ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до магістерської роботи
на ступінь вищої освіти магістр

на тему: **«ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ОЦІНКИ СУПРОВОДЖЕНОСТІ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МЕТОДАМИ СТАТИСТИЧНОГО АНАЛІЗУ»**

Виконав: студент 7 курсу, групи ППЗ-71
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Лихвар А.В.
(прізвище та ініціали)

Керівник Негоденко О.В.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти – «Магістр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного
забезпечення
_____Негоденко О.В.___
“____” _____2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТА

ЛИХВАРА АРТЕМА ВОЛОДИМИРОВИЧА
(прізвище, ім’я, по батькові)

1. Тема роботи: «Підвищення ефективності оцінки супроводжуваності об’єктно-орієнтованого програмного забезпечення методами статистичного аналізу
2. Керівник роботи: Негоденко О.В., к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом закладу вищої освіти від «16» лютого 2022 року №22.
3. Строк подання студентом роботи «1» червня 2022 року
4. Вхідні дані до роботи
Системна програмна документація;
Документація на операційні системи;
Описання файлових систем;
5. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).

5.1. Проаналізува існуючі моделі та метрики вимірювання об’єктно-орієнтованого програмного забезпечення, визначені їх переваги та недоліки.
5.2. Модефікувати модель дельта супроводженості об’єктно-орієнтованог; програмного забезпечення.
5.3. Провести практичну валідацію запропонованої моделі.

6. Графічна частина роботи представлена на 17 слайдах презентації.

1. Мета, об’єкт та предмет дослідження;
2. Існуючі моделі якості програмного забезпечення;
3. Існуючі метрики програмного забезпечення;
4. Модель дельта супроводженості об’єктно-орієнтованого програмного забезпечення DMM;
5. Математична модель дельта супроводженості об’єктно-орієнтованого програмного забезпечення;
6. Модифікована модель дельта супроводженості об’єктно-орієнтованого програмного забезпечення;
7. Показники вхідних програмних продуктів для порівняльного аналізу;
8. Результати вимірювання показників моделі DMM та DMM+;
9. Зміни в показнику абсолютної середньої різниці між показниками DMMS та DMMS+;
10. Порівняльна таблиця метричних даних змін;
11. Висновки

7. Дата видачі завдання «25» лютого 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02-27.02	Виконано
2	Вибір середовища програмування і компонентів програмної утиліти	27.02-01.03	Виконано
3	Аналіз існуючих моделей та метрик вимірювання якості програмного забезпечення	01.03-04.03	Виконано
4	Модефіковати модель дельта супроводженості об’єктно-орієнтованого програмного забезпечення	05.03-25.03	Виконано
5	Вступ, висновки, реферат	28.03-05.04	Виконано
6	Розробка обов‘язкових демонстраційних матеріалів	06.04-26.04	Виконано
7	Попередній захист роботи	29.04	Виконано
9	Пред’явлення роботи в деканат	01.06	Виконано

Студент Лихвар А.В.
(підпис) (прізвище та ініціали)

Керівник роботи Негоденко О. В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 63 с., 10 рис., 12 табл., 46 джерел.

PYTHON, ДЕЛЬТА ПОКАЗНИК, МОДЕЛІ DMM, ДЕФЕКТИ, ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

Об'єкт – процес супроводженння об'єктно-орієнтованого програмного забезпечення.

Предмет – методи та моделі підтримки супроводженння об'єктно-орієнтованого програмного забезпечення.

Мета – підвищення якості та рівня супроводженості об'єктно-орієнтованого програмного забезпечення за рахунок зменшення кількості дефектів.

Методи дослідження – науково-дослідний з використанням сучасних комп'ютерних технологій, методи верифікації та валідації.

Прогнозні припущення щодо розвитку об'єкта – супроводженння об'єктно-орієнтованого програмного забезпечення шляхом модифікація моделі дельта супроводженості (DMM) шляхом розширення вимірюваних властивостей вихідного коду.

Для досягнення поставленої мети доцільно провести аналіз різних визначень та аспектів супроводженості, а також усталених моделей та підходів вимірювання об'єктно орієнтованого програмного забезпечення, з метою визначення можливостей підвищення ефективності оцінки методами статистичного аналізу.

Результати дипломного дослідження рекомендується використовувати при розробці та супроводження об'єктно-орієнтованого програмного забезпечення.

ЗМІСТ

ВСТУП **9**

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ ДОСЛІДЖЕННЯ **10**

1.1 Аналіз сучасних моделей якості програмного забезпечення 11

1.1.1 Моделі якості МакКола та Боема ..... **Error! Bookmark not defined.**

1.1.2 Моделі якості, які спираються на характеристики продукту . **Error! Bookmark not defined.**

1.1.3 Модель якості об’єктно-орієнтованого дизайну (QMOOD) . **Error! Bookmark not defined.**

1.1.4 Модель якості ISO/IEC 25010 17

1.2 Постановка завдань дослідження 19

2 МЕТРИКИ ДЛЯ ВИМІРЮВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ **27**

2.1 Основні метрики для вимірювання якості програмного забезпечення 28

2.2 Метрики об’єктно-орієнтованого дизайну 33

3 МЕТРИЧНИЙ АНАЛІЗ СУПРОВОДЖЕНОСТІ **28**

3.1 Модель супроводженості (SIG-MM) 29

3.2 Модель дельта супроводженості (DMM) 34

3.3 Модефікована модель дельта супроводженості об’єктно-орієнтованого програмного забезпечення 36

ВИСНОВКИ **44**

ПЕРЕЛІК ПОСИЛАНЬ **46**

ДОДАТОК 1 ..... **51**

ДОДАТОК 2 ..... **52**

ДОДАТОК 3 ..... **54**

ДОДАТОК 4 ..... **55**

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ **57**

ВСТУП

Сучасні системи програмного забезпечення все глибше інтегруються в життєво важливі сфери, від керування критичної інфраструктури до пілотування транспортними засобами. Саме тому одним із найважливіших пріоритетів є зменшення можливих дефектів програмних засобів**.** Швидкість розвитку суспільних процесів та технологій обумовлює необхідність адаптації, що в свою чергу вимагає внесення коригувань програмного забезпечення.

Актуальним питанням залишається аналіз різних визначень та аспектів супроводженості, а також усталених моделей та підходів вимірювання об’єктно-орієнтованого програмного забезпечення, з метою визначення можливостей підвищення ефективності оцінки методами статистичного аналізу.

Об’єкт – процес супроводженння об’єктно-орієнтованого програмного забезпечення.

Предмет – методи та моделі підтримки супроводженння об’єктно-орієнтованого програмного забезпечення.

Мета – підвищення якості та рівня супроводженості об’єктно-орієнтованого програмного забезпечення за рахунок зменшення кількості дефектів.

Методи дослідження – науково-дослідний з використанням сучасних комп’ютерних технологій, методи верифікації та валідації.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати існуючі моделі та метрики вимірювання об’єктно-орієнтованого програмного забезпечення, визначети їх переваги та недоліки.

2. Удосконалити модель дельта супроводженості об’єктно-орієнтованого програмного забезпечення;

3. Визначення їх взаємозв’язку з підхарактеристиками супроводженості, що визначені ISO/IEC 25010:2016;

4. Визначити способи вимірювання та порогові значення;

5. Провести практичну валідацію запропонованої моделі шляхом аналізу програмних продуктів з відкритим вихідним кодом, імплементованих із застосуванням об’єктно-орієнтованої парадигми програмування, процесом розробки із застосуванням систем контролю версій та значною кількістю учасників процесу розробки та тривалою історією змін коду.

Прогнозні припущення щодо розвитку об’єкта включають супроводженння об’єктно-орієнтованого програмного забезпечення шляхом модифікація моделі дельта супроводженості (DMM) шляхом розширення вимірюваних властивостей вихідного коду.

Важливо продемонструвати стабільність та ефективність вимірювання змін об’єктно-орієнтованого програмного забезпечення шляхом порівняльного аналізу внесених змін вихідного коду, що дає можливість проведення вимірювань супроводженості в процесах з методологічними підходами безперервної доставки та неперервної інтеграції. При цьому інтерпретація результатів оцінювання дає можливість проведення причинно-наслідкового зв’язку та усунення недоліків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ ДОСЛІДЖЕННЯ

Швидкість розвитку суспільних процесів та технологій обумовлює необхідність адаптації, що в свою чергу вимагає внесення коригувань програмного забезпечення. Дослідження C. Jones демонструє стійкий рост залучення інженерів до робот із підтримки ПЗ, з 9.09 % відсотків у 1950 році до 72.73% у 2000 році та прогнозованою долею залучення 77.27% у 2025 році [1]. Вимоги до підвищення рівня якості та супроводженості є причиною багатьох дослідженнь та постійного пошуку методологій вимірювання та оцінки програмного забезпечення. Супроводженість програмного забезпечення є добре дослідженою темою. Більшість досліджень обгрунтовується важливістю супроводженості програмного забезпечення, підкреслюючи взаємозв'язок з адаптивністю та зменшення кількості дефектів.

Відповідно до звіту The Consortium for Information & Software Quality (CISQ) загальні витрати від використання програмного забезпечення низької якості (CPSQ) склали $ 2.08 трлн. при цьому, витрати викликані супроводженістю програмного забезпечення низької якості у 2020 році склали $ 520 млрд. Це демонструє відносне зменшення видатків порівняно із 2018 роком, які за оцінками склали $ 635 млрд., але здебільшого обумовлюється заміною неякісного програмного забезпечення [2]. Зазначені проблеми можуть мати різні причини, але напряму чи опосередковано всі вони пов'язані із якістю. Якість програмного забезпечення, зокрема, супроводженість відрізняються від інших властивостей складністю факторів, що їх обумовлюють, тому їх коригування майже завжди є складним та довготривалим.

1.1 Аналіз сучасних моделей якості програмного забезпечення

Моделі якості програмного забезпечення призначені для визначення критеріїв оцінки якості, та визначення термінології. Незважаючи на відмінність, існуючі моделі визначають супроводженість як один із головних атрибутів якості програмного забезпечення. Таким чином більшість методів вимірювання супроводженості, враховують не тільки супроводженість, а є частиною моделей якості.

1.1.1 Моделі якості МакКола та Боема

Модель, що також відома як Модель Дженерал Електрікс, було розроблено для військових відомств США та запропоно МакКолом в 1997 році [38], складається з 3 рівневої ієрархії та визначає взаємозв'язок між атрибутами якості програмного забезпечення. Модель визначає наступні критерії якості, що пов'язані із внутрішніми факторами якості.

- Використання продукту, визначає аспекти якості продукту під час його використання;
- Перенесення продукту, визначається як здатність продукту до зберігання робочого стану під час змін оточення;
- Модифікування продукту, стосується всіх аспектів виправлення помилок та адаптації системи.

Супроводженість, як внутрішній фактор якості пов'язується із Модифікуванням продукту, і визначається як обсяг зусиль необхідних для визначення та виправлення дефекту програми в операційному середовищі. Супроводженість як і всі внутрішні фактори якості вимірюється опосередковано, через пов'язані властивості програмного забезпечення:

- Простота;
- Краткість;
- Інформативність;
- Модульність.

Вимірювання властивостей пропонується здійснювати шляхом ранжування від 1 (ціль не досягнуто) до 10 (відмінна реалізація), без зазначення конкретних метрик чи способів вимірювання.

Запропонована в 1978 році модель Боема [39] є схожою із моделлю МакКола, оскільки також має ієрархічну структуру, що складається з 3 сегментів характеристик. Боем визначає “загальну корисність”, як основу якості програмного забезпечення, що має найвищий пріоритет, і підпорядковує усі інші характеристики. Загальна якість системи визначається сумарним значенням всіх характеристик. Наступний рівень, який визначає “загальну корисність” містить утилітарність, сіпроводженість та портативність, при цьому перші дві характеристики мають додатковий рівень підхарактеристик. На останньому рівні ієрархії модель визначає первинні характеристики, які саме і призначені для вимірювання. При цьому модель не передбачає будь яких визначень або формулювань таких вимірювань або показників.

Відповідно до моделі супроводженість визначається рівнями ієрархії (Рис.1.1).

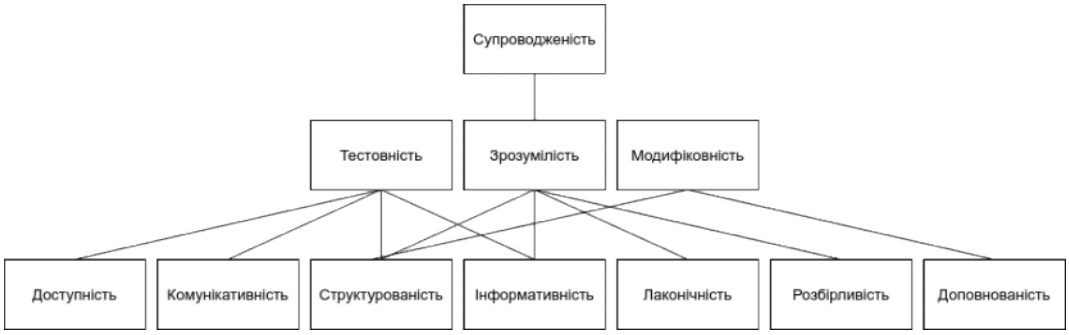


Рис.1.1 – Рівні ієрархії моделі супроводженості

1.1.2 Моделі якості, які спираються на характеристики продукту

Запропонована Друмі [40] модель якості спирається на характеристики продукту, що мають бути використані для показників та вимірювання атрибутів якості. Таким чином модель визначає взаємозв’язки між властивостями програмного продукту, та безпосередньо, атрибутами якості,

оскільки, за припущенням Друмі, властивості програмного забезпечення за своєю природою, не демонструють якість. Модель визначає супроводженість як атрибут якості пов’язаний із характеристиками продукту.

