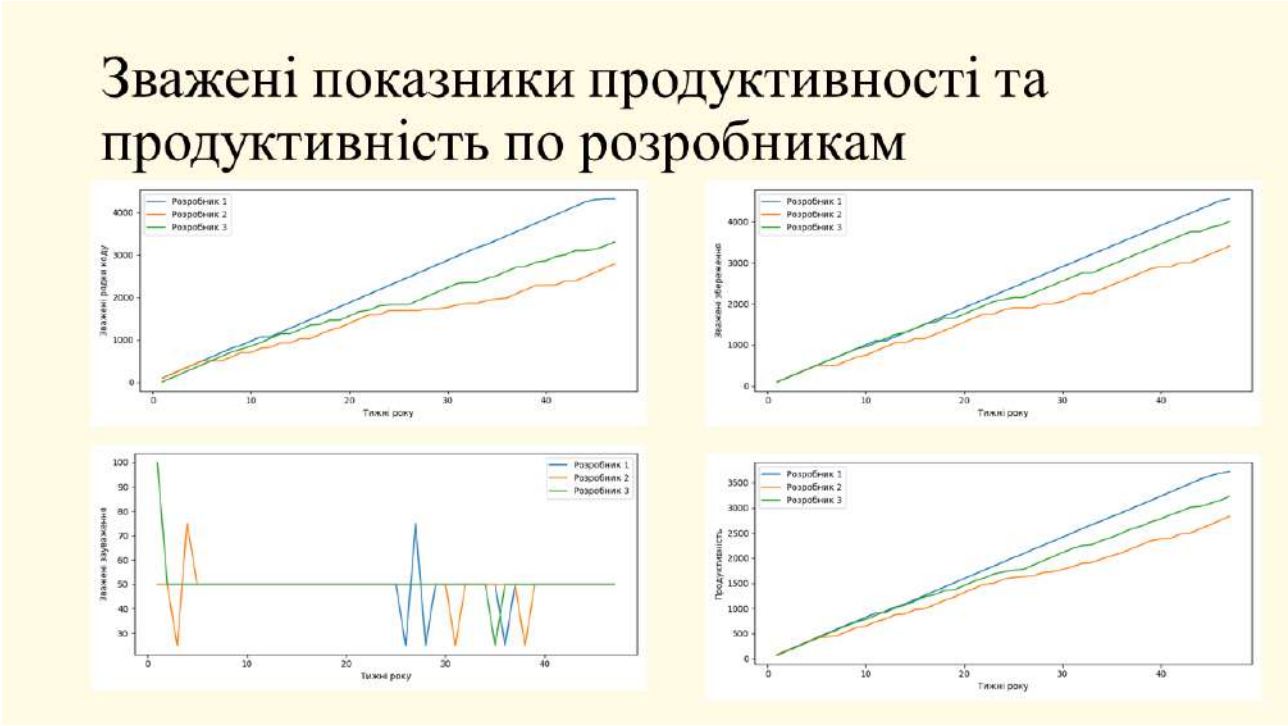


Рисунок Д.14 – Слайд «Зважені показники продуктивності та продуктивність по розробникам»

Продуктивність по проєкту

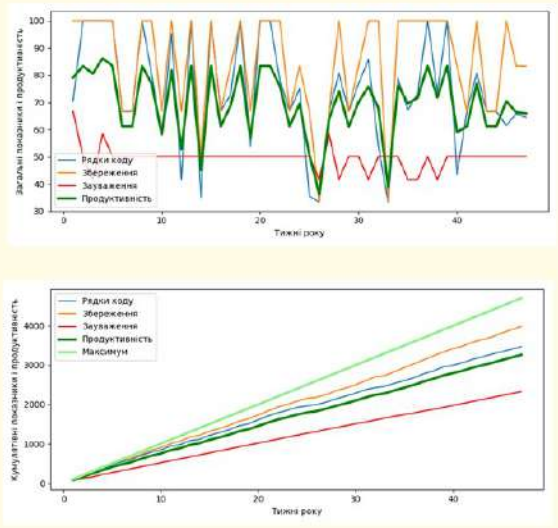


Рисунок Д.15 – Слайд «Продуктивність по проєкту»