В 1992 році Граді було представлено модель якості FURPS [41]. FURPS є акронімом назв п’яти трибутів: Functionality (функцональність), Usability (зручність використання), Reliability (надійність), Performance (продуктивність) and Supportability (підтримуваність). Вказані атрибути призначені для визначення якості програмного забезпечення. Для кожного з атрибутів, модель визначає під атрибути. Супроводженість визначається як під атрибут пітримуваності.

Модель якості ISO/IEC 9126

Результатом необхідності стандартизації визначень та методологій є серія стандартів ISO/IEC 9126 Оцінювання програмного продукту. Характеристики якості й настанови щодо їх використання.Частина перша стандарту - ISO/IEC 9126-1 “Програмна інженерія. Якість продукту. Частина 1. Модель якості.” [12], що має відповідний національний стандарт ДСТУ [13] містить модель якості програмного забезпечення. Модель є ієрархічною, і складається з 6 характеристик якості продукту, які в подальшому пов’язуться з 27 підхарактеристкиами якості.

Стандарт визначає супроводженість як високорівневу характеристику якості продукту, що пов’язаний із 4 під характеристиками. властивостями вихідного коду програми, однак не містить формулювання чи визначення метрик (Рис.1.2).

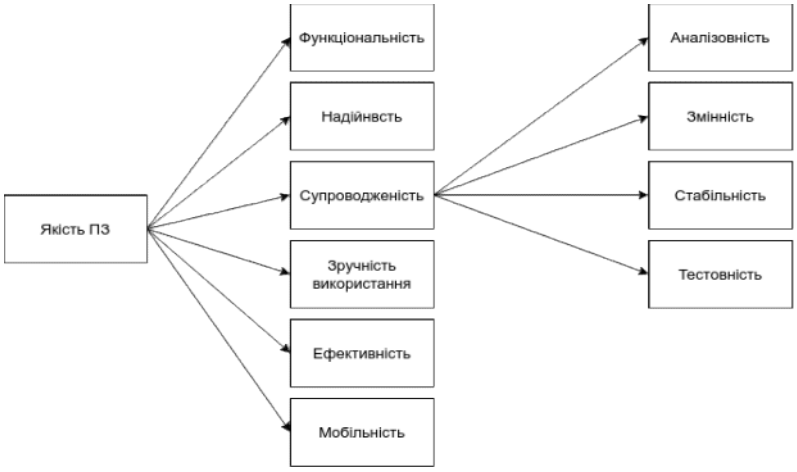


Рис.1.2 – Високорівнева характеристика якості продукту

В подальшому, стандарт був переглянутий та замінений стандартом ISO/IEC 25010:2011 [12].

1.1.3 Модель якості об’єктно-орієнтованого дизайну (QMOOD)

Модель якості об’єктно-орієнтованого дизайну (QMOOD) було запропоновано Бансія та Девісом в 2002 році [32]. Зазначена модель складається з 4 рівневої ієрархічної структури. Модель також включає набір метрик призначених для вимірювання атрибутів якості. В основі визначення атрибутів якості, з деякими модифікаціями є модель якості програмного забезпечення ISO 9126 [12]. Такожі модель визначає атрибути дизайну та відповідні їм метрики. Атрибути дизайну в свою чергу, пов’язуються із атрибутами якості. Супроводженість, що визначена ISO 9126 як атрибут якості передбачає певну стадію завершеності програмного забезпечення, тому модель сфокусирована лише на його підхарактеристиці - зрозумілості, що має дозволити використовувати модель на більш ранніх стадіях розробки.

Встановлено зв'язки властивостей дизайну та атрибутів якості [32]. Основні метрики для вимірювання властивостей дизайну включають:

Розмір дизайну в класах (DSC) є кількістю всіх класів передбачених дизайном програми.

Кількість ієрархічних структур (NOH) є кількістю ієрархій класів програми.

Середня кількість спадкодавців (ANA) є середнім значенням кількості класів, які наслідуються класом, та обчислюється підрахунком кількості класів по всіх шляхах спадкування від кореневого класу (класів) до всіх класів структури успадкування.

Метрика доступу до даних (DAM) є значенням в діапазоні від 0 до 1, та є співвідношенням кількості приватни (захищених) атрибутів класу до загальної кількості атрибутів визначених класом.

Пряма з’єднаність класів (DCC) є значенням кількості класів від яких клас залежить безпосередньо, або через визначення атрибуту або передання повідомлення (параметрів) в методах.

Пов’язаність методів класу (CAM) є значенням в діапазоні від 0 до 1, та є обрахунком взаємозалежності методів класу на підставі списку параметрів методів [33]. Метрика обчислюється шляхом підсумовування перетину параметрів методу з максимально незалежним набором параметрів всіх типів в класі.

Таблиця 1. Матриця зв'язків властивостей дизайну та атрибутів якості

		Атрибут якості					
		Повторна використовн.	Гнучкість	Зрозумілість	Функціональність	Розширюваність	Ефективність
Атрибут дизайну	Розмір дизайну	?			?		
	Ієрархічність				?		
	Абстрактність					?	?
	Інкапсуляція		?	?			?
	З’єднаність						
	Пов’язаність	?		?	?		
	Композиція		?				?
	Успадкування					?	?
	Поліморфізм		?		?	?	?
	Обмін повідомленням	?			?		

	и						
	Складність						

Міра агрегації (MOA) ця метрика вимірює ступінь зв'язку частина-ціле, реалізована за допомогою атрибутів. Значення є сумою кількості задекларованих даних, типи яких є визначеними користувачем класами.

Міра функціональної абстракції (MFA) є значенням в діапазоні від 0 до 1, є співвідношенням кількості методів успадкованих класом до загальної кількості методів, що мають доступ з інших методів класу.

Кількість поліморфних методів (NOP) є сумою кількості методів, що можуть демонструвати поліморфну поведінку.

Розмір інтерфейсу класу (CIS) є кількісним показником підрахунку публічних методів класу.

Кількість методів (NOM) є кількісним показником підрахунку всіх методів визначених класом.

Таблиця 2. Таблиця метрик для вимірювання властивостей дизайну

Властивість дизайну	**Метрика**
Розмір дизайну	Розмір дизайну в класах **(DSC)**
Ієрархічність	Кількість ієрархічних структур **(NOH)**
Абстрактність	Середня кількість спадкодавців **(ANA)**
Інкапсуляція	Метрика доступу до даних **(DAM)**
З’єднаність	Пряма з’єднаність класів **(DCC)**
Пов’язаність	Пов’язаність методів класу **(CAM)**
Композиція	Міра агрегації **(MOA)**
Успадкування	Міра функціональної абстракції **(MFA)**
Поліморфізм	Кількість поліморфних методів **(NOP)**
Обмін повідомленнями	Розмір інтерфейсу класу **(CIS)**
Складність	Кількість методів **(NOM)**

1.1.4 Модель якості ISO/IEC 25010

Стандарт ISO/IEC 25010:2016 “Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів” [10, 11] є заміною стандарту ISO 9126 [12]. Стандарт розширює модель якості двома новими високорівневими характеристиками: сумісність (є новою характеристикою), та захищеність (в попередньому стандарті є підхарактеристикою функційної придатності).

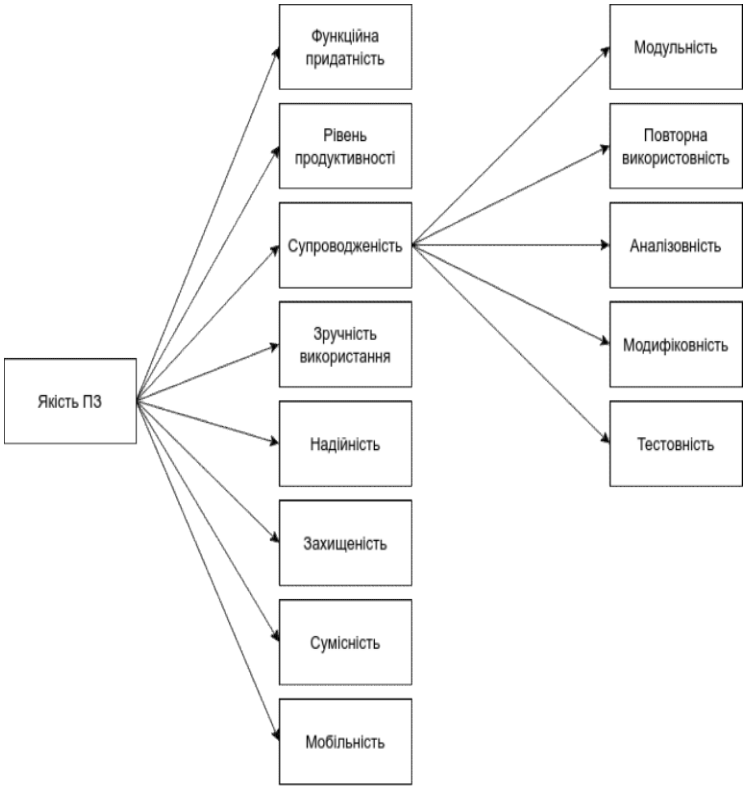


Рис.1.3 – Високорівнева характеристика якості продукту за ISO/IEC 25010

Також стандартом внесені зміни в структуру супроводженості. Введені нові підхарактеристики: “модульність” та “повторна використовність”. Підхарактеристики “змінність” та “стабільність” були замінені новою характеристикою “модифіковність”.

Стандарт дає наступні визначення супроводженості:

Супроводженість визначається як ступінь результативності та

ефективності, в якому продукт або система можуть бути модифі-ковані призначеними фахівцями з обслуговування При цьому супроводженість можна інтерпретувати як притаманну продукту або системі здатність сприяти полегшенню робіт з технічного обслуговування або як якість під час застосування;

Модульність це ступінь, в якому систему або комп'ютерну програму складено з розрізнених компонентів, таких, що зміна в одному компоненті має мінімальний вплив на інші компоненти;

Повторна використовність є ступінь, в якому актив може бути застосовано більш ніж в одній системі або у побудові інших активів;

Аналізовність визначається як ступінь результативності та ефективності, в якому можна оцінити вплив на продукт або систему передбаченої зміни в одній чи кількох частинах або діагностувати некомплект, причини відмов, чи ідентифікувати частини, які будуть модифіковані.

Модифіковність є ступінь результативності та ефективності, в якому продукт або система можуть бути модифі-ковані без внесення дефектів або деградації існуючої якості продукту. Модифіковність є комбінацією змінності та стабільності.

Тестовність це ступінь результативності та ефективності, в якому критерії тестування можна встановлювати для системи, продукту або компонента, а тести треба виконувати, щоб визначити, чи досягнуто цих критеріїв.

1.2 Постановка завдань дослідження

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати існуючі моделі та метрики вимірювання об'єктно-орієнтованого програмного забезпечення, визначети їх переваги та недоліки.

2. Удосконалити модель дельта супроводженості об’єктно-орієнтованого програмного забезпечення;

3. Визначення їх взаємозв’язку з підхарактеристиками супроводженості, що визначені ISO/IEC 25010:2016;

4. Визначити способи вимірювання та порогові значення;

5. Провести практичну валідацію запропонованої моделі шляхом аналізу програмних продуктів з відкритим вихідним кодом, імплементованих із застосуванням об’єктно-орієнтованої парадигми програмування, процесом розробки із застосуванням систем контролю версій та значною кількістю учасників процесу розробки та тривалою історією змін коду.

Прогнозні припущення щодо розвитку об’єкта включають супроводженння об’єктно-орієнтованого програмного забезпечення шляхом модифікація моделі дельта супроводженості (DMM) шляхом розширення вимірюваних властивостей вихідного коду.

Важливо продемонструвати стабільність та ефективність вимірювання змін об’єктно-орієнтованого програмного забезпечення шляхом порівняльного аналізу внесених змін вихідного коду, що дає можливість проведення вимірювань супроводженості в процесах з методологічними підходами безперервної доставки та неперервної інтеграції. При цьому інтерпретація результатів оцінювання дає можливість проведення причинно-наслідкового зв’язку та усунення недоліків.

2 МЕТРИКИ ДЛЯ ВИМІРЮВАННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Основні метрики для вимірювання якості програмного забезпечення

Кількість рядків коду (LOC) є найдавнішою, самою простою та найбільш використовуваною метрикою призначеною для вимірювання розміру програми [23]. Метрика відображає кількість рядків коду. Основними перевагами є простота та легкість імплементації. За допомогою зазначеної метрики можливо визначити функції, класи, або методи класів занадто великого розміру, але загалом метрика не є дуже показовою оскільки кількість рядків коду залежить від складності алгоритму, мови програмування, конвенційних стандартів кодування. Наприклад, з метою покращення сприйняття коду, використання умов може вимагати як мінімум 3 строки, натомість використання більш складного для сприйняття тернарного оператора в умовах вимагає лише однієї строки.

Також метрика має свої модифікації:

SLOC - кількість рядків коду без врахування коментарів та строк, що не мають символів

LLOC - кількість логічних рядків коду, наприклад, в разі розміщення кількох умов в одному рядку обраховується, кількість строк буде нараховано відповідно до кількості умов.

Цикломатична складність Томаса Мак Кабе (CC) базується на теорії графів та використовується для обчислення складності [5]. Зазначена метрика обчислюється на основі графу, що відображає цикл роботи програми. та обчислюється наступним чином:

$CC = e - n + 2p$, де

n - is the number of vertices

e - is the number of edges

p - is the number of connected components

Також було запропоновано більш спрощену версію зазначеної метрики [15]. Таким чином складність визнається кількістю умов плюс 1.

Чим вище показник складності коду, тим складнішим є його когнітивне сприйняття та підтримка. Крім того, що дослідженнями встановлено кореляцію між складністю програм та кількістю помилок [16], складний код потребує більше тестів, для врахування всіх можливих шляхів. [could be a fibonacci example p.7]

Цикломатична складність є дуже розповсюдженим показником для вимірювання складності програмного коду, і використовується для вимірювання складності функцій, методів, класів та модулів програм.

Міри складності Холстеда виникли як альтернатива використанню кількості рядків коду для вимірювання складності коду. Основною метою зазначеної методології є визначення вимірювальних властивостей програмного продукту та відносин між ними. Обчислення відбувається на підставі наступної нотації:

n_1 - кількість унікальних операторів;

n_2 - кількість унікальних операндів;

N_1 - загальна кількість операторів;

N_2 - загальна кількість операндів.

Використовуючи зазначену нотацію виконується обчислення наступних показників:

Словник програми - кількість унікальних операторів та операндів:

$$n = n_1 + n_2$$

Тривалість програми - загальна кількість операторів та операндів:

$$N = N_1 + N_2$$

Обсяг програми, призначений для вимірювання розміру програми:

$$V = N \times log_2(n)$$

Рівень складності, відображає складність супроводженості програми:

$$D = \frac{n_1}{n_2} \times \frac{N_1}{N_2}$$

Обсяг програмування, показник зусиль необхідних для імплементації алгоритму мовою програмування:

$$E = V \times D$$

Рівень програми, визначає рівень до програма є зрозумілою:

$$L = \frac{1}{D}$$

Час імплементації, показник що відображає час в секундах, що необхідний для імплементації алгоритму мовою програмування:

$$T = \frac{E}{18}$$

Зазначені метрики використовуються для вимірювання якості, супроводженості програмного забезпечення, однак до її недоліків відноситься не врахування структурних особливостей програмного коду, взаємодії між одиницями програми [17].

Запропонований в 1994 році Чідамбером і Кемерером, набір метрик, також відомий як C&K метрики, є першими, що враховують особливості об'єктно-орієнтованої парадигми програмування: наслідування, поліморфізм, пов'язаність та залежність класів. Ці метрики досі зберігають популярність, мають широке застосування, а також використовуються як підґрунтя для створення нових метрик [8].

Залежність між класами (CBO) - кількість не успадкованих класів пов'язаних із даним класом. Клас є пов'язаним із іншим класом, якщо метод одного класу використовується методом іншого класу, або викликає змінну іншого класу. Система з низьким значенням має високу модульність та повторну використовність.

Відповідь для класу (RFC) кількість методів класу та кількість всіх методів що викликаються методами цього класу. Високе значення може бути ознакою класу високої складності, що може свідчити про складність когнитивного сприйняття та тестування.

Відсутність пов'язаності в методах (LCOM) - кількість пар методів класу, що не мають спільної змінної цього класу, не враховуючи кылькысть пар, що мають спільну змінну.Чим більше зазначений показник, тим складніше підтримувати клас, з ймовірністю, що клас має бути розділений на підкласи або кілька класів.

Зважені методи на клас (WMC) - сумму показників складності всіх методів класу. Класи з високим показником складніше підтримувати та мають значний вплив на підкласи. Обрахунок показнику складності не визначений, для забезпечення найбільш широкого застосування цієї метрики.

Глибина дерева успадкування (DIT) - кількість класів які успадковані даним класом. Показник класу кореневого класу, що не є спадкоємцем жодного класу дорівнює “0”. Чим глибше дерево спадкування, тим вище показник, і відповідно більшість успадкованих (підкласів) класів, що знижує можливість підтримуваності даного класу.

Кількість спадкоємців класу (NOC) - кількість прямих спадкоємців (підкласів) класу. Враховуючи, що наслідування є формою повторного застосування, наявність великої кількості спадкоємців класу може свідчити про високі показники повторного застосування. Однак, висока кількість спадкоємців, може спровокувати багато змін при змінах в спадкодавці (суперкласі), що, в свою чергу, негативно впливає на можливості підтримуваності.

Вказані метрики отримали багато критики через недосконаліть LCOM показника та інших, однак є найбільш поширеними та валідованими на практиці в різних мовах програмування, чим довели свою ефективність та практичність.

LCOM метрика також відома як LCOM1 і має свої модифікації: LCOM2, LCOM3, LCOM4 [18-20, 21, 43-45].

П’ять нових метрик також були запропоновані Лі та Генрі [22]. З’єднаність через передачу повідомлень (MPC) є сумою запитів, що відправляються класом. Таким чином високий показник демонструє високу з’єднаність класів, що в свою чергу визначає високу складність програмного забезпечення.

З’єднаність через абстракцію даних (DAC) є сумою атрибутів класу які є іншими класами. Чим більше атрибутів типів яких є інші класи, тим складніше зв’язаність між даним класом та іншими.

Кількість методів (NOM) - кількість методів класу. Складність класу підвищується із збільшенням кількості методів класу.

Розмір процедур або функцій (SIZEl) є кількістю двокрапок в класі. Ця метрика була запропонована як альтернатива LOC для вимірювання розміру об'єктно орієнтованих програм.

Розмір властивостей визначених класом (SIZE2) це сума кількості атрибутів та методів класу, що використовується для вимірювання розміру класу.

Зазначені метрики не є складними в обчислені, і більшість з них можуть бути визначені на етапі проектування за допомогою UML діаграм. Однак Розмір властивостей визначених класом (SIZE2) здебільшого залежить від мови імплементації.

В 1998 році Лі було запропоновано 6 додаткових метрик [9]:

Кількість спадкодавчих скласів (NAC) є фактичною альтернативою Глибини дерева успадкування (DIT), що були запропоновані Чідамбером і Кемерером, та є показником кількості успадкованих класів;

Кількість локальних методів (NLM) визначається як як кількість локальних методів визначених в класі та доступні поза межами класу. Незважаючи на інше теоретичне підґрунтя, метрика призначена для вимірювання тієї ж властивості що і Зважені методи на клас (WMC) Чідамбера і Кемерера. Показник визначає розмір локального інтерфейсу класу який використовується іншими класами, що здійснює вплив на класи які його спадкують та безпосередньо впливає на обсяг зусилль з дизайну класу, імплементації, перевикористані класу та тестування;

Складність методів класу (CMC) визначається як сума внутрішньої структурної складності всіх локальних методів класу;

Кількість спадкуючих класів (NDC) є альтернативою Кількість спадкоємців класу (NOC) Чідамбера і Кемерера, та визначається як загальна кількість спадкуючих класів (підкласів). Відповідно до теоретичного підґрунтя, метрика призначена для вимірювання обсягу в п л и в у класу на підкласи внаслідок наслідування;

З’єднаність через абстрактний тип даних (CTA) визначається як загальна кількість класів, що використовуються в якості абстрактних типів даних в декларації атрибутів класів. Два класи є з’єднаними, якщо один клас, використовує інший в

якості абстрактного типу даних. Метрика призначена для для визначення обсягу сервісів інших класів, для забезпечення класу надання сервісу іншим класам.

З'єднаність через відправку повідомлення (CTM) є кількість різних повідомлень. що відправляються класом іншим класам, за виключенням повідомлень, що відправляються локальним об'єктам створеним локальними методами класу. два класи вважаються з'єднаними якщ о один клас надсилає повідомлення об'єкту іншого класу який не успадковує або не використовує як в якості абстрактного типу даних. Метрика призначена для визначення кількості методів інших класів, що необхідні класу для функціонування.

2.2 Метрики об'єктно-орієнтованого дизайну

З метою підвищення якості об'єктно-орієнтованого програмного забезпечення в 1994 році Фернандо Бріто і Абреу було запропоновано набір метрик об'єктно-орієнтованого дизайну загальновідомого як набір метрик MOOD. Набір включає наступні метрики:

Коефіцієнт успадкування методів (MIF) є співвідношенням наслідуваних методів до загальної кількості методів в класах. Клас, що наслідує багато методів збільшує показник. Підклас, що заміщує методи суперкласу, та визначає власні методи, зменьшує показник метрики;

Коефіцієнт успадкування атрибутів (AIF) є співвідношенням наслідуваних атрибутів до загальної кількості атрибутів в класах;

Коефіцієнт залежності (COF) є співвідношенням існуючих залежностей між класами до максимальної кількості можливих залежностей. Залежність класу визначається в тому випадку, якщо він викликає метод або здійснює доступ до атрибуту іншого класу. При цьому успадкування та опосередкована залежність не врховуються. Залежність класів підвищує складність, зменьшує інкапсуляцію та потенційно повторне використання, що впливає на когнитивне сприйняття та підтримуваність;

Коефіцієнт поліморфізму (PF) вимірює ступінь заміщення методів в дереві успадкування класів. Він дорівнює кількості фактичних заміщення методів, поділених на максимально можливу кількість заміщення методів. Якщо всі методи всіх класів заміщенні успадкованими класами, цей показник буде дорівнювати 100%. Показник, що дорівнює 0% може свідчити про відсутність спадковості класів або поліморфізму;

Коефіцієнт прихованих методів (MHF) є значенням, що відображає кількість прихованих методів всіх класів в системі. Прихованими є всі методи, що стосуються імплементації функціональності самого класу або приватні методи. Видимим методами є методи інтерфейсу класу. Якщо всі методи є прихованими, показник дорівнює 100%, і навпаки, якщо всі методи відносяться до інтерфейсу класу, показник дорівню 0%. Низький показник демонструє неефективність абстрактної імплементації, що підвищує вірогідність помилок;

Коефіцієнт прихованих атрибутів (AHF) є значенням, що відображає кількість прихованих атрибутів всіх класів в системі. Прихованими є всі атрибути, що стосуються імплементації функціональності самого класу або приватні атрибути. Видимим методами є методи інтерфейсу класу. Якщо всі атрибути є прихованими, показник дорівнює 100%. В ідеальній імплементації всі атрибути мають бути прихованими. Чим більше клас, наслідує неприхованих атрибутів тим нижче буде показник, що може свідчити про проблеми в імплементації наслідування.

Ефективність використання наведених метрик були підтверджені дослідженнями Ф. Абреу та Р. Харрісона, крім того також було встановлено похідність з метриками запропонованими С. Чідамбером і С. Кемерером, однак останні досліджують показники не на рівні програми в цілому, а на рівні класів, а тому є більш ефективними з точку зору щоденної інженерної практики [8, 30-32].

3 МЕТРИЧНИЙ АНАЛІЗ СУПРОВОДЖЕНОСТІ

В спільній статті представленій на на Міжнародній конференції з пітримуваності програмного забезпечення в 1992 році, Оман та Хагемейстер зібрали та проаналізували 60 метрик призначених для вимірювання підтримуваності програмного забезпечення [24]. Оскільки більшість зазначених метрик були занадто складні для обчислення, в основному через те, що потребували історичних або суб'єктивних даних, в 1994 році було запропоновано Індекс підтримуваності (MI) [25]. Для того щоб знайти просту та практичну модель, що могла була б бути застосована до різних типів програмного забезпечення було проведено серію з 50 статистичних регресійних тестів. З 3 ймовірних моделей, що були протестовані, було обрано модель що складається з Обсягу Холстеда, Цикломатичної складності МакКаби, Кількості рядків коду та Кількості рядків коментарів. Оригінальна формула Індексу підтримуваності (MI) представлена настпуним чином:д

$$MI = 171 - 5{,}2 \ln(HV) - 0{,}23(CC) - 16{,}2 \ln(LOC) + 0{,}99(CMT),$$ де

HV - середній показник Обсягу Холстеда модуля;

CC - середній показник Цикломатичної складності МакКаби модуля;

LOC - середній показник Кількості рядків коду модуля;

CMT - середній показник Кількості рядків коментарів модуля.

Незважаючи на популярність Індексу підтримуваності, цей показник вважається дуже суперечливим. Багато критики спричинено, рызними факторами, якими э недостатнэ обгрунтування формули, обрахунків середных значень, та неоднозначність зв'язку між значенням загального результату та показниками конкретними метрик. [26].