Висновки

- На поточний момент існують всі умови для розвитку систем комплексної оцінки продуктивності.
- Python як мова програмування підходить для створення прототипів систем обробки даних.
- Окремий аналіз прямих показників продуктивності не дає можливості зробити об’єктивні висновки.
- Числове значення продуктивності отримане шляхом розрахунку багатьох параметрів є більш об’єктивним, дозволяє бачити пріоритет проблем і визначати порядок роботи над покращеннями.
- Використання систем оцінювання продуктивності має бути обов’язковим етапом організації процесу розробки.

Рисунок Д.16 – Слайд «Висновки»

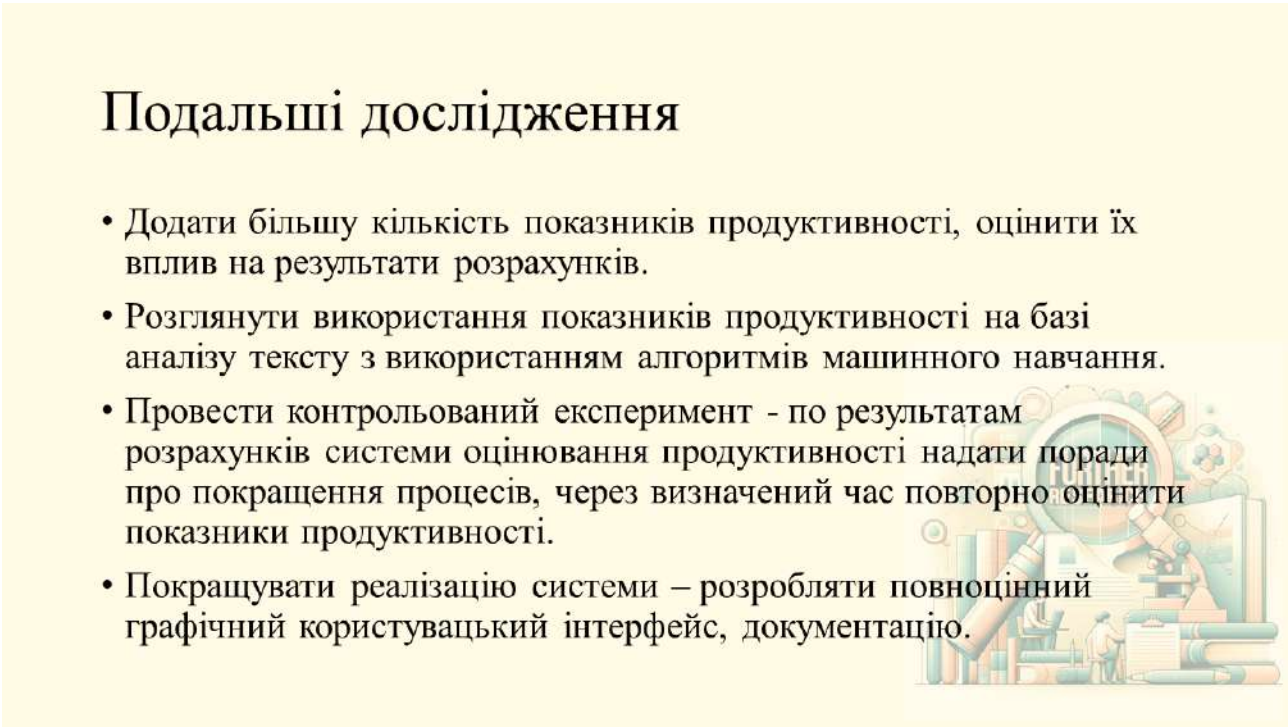


Рисунок Д.17 – Слайд «Подальші дослідження»