Зазначений показник також має кілька популярних модифікацій. Формула Інституту програмної інженерії представлена наступним чином [27]:

$$MI = 171 - 5{,}2 \times \ln(HV) - 0{,}23 \times CC - 16{,}2 \times \ln(LOC) - 50 \times sin\sqrt{2{,}4 \times CMT}$$

3.1 Модель супроводженості (SIG-MM)

Модель супроводженості, відому як модель SIG-MM, було запропоновано консалтинговою організацією Software Improvement Group в 2007 році [34], як альтернативу Індекс супроводженості Омана та Хагемейстера. Модель є незалежною від мови програмування та архітектури програмного забезпечення, має прості для розуміння та пояснення показники і базується на визначені зв'язків характеристик якості системного рівня визначених стандартом ISO 9126-1 [12] з характеристиками властивостей вихідного коду та їх метричними показниками. Значною мірою цей підхід ґрунтується на необхідності визначення причинно-наслідкових зв'язків між властивостями вихідного коду та супроводженостю, оскільки останній є складним та багатокомпонентним атрибутом якості.

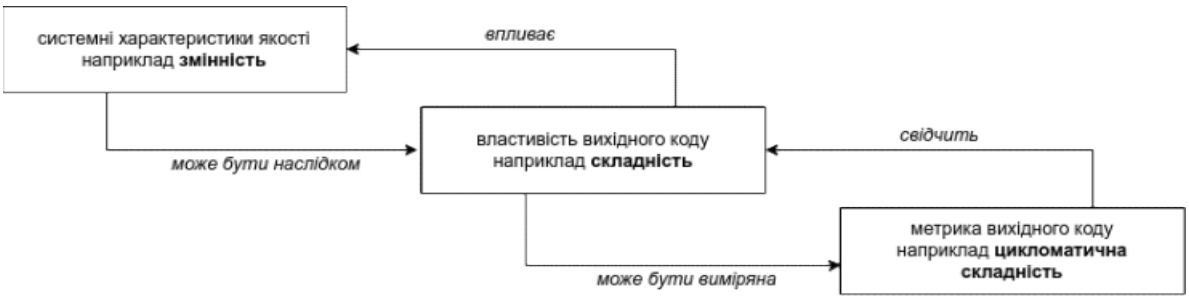


Рис. 3.1 − Зв'язок характеристик якості системного рівня з характеристиками властивостей вихідного коду та їх метричними показниками відповідно до моделі супроводженості SIG

Для ранжування результатів оцінювання окремої властивості коду використовується проста символьна шкала ++ / + / o / - / --.

Розмір програми є одним із простих та безпосередніх показників супроводженості, оскільки чим більше розмір тим більше зусиль вимагається для когнитивного сприйняття, внесення змін, тестування.

Показник встановлюється для кожної мови окремо на підставі досліджень [35], що визначають зв'язок між середньою кількістю рядків коду (LOC) в окремій мові програмування на одну функціональну точку, та

кількістю функціональних точок, що можуть бути вироблені однією людиною впродовж одного місяця.

Для цілей моделі, що розглядається, розмір програми визначається людино-роками необхідними для створення програми, з наступним ранжуванням:

Ранг	Кількість людино-років
++	0 − 8
+	8 − 30
o	30 − 80
-	80 − 160
--	> 160

Таким чином програмний продукт, що потребує 160 людино-років, вважається занадто великим, і для систем імплементованих на Java дорівнює 1.3 мільйону строк коду або 2.6 для мови COBOL.

ТабМатриця зв’язків підхарактеристик якості ISO 9126-1 з характеристиками властивостей вихідного коду відповідно до моделі супроводженості SIG

Таблиця 3. Матриця зв’язків підхарактеристик якості ISO 9126-1 з характеристиками властивостей вихідного коду відповідно до моделі супроводженості SIG

Підхаристики супроводженості ISO 9126-1	**Властивості вихідного коду**				
	Розмір програми	Складність на одиницю програми	Дублювання	Розмір одиниці програми	Модульне тестування
Аналізовність	**X**		**X**	**X**	**X**
Змінність		**X**	**X**		
Стабільність					**X**

Тестовність		**X**		**X**	**X**

Сладність на одиницю програми є властивістю вихідного коду та визначається як ступінь внутрішньої складності одиниць вихідного коду, з яких він складається.

Складність категоризуєьтся наступним чином:

Цикломатична складність	Визначення ризику
1-10	проста
11-20	ускладнена, помірний риск
21-50	складна, високий ризик
> 50	відсутність тестованості, дуже високий ризик

З подальшою агрегацією складності на одиницю програми для визначення співвідношення рядків коду кожного рівня ризику у відсотках. Тобто, якщо програма складається з 20000 рядків коду, і при цьому сума рядків коду одиниць програми з високим ризиком складності складає 1000 рядків коду, то сукупне значення для категорії високого ризику буде 5%.

Обрахунок відносних обсягів із підсумковим розподілення відносно різних рівнів ризику використовується для ранжування системи в цілому:

	Максимальна відносна кількість рядків коду (LOC)		
Ранг	Помірний ризик	Високий ризик	Дуже високий ризик
++	25%	0%	0%
+	30%	5%	0%
o	40%	10%	0%
-	50%	15%	5%
--	-	-	-

Таким чином, для прикладу, якщо ранг програми визначений як “+”, то кількість рядків коду з високим ризиком не перевищує 5%, з високим ризиком 15% та 50% рядків коду знаходяться в межах помірного ризику.

Дублювання (або клонування коду) знижує когнітивне сприйняття програми, можливість внесення змін та невмотивовано збільшує розмір програми. Та визначається як повтореня блоку коду більше 6 строк, при цьому для визначення повторення не враховуються пробіли на початку строк.

Дюблювання програми ранжується за наступними параметрами:

Ранг	Дублювання
++	0 - 3%
+	3 - 5%
o	5 - 10%
-	10 - 20%
--	20 - 100%

Розмір одиниці програми є важливим показником, оскільки великі за розміром одиниці програми потребують більше витрат на підтримку, також, цей показник додатково опосередковано відображає можливу складність. Показник визначається як кількість рядків коду (LOC) з подальшою категоризацією за розміром і ранжуванням подібним до визначення складності на одиницю програми.

Модульне тестування обчислюється відносним показником покриття програми тестами одиниць програми. Зазначена практика не є статичним аналізом і відноситься до динамічного аналізу коду. Вказаний показник в подальшому використовується в наступній схемі ранжування:

Ранг	Рівень покриття модульними тестами
++	95-100%
+	80-95%

o	60-80%
-	20-60%
--	0-20%

Обчислення загальної оцінки системи здійснюється середнього зважування показників кожної властивості вихідного коду (Таб.4).

Наприклад: розмір програми оцінюється як невеликий “++”, з дуже високою складністю на одиницю програми “--”, високим дублюванням та розміром одиниці програми “-” і помірним тестуванням. Відповідно аналізованість такого програмного забезпечення як і стабільність є середніми, при цьому змінність та тестованість низькою, що усереднюється низькою “-” оцінкою підтримуваності. Однак, зазначені результати є більш ефективними у визначенні причинно-наслідкового зв'язку. Таким чином для підвищення супроводженості необхідно провести рефакторинг направлений на одиниці програми дуже високої складності з метою її зниження, та зменшення розміру самих одиниць програми як і прибирання дублювань.

Таблиця 4. Загальна оцінка системи здійснюється середнього зважування показників кожної властивості вихідного коду

Підхарактеристики супроводженості ISO 9126-1	**Властивості вихідного коду**					
	Розмір програми	Складність на одиницю програми	Дублювання	Розмір одиниці програми	Модульне тестування	
	++	--	-	-	o	
Аналізовність	x		x	x	x	o
Змінність		x	x			-
Стабільність					x	o
Тестовність		x		x	x	-

3.2 Модель дельта супроводженості (DMM)

Відомі дослідники Марко ді Біасе, Аюші Растогі, Магіель Брантінк, Арі ван Деурсен в 2019 році запропонували модель дельта супроводженості [36]. Модель базується на Моделі супроводженості (SIG-MM) запропонованою Software Improvement Group, однак на відміну від останньої вона призначена для порівняння та аналізу часткових змін вихідного коду, а не програми в цілому. Модель інтегрується з системами контролю версій, що дозволяє інтеграцію з інструментами DevOps для застосування при аналізі постійних перевірок вихідного коду. Ранжування ризиків підставі порогових значень Моделі супроводженості (SIG-MM) [34]. Оцінка властивостей вихідного коду та категоризація ризиків відбувається за наступними параметрами (Табл.5).

Таблиця 5. Опис властивостей системи та їх зв'язок із пороговими значеннями призначеними для визначення ризику вихідного коду

Властивість вихідного коду	**Опис**	**Критерій низького ризику**
Дублювання	Повторенням більше ніж 6 строк вихідного коду на текстовому рівні, із урахуванням пробілів	Рядки коду, що не мають дублювань
Розмір одиниці програми	Кількість рядків коду (LOC) одиниці програми, за виключенням коментарів, або строк складених лише з пробілів	Одиниці програми, що містять до 15 рядків коду (LOC)
Складність одиниці програми	Цикломатична складність Мак Кабе (CC) [5] одиниці програми (підпрограми, функції, методу)	Одиниці програми зі складністю не більше 5
Розмір інтерфейсу одиниці програми	Кількість формальних параметрів інтерфейсу	Одиниці що мають не більше 2 параметрів
Залежність модуля	Кількість вхідних залежностей модуля. Модулем є групуючий елемент одиниць програми, файл.	Модуль що має не більше 10 вхідних залежностей

Дельта профіль ризику (RDP) є показником різниці між показниками LOC внесених змін у файл та попереднього стану файлу за певною

категорією ризику властивості коду. LOC показники визначаються відповідно до властивості коду за методологією SIG-MM [34].

$$RPD(f^1, f, cp, r) = LOC(f, cp, r) - LOC(f^1, cp, r), \text{ де}$$

LOC - кількість рядків коду

r - категорія ризику

cp - властивість коду

f - файл версії із змінами

f^1 - файл попередньої версії

Дельта профілю низького ризику (LRPD) є сумою показників ризику всіх змін у файлах, що мають низький показник ризику, тобто змін з високим рівнем підтримки властивостей коду, що зумовлюють супроводженість і визначається формулами:

$$HRPD(cp) = CRPDD(cp, \textit{низький}) + \sum_{h \in (\textit{середній, високий, дуже високий})} CRDPI(cp, h)$$

$$CRPDD(cp, r) = \sum_{\{f^1, f\} \in D} RPDD(f^1, f, cp, r)$$

$$RPDD(f^1, f, cp, r) = |\, min(0, RPD(f^1, f, cp, r))\,|$$

Дельта профіля високого ризику (HRPD) є сумою показників ризику всіх змін, що знижують супроводженість:

$$LRPD(cp) = CRPDI(cp, \textit{низький}) + \sum_{h \in (\textit{середній, високий, дуже високий})} CRDPD(cp, h)$$

$$CRPDI(cp, r) = \sum_{\{f^1, f\} \in D} RPDI(f^1, f, cp, r)$$

$$RPDI(f^1, f, cp, r) = max(0, RPD(f^1, f, cp, r))$$

Дельта показник властивості коду (DS) є співвідношенням дельта показника низького ризику (LRPD) відносно суми показників всіх категорій ризиків:

$$DS(cp) = \frac{LRDP(cp)}{LRDP(cp) + HRPD(cp)}$$

Показник дельта супроводженості (DDMS) є середнім значенням дельта показників для кожної властивості вихідного коду в діапазоні від 0, що є найменшим показником супроводженості, до 1, що є найвищим показником супроводженості:

CP = {Дублювання, Розмір одиниці програми, Складність одиниці програми, Взаємодія одиниці програми, Залежність модуля}

$$\text{DDMS} = \frac{\sum_{cp \in CP} DS(cp)}{|CP|}$$

Модель дельта супроводженості на відміну від інших розглянутих моделей потребує значно меньше даних, оскільки аналізу піддається не вся база вихідного коду програмного продукту, а лише файли змін, що дає можливість отримати метричні показники щодо невеликих змін коду, і забезпечує можливість інтеграції з системами контроля версій та застосування в процесах з методологічними підходами безперервної доставки та неперервної інтеграції.

3.3 Модефікована модель дельта супроводженості об'єктно-орієнтованого програмного забезпечення

Супроводженість є складною концепцією, неодноразово розглянутою дослідженнями, що пропонують різні формулювання та підходи до вимірювання. Однак, більшість досліджених підходів, в тій чи іншій мірі мають:

1) неоднозначність термінології та визначень критеріїв якості;

2) абстрактність, відсутність визначень формулювань та способів вимірювання;

3) складність або неможливість інтерпретації, або проведення причинно-наслідкового аналізу результатів вимірювання.

Модель супроводженості (SIG-MM) не має означених недоліків, однак базується на вимірюванні всього вихідного коду програмного продукту, що

обумовлює слабка репрезентативність результатів вимірювання при незначних змінах вихідного коду.

Прикладом цього є запис #402331 в системі реєстрації дефектів програмного продукту Mozilla Rhino (https://bugzilla.mozilla.org/showbug.cgi?id=402331). Зазначений дефект було виправлено комітом #262602 (https://github.com/mozilla/rhino/commit/262602), який в діапазоні вимірювань від -5 до 5. Модель супроводженості (SIG-MM) має рейтинг −0.007. Зазначений результат не демонструє будь-яких істотних змін в супроводженості, що не відповідає дійсності, оскільки внесені змінами 200 рядків коду істотно негативно впливають на супроводженість. Однак відносно розміру всіх строк коду Mozilla Rhino, в порівнянні до якого зміни були валідовані, показник отримав не репрезентативний результат [36]. Модель дельта супроводженості (DMM) не містить наведених недоліків, однак не враховує специфіки об’єктно-орієнтованої парадігми програмування.

Враховуючи вищевикладене, пропонується модифікація моделі дельта супроводженості (DMM) шляхом розширення вимірюваних властивостей вихідного коду, визначення їх взаємозв’язку з підхарактеристиками супроводженості, що визначені ISO/IEC 25010:2016, визначення способів вимірювання та порогових значень [10, 11].

В подальшому базова модель позначається DMM, результуючий показник DMMS. Посилання на запропоновану модель позначаються як DMM+, показник відповідно DMMS+.

Обчислення відбувається з урахуванням в порядку встановленому для Моделі дельта супроводженості (DMM) [46] з наступними змінами визначень:

RC = {низький, високий};

CP = {Пов'язаність класу, Складність класу, Складність методу, Кількість методів, Розмір методу, Кількість параметрів, Дублювання, Залежність модуля}.

Таблиця 6. Матриця зв’язків підхарактеристик якості ISO 25010 з характеристиками властивостей вихідного коду

Властивості вихідного коду	**Підхарактеристики супроводженості ISO 25010**				
	Модульність	Повторна використовність	Аналізовність	Модифіковність	Тестовність
Пов'язаність класу	**X**	**X**		**X**	**X**
Складність класу			**X**		
Складність класу			**X**	**X**	**X**
Кількість методіва		**X**			**X**
Розмір методу			**X**		
Кількість параметрів		**X**			**X**
Дублювання	**X**		**X**	**X**	
Залежність модуля		**X**		**X**	

Дослідженням 111 систем програмного забезпечення проведеним Філо, Тарсіо Г.С. та М. Бігонья [42] запропоновано визначення порогових значень метрик об’єктно-орієнтованого програмного забезпечення (Табл.7).

Практичну валідацію запропонованої моделі проведено шляхом аналізу програмних продуктів з відкритим вихідним кодом, імплементованих із застосуванням об’єктно-орієнтованої парадигми програмування, процесом розробки із застосуванням систем контролю версій та значною кількістю учасників процесу розробки та тривалою історією змін коду.

Таблиця 7. Таблиця метрик програмного забезпечення Філо,

Тарсіо Г.С. та М. Бігонья

	Рівень		
Метрика	**Кращий**	**Середній**	**Поганий**
WMC	m ≤ 11	11 < m ≤ 34	m > 34
NOC	m ≤ 11	11 < m ≤ 28	m > 28
NOM	m ≤ 6	6 < m ≤ 14	m > 14
MLOC	m ≤ 10	10 < m ≤ 30	m > 30
PAR	m ≤ 2	2 <m ≤ 4	m> 4
VG	m ≤ 2	2 <m ≤ 4	m> 4

Оскільки аналіз вихідного коду програмних продуктів потребує дослідження абстрактного синтаксичного дерева (АСД), тому з метою спрощення імплементації застосунку, призначеного для аналізу, всі програмні продукти обрані з вимого імплементації об’єктно-орієнтованої частини спільною мовою програмування.

Таблиця 8. Опис властивостей вихідного коду та їх зв’язок із пороговими значеннями призначеними для визначення ризику вихідного коду

Властивість вихідного коду	**Коротка назва**	**Опис**	**Критерій низького ризику**
Пов'язаність класу	LCOM	Відсутність пов’язаності в методах LCOM4	LCOM = 1
Складність класу	WMC	Зважені методи на клас WMC, з обрахування цикломатичної складності CC	WMC ≤ 11
Складність методу	VG	Цикломатична складність методу	VG ≤ 2
Кількість методів	NOM	Кількість методів одного класу	NOM ≤ 6
Розмір методу	MLOC	Розмір методу в рядках коду (SLOC)	MLOC ≤ 10
Кількість параметрів	PAR	Кількість формальних параметрів інтерфейсу метода	PAR ≤ 2
Дублювання	DCL	Повторенням більше ніж 6 строк вихідного коду на текстовому рівні, із урахуванням пробілів, без урахування строк без символів.	DCL = 0

Залежність модуля	MC	Кількість вхідних залежностей модуля. Модулем є групуючий елемент одиниць програми, файл.	MC ≤ 10

Для порівняльного аналізу було обрано 6 програмних продуктів різною функціональної призначенності, з відкритим вихідним кодом, з імплементацією об’єктно-орієнтованої частини мовою програмування Python.

Таблиця 9. Порівняльний аналіз програмних продуктів

Репозиторій	Гілка	Дата першого комміту	Кількість комітів	Кількість комітів до Python файлів	Кількість файлів	Кількість модулів Python	Кількість класів (DSC)	Кількість методів
sentry	master	12.05.2008	43102	20807	10016	2256	3141	8247
zulip	main	28.08.2012	47222	17956	5712	1102	1050	1264
saleor	main	12.02.2013	19611	10999	2260	1428	2274	2484
django	main	13.07.2005	30744	16276	6625	854	1753	6844
odoo	15.0	07.12.2006	148369	87689	29301	3351	2660	13956
tensorflow	master	06.11.2015	129359	36224	26482	1240	1963	11991

Sentry є сервісом призначеним для моніторингу дефектів програмного забезпечення в режимі реального часу.

Zulip є застосунком для синхронного та асинхронного спілкування команд, що розробляється розподіленою командою розробників.

Saleor є комерційною платформою з відкритим вихідним кодом, призначеним для роботи із великою кількістю даних та навантаженнями.

Django є фреймворком призначеним для створення серверних застосунків та високорівневим доступом до баз данних.

Tensorflow є комплексною платформою для машинного навчання, з розгалуженою екосистемою застосунків, і великою спільногото дослідників.

Odoo є комплектом застосунків для автоматизації всіх основних аспектів комерційної діяльності.

Область аналізу охоплює лише зміни в файлах (модулях), що містять інструкції на мові програмування Python, та мають розширення “*.py”, при цьому не враховуються зміни до файлів із інструкціями для модульного тестування.

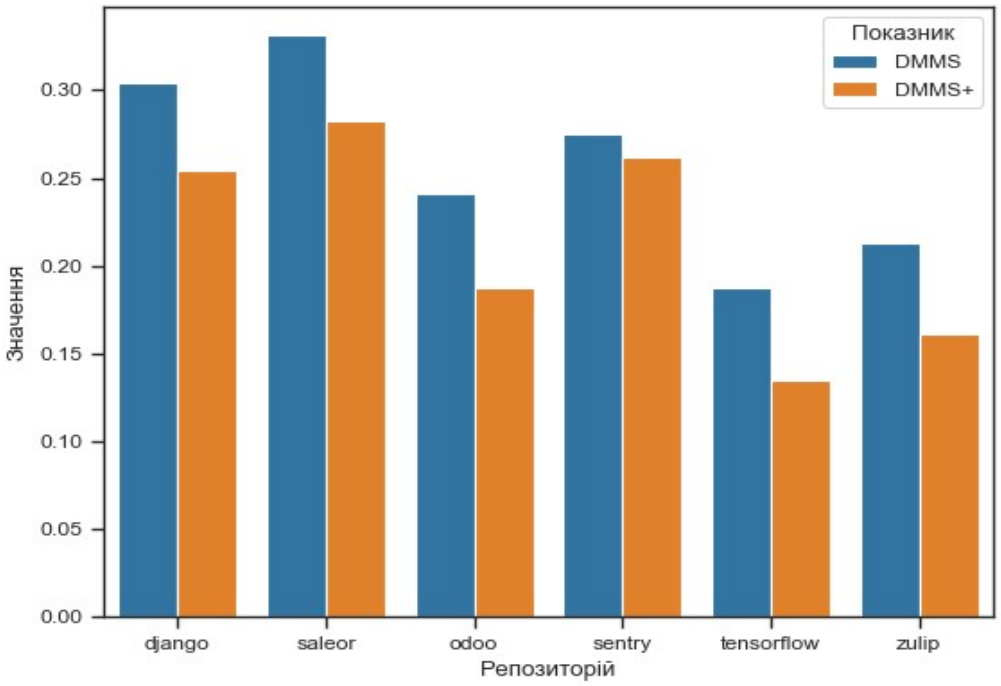
З метою проведення дослідження та вимірювань, було імплементовано програму, з підтримкою інтерфейсу командної строки та забезпечення одночасного вимірювання показників моделі DMM та DMM+. Вихідний код програми розміщений в репозиторії https://gitlab.com/a10r/dmm, а також в Додаток 1.

Відповідно до результатів проведеного аналізу значний показник коефіцієнта кореляції Пірсона в значенні від 0.77 до 0.86 демонструє сильну позитивну кореляцію між значеннями DMMS та DMMS+. Таким чином, показники вимірювання DMMS+ разом із DMMS відображають взаємозв’язок із змінами вихідного коду, що впливають на супроводженість. Значення показників DMMS та валідація були підтверджені [36] в ході емпіричних досліджень (Додаток 2). Кореляція показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії:

1) Tensorflow - r = 0.86,
2) Sentry - r = 0.8,
3) Django - r = 0.82,
4) Odoo - r = 0.84,
5) Saleor - r = 0.77,
6) Zulip - r = 0.77.

Незважаючи на позитивну кореляцію, показники демонструють флуктуацію за показником абсолютної середньої різниці. Важлива показати зміни в показнику абсолютної середньої різниці між показниками DMMS та DMMS+ за результатами аналізу 1000 внесених змін до кожного з репозиторіїв (Таблиця 10).

Таблиця 10. Ілюстрація змін в показнику абсолютної середньої різниці між показниками DMMS та DMMS+



Дослідження змін внесених комітом 2798c937deb6625a4e6a36e70d4d60ce5faac954, демонструє реакцію показника DMMS+ на зміни в аспектах об’єктно-орієнтованого дизайну, що знижують супроводженість, але не були відзначені при вимірюваннях методами DMM (Додаток 3).

Також проведено порівняння метричних даних змін внесених до файлу django/db/models/expressions.py різним комітом (Додаток 4).

На відміну від DMM, що дорівнює 0.41, DMMS+ дорівнює 0.21 оскільки внесеними змінами було збільшено розмір класу, що за цією ознакою має низьку супроводженість, додано рядки в клас, але клас при цьому перевищує порогову складність і не втратив її, внаслідок змін. Також змінами внесено додатковий метод до класу, що перевищує порогове значення кількості класів, також внесені, направлення на збільшення зміни, в метод, що не втратив внаслідок змін, ризикованості за кількістю параметрів.

ВИСНОВКИ

У результаті виконання даної дипломної роботи підвищено якість та рівень супроводженості об’єктно-орієнтованого програмного забезпечення за рахунок зменшення кількості дефектів.

1. Проаналізовано існуючі моделі та метрики вимірювання об’єктно-орієнтованого програмного забезпечення, визначені їх переваги та недоліки.

2. Модефіковано модель дельта супроводженості об’єктно-орієнтованого програмного забезпечення, шляхом розширення вимірюваних властивостей вихідного коду, визначення їх взаємозв’язку з підхарактеристиками супроводженості, що визначені ISO/IEC 25010:2016. Визначено способи вимірювання та порогові значення.

3. Проведено практичну валідацію запропонованої моделі шляхом аналізу програмних продуктів з відкритим вихідним кодом, імплементованих із застосуванням об’єктно-орієнтованої парадигми програмування, процесом розробки із застосуванням систем контролю версій та значною кількістю учасників процесу розробки та тривалою історією змін коду.

4. Для порівняльного аналізу використання модефікованої моделі було обрано 6 програмних продуктів різною функціональної призначенності, з відкритим вихідним кодом, з імплементацією об’єктно-орієнтованої частини мовою програмування Python.

5. Встановлено, що відповідно до результатів проведеного аналізу значний показник коефіцієнта кореляції Пірсона в значенні від 0.77 до 0.86 демонструє сильну позитивну кореляцію між значеннями DMMS та DMMS+. Таким чином, показники вимірювання DMMS+ разом із DMMS відображають взаємозв’язок із змінами вихідного коду, що впливають на супроводженість. Значення показників DMMS та валідація були підтверджені в ході емпіричних досліджень.

6. Продемонстровано стабільність та ефективність вимірювання змін об’єктно-орієнтованого програмного забезпечення шляхом порівняльного

аналізу внесених змін вихідного коду, що дає можливість проведення вимірювань супроводженості в процесах з методологічними підходами безперервної доставки та неперервної інтеграції. При цьому інтерпретація результатів оцінювання дає можливість проведення причинно-наслідкового зв’язку та усунення недоліків.

ПЕРЕЛІК ПОСИЛАНЬ

1. C. Jones, “The Economics of Software Maintenance in the Twenty First Century,” 2006, p.4

2. The Cost of Poor Software Quality in the US: A 2020 Report: The Consortium for Information & Software Quality (CISQ) 4,16pp

3. A. J. Albrecht, "Measuring Application Development Productivity," Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California, October 14–17, IBM Corporation (1979), pp. 83–92.

4. Maurice H. Halstead, Elements of Software Science (North-Holland, Amsterdam, 1977)

5. T. J. McCabe, A complexity measure, IEE Trans. Softw. Eng. 72(2) (1976) 308-320

6. K.D. Welker, P. Oman and G.G. Atkinson, Development and application of an automated source code maintainability index, J/Softw/ Maint/ Res/ Pract 9(3) (1997) 127-159.

7. W. Stevens, G. Myers, L. Constantine, “Structured Design”, IBM Systems Journal, 13 (2), 115-139, 1974.

8. S.F. Chidamber and C.F. Kamerer, A metric suit for object-oriented design, IEEE Trans. Softw. Eng. 20(6) (1994) 476-493.

9. Li W., “Another Metric Suite for Object- Programming”, The Journal of System and Software, Vol. 44, Issue 2, December 1998, pp. 155-162.

10. ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models

11. ДСТУ ISO/IEC 25010:2016 Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів

12. ISO/IEC 9126-2001. Software engineering -- Product quality— Part 1: Quality model.

13. ДСТУ ISO/IEC 9126-1:2013 Програмна інженерія. Якість продукту. Частина 1. Модель якості.

14. Plösch, R., Gruber, H., Korner, C., & Saft, M. (2010). A Method for Continuous Code Quality Management Using Static Analysis. In 7th International Conference on the Quality of Information and Communications Technology (QUATIC) (pp. 370–375). Porto, Portugal: IEEE

15. A. Madi, O.K. Zein, and Seifedine Kadry. On the improvement of cyclomatic complexity metric. International Journal of Software Engineering and its Applications, 7:67–82, 01 2013.

16. M. Schroeder. A practical guide to object-oriented metrics. IT Professional, 1(6):30–36, 1999.

17. A. Madi, O.K. Zein, and Seifedine Kadry. On the improvement of cyclomatic complexity metric. International Journal of Software Engineering and its Applications, 7:67–82, 01 2013.

18. N. I. Chucher and J. S. Martin, Comments on a metrics suite for object oriented design, IEEE Trans. Softw. Eng. 21(3) (1995) 263-265.

19. T. Mayer and T. Hall, A critical analysis of current 00 design metrics, Softro. Qual. J. 8(2)(1999) 97–110.

20. M. Hitz and B. Montazeri, Chidamber and Kemerer's metrics suite: A measurement theory perspective, IEEE Trans. Softw. Eng. 22(4) (1996) 267–271.

21. Ramanath Subramanyam and M. Krishnan. Empirical analysis of ck metrics for object oriented design complexity: Implications for software defects. Software Engineering, IEEE Transactions on, 29:297– 310, 05 2003.

22. Wei Li and Sallie Henry. Object-oriented metrics that predict maintainability. Journal of Systems and Software, 23(2):111 – 122, 1993.

23. Alberto S. Nuñez-Varela, Héctor G. Pérez-Gonzalez, Francisco E. Martínez-Perez, and Carlos Soubervielle-Montalvo. Source code metrics: A systematic mapping study. Journal of Systems and Software, 128:164 – 197, 2017.

24. P. Oman and J. Hagemeister, “Metrics for assessing a software system’s maintainability,” in Proceedings of the Conference on Software Maintenance, pp. 337–344, Orlando, FL, USA, November 1992.

25. O. Paul and J. Hagemeister, “Construction and testing of polynomials predicting software maintainability,” Journal of Systems and Software, vol. 24, no. 3, pp. 251–266, 1994.

26. S. Counsell, X. Liu, S. Eldh et al., “Re-visiting the “Maintainability Index” metric from an object-oriented perspective,” in Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications, pp. 84–87, IEEE, Funchal, Portugal, August 2015.

27. John T. Foreman, Jon Gross, Robert Rosenstein, David Fisher, Kimberly Brune, Software Engineering Institute, CMU/SEI-97-HB-001, p. 231

28. Microsoft Visual Studio documentation, https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning

29. Abreu, Fernando Brito e and Rogério Carapuça. “Object-Oriented Software Engineering: Measuring and Controlling the Development Process.” (1994).

30. e Abreu, F. B. & Melo, W. 1996. Evaluating the impact of object-oriented design on software quality. In Software Metrics Symposium, 1996., Proceedings of the 3rd International, 90–99. IEEE.

31. Harrison, R., Counsell, S. J., & Nithi, R. V. 1998. An evaluation of the mood set of object-oriented software metrics. IEEE Transactions on Software Engineering, 24(6), 491–496.

32. Basili, V. R., Briand, L. C., & Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on software engineering, 22(10), 751–761.

33. J. Bansiya and C. Davis, ªClass Cohesion Metric For Object-Oriented Designs,º J. Object-Oriented Programming, vol. 11, no. 8, pp. 47-52, Jan. 1999.

34. I. Heitlager, T. Kuipers and J. Visser, "A Practical Model for Measuring Maintainability," 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007), 2007, pp. 30-39, doi: 10.1109/QUATIC.2007.8.

35. Software Productivity Research LCC, “Programming Languages Table,” Feb. 2006, version 2006b.

36. M. di Biase, A. Rastogi, M. Bruntink and A. van Deursen, "The Delta Maintainability Model: Measuring Maintainability of Fine-Grained Code Changes," 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), 2019, pp. 113-122, doi: 10.1109/TechDebt.2019.00030.

37. T. L. Alves, J. P. Correia, and J. Visser. Benchmark-Based Aggregation of Metrics to Ratings. In 2011 Joint Conference of the 21st Interna-tional Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement, pages 20–29, 2011.

38. Jim A. McCall, Paul K. Richards, and Gene F. Walters. Factors in Software Quality. Volume-III. Preliminary Handbook on Software Quality for an Acquisiton Manager.

39. Barry W. Boehm, John R. Brown, and Mlity Lipow. “Quantitative evaluation of software quality”. In: Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press, 1976, pp. 592–605. (Visited on 2017-01-31).

40. R. Geoff Dromey. “A model for software product quality”. In: IEEE Transactions on software Engineering 21.2 (1995), pp. 146–162.

41. Robert B. Grady. Practical software metrics for project management and process improvement. Prentice-Hall, Inc., 1992. ISBN : 0-13-720384-5.

42. Filó, Tarcísio G. S. and Mariza Bigonha. “A Catalogue of Thresholds for Object-Oriented Software Metrics.” (2015).

43. W. Li and S.M. Henry, Maintenance metrics for the object oriented paradigm. In Proceedings of 1st International Software Metrics Symposium, Baltimore, MD, 1993, pp. 52-60.

44. B. Henderson-Sellers, Software Metrics, Prentice Hall, Hemel Hempstaed, U.K., 1996.

45. M. Hitz and B. Montazeri, Measuring coupling and cohesion in object oriented systems, Proceedings of the International Symposium on Applied Corporate Computing, 1995, pp. 25-27.

46. M. di Biase, A. Rastogi, M. Bruntink, and A. van Deursen. The Delta Maintainability Model: Measuring Maintainability of Fine-Grained Code Changes Technical Report, 2019.

ДОДАТКИ

DMM metrics command-line utility:

The utility is intended for the analysis of git repositories for gathering DMM related metrics, DDM Score aggregation, and additional object-oriented metrics.

To use git and python3 need to be installed on the system.

Repositories configurations can be found/changed in `config.toml` file.

Please make sure to install dependencies with `pip`

```
$ pip install -r requirements.txt
```

To run the analysis use the `profiler.py` module with

```
$ python3 profiler.py --rev=HEAD --offset=1000 --repository=NAME
```

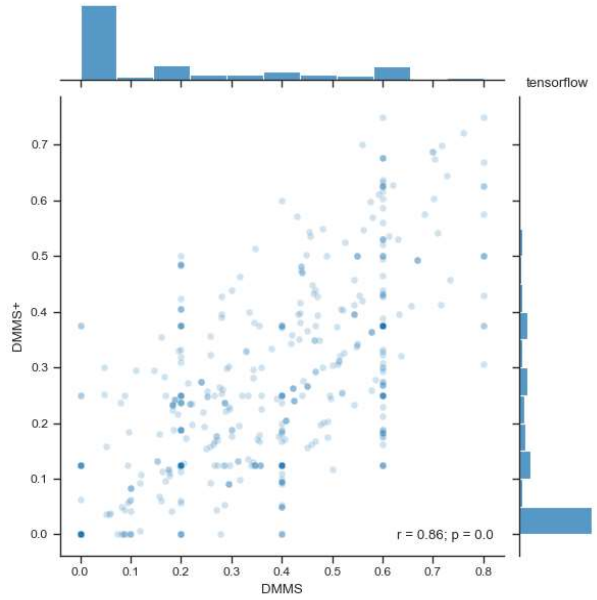
where NAME is repository name as appeared in `config.toml`.

For help please run

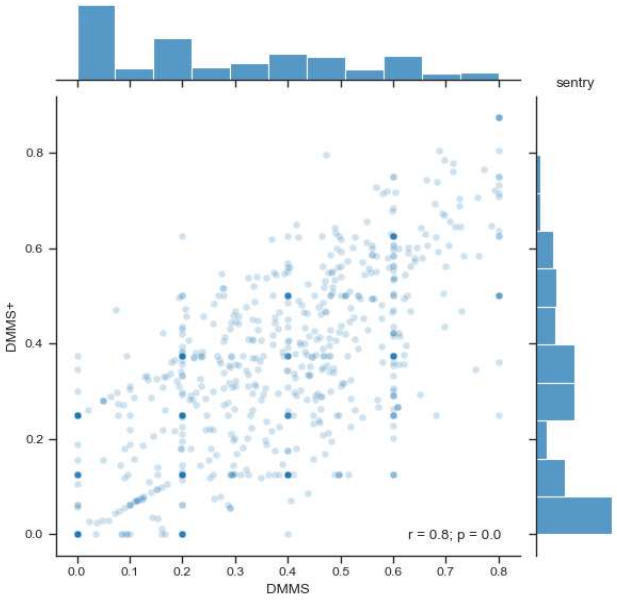
```
$ python3 profiler.py --help
```

Додаток 1

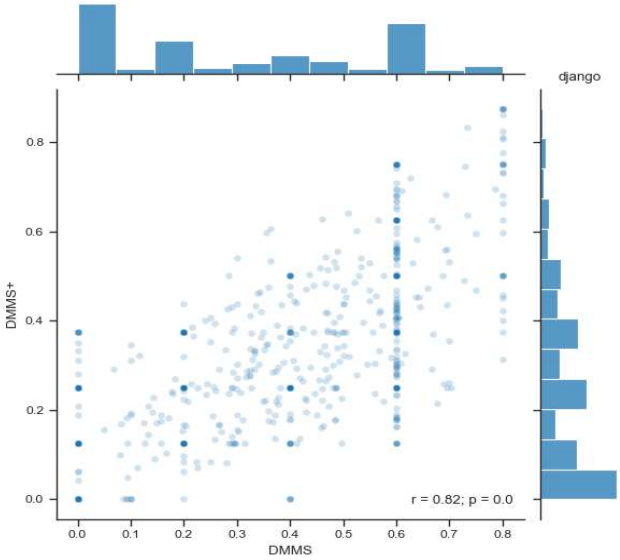
Додаток 2



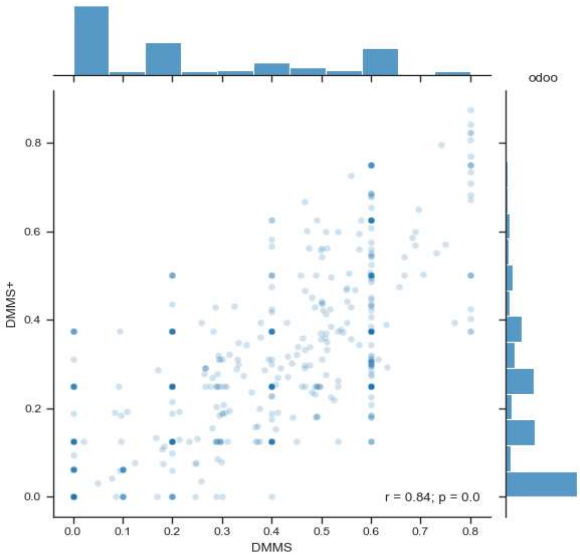
Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Tensorflow r = 0.86



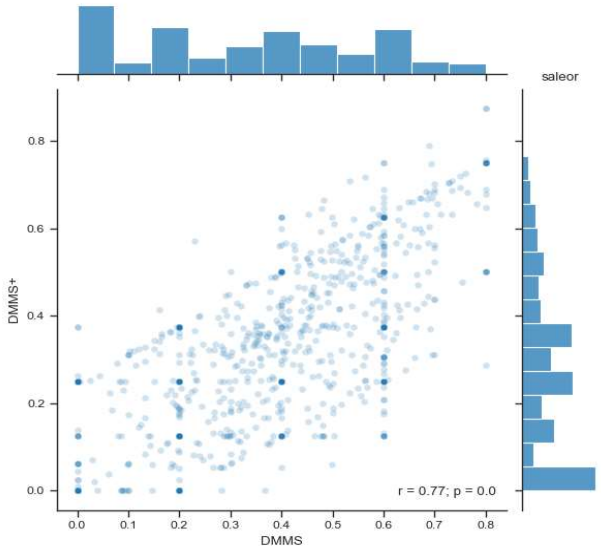
Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Sentry r = 0.8



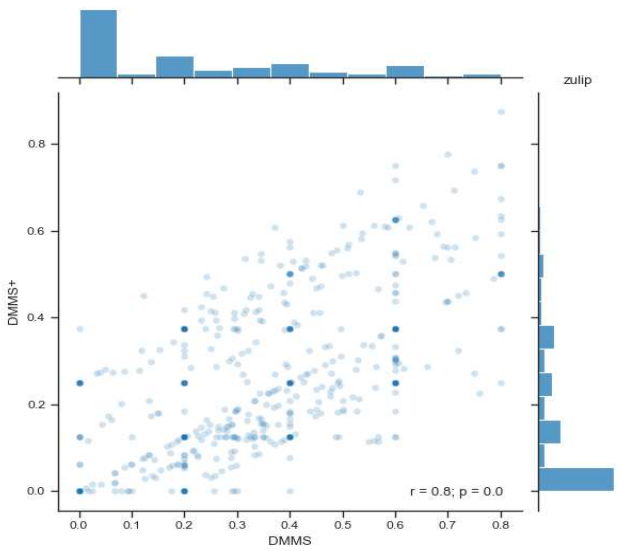
Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Django r = 0.82



Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Odoo r = 0.84



Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Saleor r = 0.77



Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Zulip r = 0.77

Додаток 3

Фрагмент змін внесений комітом 2798c937deb6625a4e6a36e70d4d60ce5faac954 в репозиторій Django (https://github.com/django/django/commit/2798c937deb6625a4e6a36e70d4d60ce5faac954)

```
--- a/django/db/models/expressions.py
+++ b/django/db/models/expressions.py
@@ -402,6 +402,18 @@ class BaseExpression:
     def copy(self):
         return copy.copy(self)

+    def prefix_references(self, prefix):
+        clone = self.copy()
+        clone.set_source_expressions(
+            [
+                F(f"{prefix}{expr.name}")
+                if isinstance(expr, F)
+                else expr.prefix_references(prefix)
+                for expr in self.get_source_expressions()
+            ]
+        )
+        return clone
+
     def get_group_by_cols(self, alias=None):
         if not self.contains_aggregate:
             return [self]

--- a/django/db/models/sql/compiler.py
+++ b/django/db/models/sql/compiler.py
@@ -912,10 +912,15 @@ class SQLCompiler:
                 ):
                     item = item.desc() if descending else item.asc()
                 if isinstance(item, OrderBy):
-                    results.append((item, False))
+                    results.append(
+                        (item.prefix_references(f"{name}{LOOKUP_SEP}"), False)
+                    )
                     continue
                 results.extend(
-                    self.find_ordering_name(item, opts, alias, order,
already_seen)
+                    (expr.prefix_references(f"{name}{LOOKUP_SEP}"), is_ref)
+                    for expr, is_ref in self.find_ordering_name(
+                        item, opts, alias, order, already_seen
+                    )
                 )
             return results
```

Додаток 4

Порівняльна таблиця метричних даних змін внесених до файлу django/db/models/expressions.py комітом 2798c937deb6625a4e6a36e70d4d60ce5faac954

			A		**B**	
Властивість вихідного коду	**Метрика**	**Критерій ризику**	**Кількість**	**SLOC**	**Кількість**	**SLOC**
Пов'язаність класу	LCOM	LCOM = 1	15	225	15	225
		LCOM = 2	13	937	13	948
Складність класу	WMC	WMC ≤ 11	15	225	15	225
		WMC > 11	13	937	13	948
Кількість методів класу	NOM	NOM ≤ 6	15	265	15	265
		NOM > 6	13	897	13	908
Складність методу (CC)	VG	VG ≤ 2	153	515	153	515
		VG > 2	41	570	42	581
Розмір методу	MLOC	MLOC ≤ 10	166	558	166	558
		MLOC > 10	28	527	29	538
Кількість параметрів метода	PAR	PAR ≤ 2	163	706	164	717
		PAR > 2	31	379	31	379
Дублювання	DCL	DCL= 0	1481	1492	1481	1492
		DCL > 0	6	6	6	6
Залежність модуля	MC	MC ≤ 10	0	0	0	0
		MC > 10	1	15	1	15

Порівняльна таблиця метричних даних змін внесених до файлу django/db/models/sql/compiler.py комітом 2798c937deb6625a4e6a36e70d4d60ce5faac954

			A		B	
Властивість вихідного коду	**Метрика**	**Критерій ризику**	**Кількість**	**SLOC**	**Кількість**	**SLOC**
Пов'язаність класу	LCOM	LCOM = 1	5	1392	5	1397
		LCOM = 2	0	0	0	0
Складність класу	WMC	WMC ≤ 11	1	18	1	18
		WMC > 11	4	1374	4	1379
Кількість методів	NOM	NOM ≤ 6	4	332	4	332
		NOM > 6	1	1060	1	1065
Складність методу (CC)	VG	VG ≤ 2	13	90	13	90
		VG > 2	32	1294	32	1299
Розмір методу	MLOC	MLOC ≤ 10	19	130	19	130
		MLOC > 10	26	1254	26	1259
Кількість параметрів метода	PAR	PAR ≤ 2	35	1025	35	1025
		PAR > 2	10	359	10	364
Дублювання	DCL	DCL= 0	1784	1784	1789	1789
		DCL > 0	0	0	0	0
Залежність модуля	MC	MC ≤ 10	0	0	0	0
		MC > 10	1	24	1	24

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1

Підвищення ефективності оцінки супроводженості об'єктно-орієнтованого програмного забезпечення методами статистичного аналізу

Студент: Лихвар Артем Володимирович, ППЗМ-71
Науковий керівник: к.т.н., доц., Негоденко Олена Василівна

Київ-2022

2

Актуальність роботи

Проблема: вплив дефектів на сучасні системи програмного забезпечення, які все глибше інтегруються в життєво важливі сфери, від керування критичної інфраструктури до пілотування транспортними засобами.

Об'єкт: процес супроводженння об'єктно-орієнтованого програмного забезпечення.

Предмет: методи та моделі підтримки супроводженння об'єктно-орієнтованого програмного забезпечення.

Мета: підвищення якості та рівня супроводженості об'єктно-орієнтованого програмного забезпечення за рахунок зменшення кількості дефектів.

Завдання: модифікація моделі дельта супроводженості (DMM) шляхом розширення вимірюваних властивостей вихідного коду.

3

Існуючі моделі якості програмного забезпечення

Назва	Призначення	Переваги	Недоліки
МакКола	Визначає взаємозв'язок між атрибутами якості програмного забезпечення.	Супроводженість, як внутрішній фактор якості пов'язується із Модифікуванням продукту, і визначається як обсяг зусиль необхідних для визначення та виправлення дефекту програми в операційному середовищі.	Супроводженість як і всі внутрішні фактори якості вимірюється опосередковано, через пов'язані властивості програмного забезпечення, без зазначення конкретних метрик чи способів вимірювання.
Боема	"Загальна корисність" - основа якості програмного забезпечення, що має найвищий пріоритет, і підпорядковує усі інші характеристики.	Загальна якість системи визначається сумарним значенням всіх характеристик.	Модель не передбачає будь яких визначень або формулювань вимірювань або показників.
Друмі	Спирається на характеристики продукту, що мають бути використані для показників та вимірювання атрибутів якості.	Супроводженість визначається як атрибут якості пов'язаний із характеристиками продукту.	Властивості програмного забезпечення за своєю природою, не демонструють якість.
Об'єктно-орієнтованого дизайну (QMOOD)	Складається з 4 рівневої ієрархічної структури та включає набір метрик призначених для вимірювання атрибутів якості.	Сфокусована лише на його підхарактеристиці - зрозумілості, що має дозволити використовувати модель на більш ранніх стадіях розробки.	

4

Існуючі метрики програмного забезпечення

Назва метрики	Призначення	Переваги	Недоліки
Кількість рядків коду (LOC)	вимірювання розміру програми	простота та легкість імплементації	не є дуже показовою, оскільки кількість рядків коду залежить від складності алгоритму, мови програмування, конвенційних стандартів кодування
Цикломатична складність (CC)	базується на теорії графів та використовується для обчислення складності	дослідження встановлює кореляцію між складністю програм та кількістю помилок	складний код потребує більше тестів, для врахування всіх можливих шляхів
Міри складності Холстеда	кількість рядків коду використовується для вимірювання складності коду	використовуються для вимірювання якості та супроводженості програмного забезпечення	не врахування структурних особливостей програмного коду, взаємодії між одиницями програми
Показники Лі та Генрі	визначаються 5 складовими, що характеризують різні аспекти програмного забезпечення	прості в обчислені, і більшість з них можуть бути визначені на етапі проектування за допомогою UML діаграм	розмір властивостей визначених класом (SIZE2) здебільшого залежить від мови імплементації
Метрики об'єктно-орієнтованого дизайну (MOOD)	визначається 6 метриками, що характеризують різні аспекти програмного забезпечення	досліджують показники не на рівні програми в цілому, а на рівні класів, а тому є більш ефективними з точку зору щоденної інженерної практики	

5

Модель дельта супроводженості об'єктно-орієнтованого програмного забезпечення DMM

Властивості системи та їх зв'язок із пороговими значеннями, призначеними для визначення ризику вихідного коду в

Властивість вихідного коду	Опис	Критерій низького ризику
Дублювання	Повторенням більше ніж 6 строк вихідного коду на текстовому рівні, із урахуванням пробілів	Рядки коду, що не мають дублювань
Розмір одиниці програми	Кількість рядків коду (LOC) одиниці програми, за виключенням коментарів, або строк складених лише з пробілів	Одиниці програми, що містять до 15 рядків коду (LOC)
Складність одиниці програми	Цикломатична складність Мак Кабе (CC) одиниці програми (підпрограми, функції, методу)	Одиниці програми зі складністю не більше 5
Розмір інтерфейсу одиниці програми	Кількість формальних параметрів інтерфейсу	Одиниці що мають не більше 2 параметрів
Залежність модуля	Кількість вхідних залежностей модуля. Модулем є групуючий елемент одиниць програми, файл.	Модуль що має не більше 10 вхідних залежностей

6

Математична модель дельта супроводженості об'єктно-орієнтованого програмного забезпечення (DMM)

Показник	Порядок обчислення
Дельта профіль ризику (RDP)	RPD(f^1, f, cp, r) = $LOC(f, cp, r)$ - $LOC(f^1, cp, r)$ де LOC - кількість рядків коду, r - категорія ризику, cp - властивість коду, f - файл версії із змінами, f^1 - файл попередньої версії
Дельта профілю низького ризику (LRPD)	LRPD(cp) = CRPDI(cp, низький) + $\sum_{h \in (\text{середній, високий, дуже високий})} CRDPD(cp, h)$ де CRPDI(cp, r) = $\sum_{\{f^1, f\} \in D}$ RPDI (f^1, f, cp, r) RPDI(f^1, f, cp, r) = max(0, RPD(f^1, f, cp, r))
Дельта профіля високого ризику (HRPD)	HRPD(cp) = CRPDD(cp, низький) + $\sum_{h \in (\text{середній, високий, дуже високий})} CRDPI(cp, h)$ де CRPDD(cp, r) = $\sum_{\{f^1, f\} \in D}$ RPDD (f^1, f, cp, r) RPDD(f^1, f, cp, r) = | min(0, RPD(f^1, f, cp, r))
Дельта показник властивості коду (DS)	DS(cp) =$LRDP(cp)$ / ($LRDP(cp)$+$HRPD(cp)$)
Показник дельта супроводженості (DDMS)	CP = {Дублювання, Розмір одиниці програми, Складність одиниці програми, Взаємодія одиниці програми, Залежність модуля} DDMS = $\sum_{cp \in CP} DS(cp)$ / |CP|

7

Модифікована модель дельта супроводженості об’єктно-орієнтованого програмного забезпечення (DMM+)

RC = {низький, високий}

CP = {Пов'язаність класу, Складність класу, Складність методу, Кількість методів, Розмір методу, Кількість параметрів, Дублювання, Залежність модуля}

Властивості вихідного коду та їх зв’язок із пороговими значеннями, призначеними для визначення ризику вихідного коду

Властивість вихідного коду	Коротка назва	Опис	Критерій низького ризику
Пов'язаність класу	LCOM	Відсутність пов'язаності в методах LCOM4	LCOM = 1
Складність класу	WMC	Зважені методи на клас WMC, з обрахування цикломатичної складності CC	WMC ≤ 11
Складність методу	VG	Цикломатична складність методу	VG ≤ 2
Кількість методів	NOM	Кількість методів одного класу	NOM ≤ 6
Розмір методу	MLOC	Розмір методу в рядках коду (SLOC)	MLOC ≤ 10
Кількість параметрів	PAR	Кількість формальних параметрів інтерфейсу метода	PAR ≤ 2
Дублювання	DCL	Повторенням більше ніж 6 строк вихідного коду на текстовому рівні, із урахуванням пробілів, без урахування строк без символів.	DCL = 0
Залежність модуля	MC	Кількість вхідних залежностей модуля. Модулем є групуючий елемент одиниць програми, файл.	MC ≤ 10

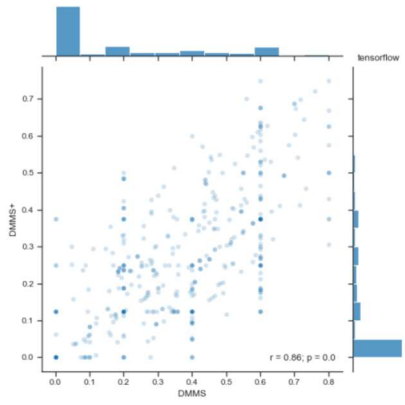
8

Показники вхідних програмних продуктів для порівняльного аналізу

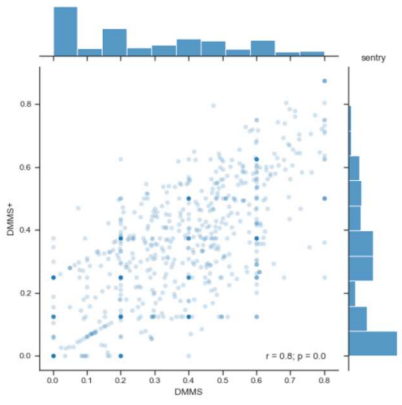
Репозиторій	Гілка	Дата першого комміту	Кількість комітів	Кількість комітів до Python файлів	Кількість файлів	Кількість модулів Python	Кількість класів (DSC)	Кількість методів
sentry	master	12.05.2008	43102	20807	10016	2256	3141	8247
zulip	main	28.08.2012	47222	17956	5712	1102	1050	1264
saleor	main	12.02.2013	19611	10999	2260	1428	2274	2484
django	main	13.07.2005	30744	16276	6625	854	1753	6844
odoo	15.0	07.12.2006	148369	87689	29301	3351	2660	13956
tensorflow	master	06.11.2015	129359	36224	26482	1240	1963	11991

9

Результати вимірювання показників моделі DMM та DMM+



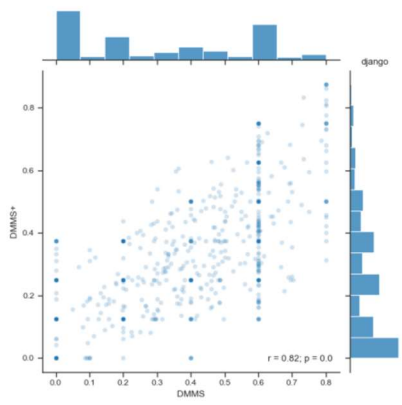
Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Tensorflow. r = 0.86



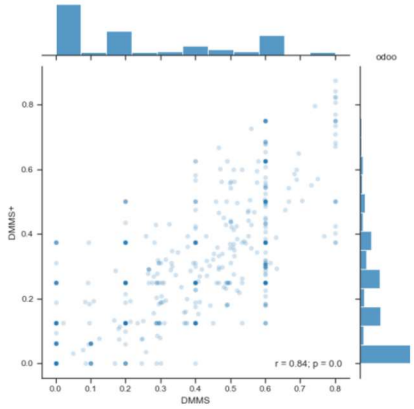
Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Sentry. r = 0.8

10

Результати вимірювання показників моделі DMM та DMM+



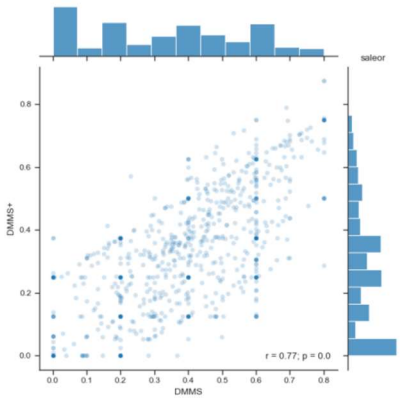
Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Django. r = 0.82



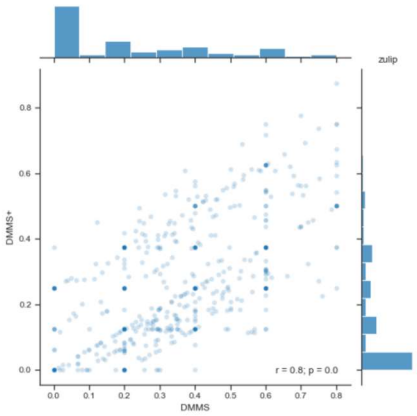
Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Odoo. r = 0.84

11

Результати вимірювання показників моделі DMM та DMM+



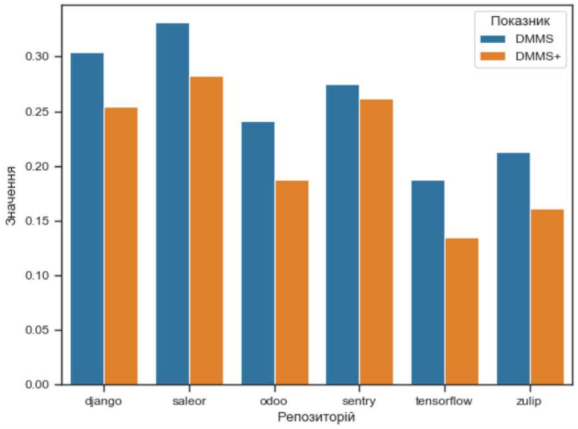
Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Saleor. r = 0.77



Ілюстрація кореляції показників DMMS та DMMS+ згідно аналізу 1000 внесених змін в репозиторії Zulip. r = 0.77

12

Зміни в показнику абсолютної середньої різниці між показниками DMMS та DMMS+ за результатами аналізу 1000 внесених змін до кожного з репозиторіїв



13

Порівняльна таблиця метричних даних змін внесених до файлу django/db/models/expressions.py

			A		B	
Властивість вихідного коду	**Метрика**	**Критерій ризику**	**Кількість**	**SLOC**	**Кількість**	**SLOC**
Пов'язаність класу	LCOM	LCOM = 1	15	225	15	225
		LCOM = 2	13	937	13	948
Складність класу	WMC	WMC ≤ 11	15	225	15	225
		WMC > 11	13	937	13	948
Кількість методів класу	NOM	NOM ≤ 6	15	265	15	265
		NOM > 6	13	897	13	908
Складність методу (CC)	VG	VG ≤ 2	153	515	153	515
		VG > 2	41	570	42	581
Розмір методу	MLOC	MLOC ≤ 10	166	558	166	558
		MLOC > 10	28	527	29	538
Кількість параметрів метода	PAR	PAR ≤ 2	163	706	164	717
		PAR > 2	31	379	31	379
Дублювання	DCL	DCL= 0	1481	1492	1481	1492
		DCL > 0	6	6	6	6
Залежність модуля	MC	MC ≤ 10	0	0	0	0
		MC > 10	1	15	1	15

14

Порівняльна таблиця метричних даних змін внесених до файлу django/db/models/sql/compiler.py

			A		B	
Властивість вихідного коду	**Метрика**	**Критерій ризику**	**Кількість**	**SLOC**	**Кількість**	**SLOC**
Пов'язаність класу	LCOM	LCOM = 1	5	1392	5	1397
		LCOM = 2	0	0	0	0
Складність класу	WMC	WMC ≤ 11	1	18	1	18
		WMC > 11	4	1374	4	1379
Кількість методів	NOM	NOM ≤ 6	4	332	4	332
		NOM > 6	1	1060	1	1065
Складність методу (CC)	VG	VG ≤ 2	13	90	13	90
		VG > 2	32	1294	32	1299
Розмір методу	MLOC	MLOC ≤ 10	19	130	19	130
		MLOC > 10	26	1254	26	1259
Кількість параметрів метода	PAR	PAR ≤ 2	35	1025	35	1025
		PAR > 2	10	359	10	364
Дублювання	DCL	DCL= 0	1784	1784	1789	1789
		DCL > 0	0	0	0	0
Залежність модуля	MC	MC ≤ 10	0	0	0	0
		MC > 10	1	24	1	24

15

Висновки

1. Проаналізовано існуючі моделі та метрики вимірювання об'єктно-орієнтованого програмного забезпечення, визначені їх переваги та недоліки.
2. Модефіковано модель дельта супроводженості об'єктно-орієнтованого програмного забезпечення, шляхом розширення вимірюваних властивостей вихідного коду, визначення їх взаємозв'язку з підхарактеристиками супроводженості, що визначені ISO/IEC 25010:2016. Визначено способи вимірювання та порогові значення.
3. Проведено практичну валідацію запропонованої моделі шляхом аналізу програмних продуктів з відкритим вихідним кодом, імплементованих із застосуванням об'єктно-орієнтованої парадигми програмування, процесом розробки із застосуванням систем контролю версій та значною кількістю учасників процесу розробки та тривалою історією змін коду.
4. Для порівняльного аналізу використання модефікованої моделі було обрано 6 програмних продуктів різною функціональної призначенності, з відкритим вихідним кодом, з імплементацією об'єктно-орієнтованої частини мовою програмування Python.
5. Встановлено, що відповідно до результатів проведеного аналізу значний показник коефіцієнта кореляції Пірсона в значенні від 0.77 до 0.86 демонструє сильну позитивну кореляцію між значеннями DMMS та DMMS+. Таким чином, показники вимірювання DMMS+ разом із DMMS відображають взаємозв'язок із змінами вихідного коду, що впливають на супроводженість. Значення показників DMMS та валідація були підтверджені в ході емпіричних досліджень.
6. Продемонстровано стабільність та ефективність вимірювання змін об'єктно-орієнтованого програмного забезпечення шляхом порівняльного аналізу внесених змін вихідного коду, що дає можливість проведення вимірювань супроводженості в процесах з методологічними підходами безперервної доставки та неперервної інтеграції. При цьому інтерпретація результатів оцінювання дає можливість проведення причинно-наслідкового зв'язку та усунення недоліків.

16

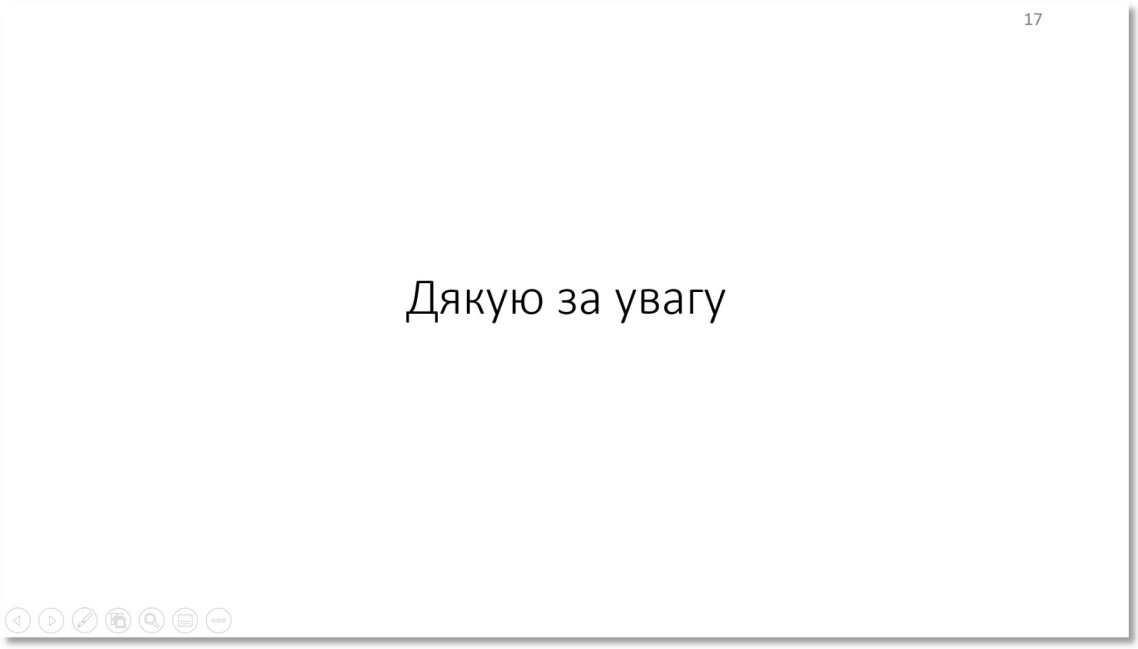
Публікації

За матеріалами опубліковано такі тези:

1. Лихвар А.В. Модифікована модель дельта супроводженості об'єктно-орієнтованого програмного забезпечення / Лихвар А.В., Негоденко О.В. // Науково-технічній конференції «Застосування програмного забезпечення в ІКТ», (м. Київ, 20 квітня 2022 року). – Kyiv, 2022. – с. 127

та такі статті:

1. Лихвар А.В. Підвищення ефективності оцінки супроводженості об'єктно-орієнтованого програмного забезпечення методами статистичного аналізу / Негоденко О.В., Золотухіна О.А., Коваленко Д.С. // Телекомунікаційні та інформаційні технології. – Kyiv, 2021. – № 4 (73) – с. 4-11


17
Дякую за увагу