Міністерство освіти і науки України
Харківський національний університет
радіоелектроніки

Факультет _____комп'ютерних наук_____
(повна назва)

Кафедра _____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

ДОСЛІДЖЕННЯ МЕТОДІВ АНАЛІЗУ ДИНАМІКИ ЗМІН У РЕПОЗИТОРІЯХ ПЗ

(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-2__
спеціальності _121 – Інженерія програмного забезпечення_____
(код і повна назва спеціальності)
освітньо-наукової програми _Інженерія програмного_забезпечення_____
(повна назва освітньої програми)

_____Свірчков В.В._____
(прізвище, ініціали)

Керівник _____проф. Лєсна Н.С._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук

Кафедра програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)

Освітньо-наукова програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____
(підпис)
«_____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

Студентові Свірчкову Володимиру Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів аналізу динаміки змін у репозиторіях ПЗ
затверджена наказом по університету від «__»_____2020 р № ____

2. Термін подання студентом роботи до екзаменаційної комісії «11» травня 2020 р.

3. Вихідні дані до роботи *Алгоритми обробки великих обсягів даних, алгоритми аналізу дінаміки, методи стримінгу великих даних та пояснювальна записка. Використовувати ОС Windows, середовище об'єктно-орієнтованого проектування.*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз проблемної галузі і постановка задачі, методи пошуку корисних даних, опис об'єктних моделей, використовувані. методи та алгоритми, архітектура програмної системи, опис розробленої програмної системи. , результати тестування програмної системи*

5. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта	
		підпис	дата
Спецчастина	проф. Лєсна Н.С.		

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1.	Аналіз предметної галузі	25 березня 2020 р.	
2.	Огляд існуючих методів	31 березня 2020 р.	
3.	Методи розробки рефакторінгу опен-соур проектів	15 квітня 2020 р.	
4.	Підготовка пояснювальної записки	20 квітня 2020 р.	
5.	Спецчастина	28 квітня 2020 р.	
6.	Підготовка презентації та доповіді	03 травня 2020 р.	
7.	Попередній захист	05 травня 2020 р.	
8.	Нормоконтроль, рецензування	07 травня 2020 р.	
9.	Занесення диплома в електронний архів	08 травня 2020 р.	
10.	Допуск до захисту в зав. кафедри	10 травня 2020 р.	

Дата видачі завдання _ «____»_ _____2020 р.

Студент _____
(підпис)

Керівник роботи _____ __проф. Лєсна Н.С._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 91 с., 9 табл., 35 рис., 3 дод., 28 джерел.

ВІДКРИТЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СТАТИСТИЧНІ МЕТОДИ, МЕТРІКИ, ПРОГНОЗУВАННЯ, ІНТЕНСИВНІСТЬ РОБОТИ НАД ПРОЕКТОМ

Об'єкт дослідження: процеси розробки проектів з відкритим кодом, методи збору інформації про історію розвитку Open Source проектів, методи оцінки стану проектів.

Мета дослідження – оцінити стан і перспективи розвитку Open Source проектів на основі оцінки метрик, що характеризують історію роботи над проектами.

Методи дослідження – моделі і методи розрахунку метрик складності програмного забезпечення, статистичний аналіз.

В результаті розроблено алгоритми, що дозволяють керівникам розробки проектів більш ефективно планувати подальшу розробку.

OPEN SOURCE SOFTWARE, STATISTICAL METHODS, METRICS, FORECASTING, INTENSITY OF WORK ON THE PROJECT

Object of study: open source project development processes.

Subject of research: methods for collecting information about the history of the development of Open Source project, methods for assessing the status of projects.

The purpose of the study is to assess the state and prospects of development of Open Source projects based on the assessment of metrics characterizing the history of work on projects.

Methods for analyzing the state of open source projects based on the analysis of metrics that characterize the intensity of work on projects have been further developed, making it possible to increase the validity of decisions made on managing open source projects or on using the results of these projects.

ЗМІСТ

Вступ ……6
1 Аналіз проблеми моніторингу репозиторіїв open sourse проектів……..8
1.1 Аналіз властивостей репозиторіїв OS проектів …….……8
1.2 Аналіз діяльності репозиторіїв проектів з відкритим вихідним кодом .12
1.3 Огляд типів ліцензій відкритого ПЗ ……...15
1.4 Постановка задач дослідження ……...17
2 Аналіз методів оцінювання динаміки зміни репозиторіїв open source проектів.19
2.1 Аналіз метрик оцінки складності программ ……19
2.2 Метрики складності потоку керування програми ……23
2.3 Аналіз факторів і висновків програмного експерименту ……31
2.4 Аналіз алгоритмів первинного перетворення даних ……..…...32
3 Аналіз результатів досліджень ……....36
3.1 Аналіз методів перевірки гіпотез ……36
3.2 Алгоритми обробки аналізу динаміки зміни репозиторіїв ……...38
3.3 Алгорити обчислення параметрів законів розподілу метрик класів …...41
4 Опис розробленого програмного забезпечення ……..45
4.1 Опис утіліти оцінювання метрік ……45
4.2 Аналіз кореляції між метриками проектів ……50
4.3 Реалізація алгоритму перевірки гіпотез ……54
5 Опис можливості використання отриманих результатів ……...57
5.1 Алгоритм розрахунку активності роботи над проектом з плином часу ..57
5.2 Аналіз метрік проектів Open Source ……...58
Висновки ……...64
Перелік джерел посилання …….....66
Додаток А Програмний код …….....69
Додаток Б Слайди презентації …….....76
Додаток В Апробація результатів роботи…….……..90

ВСТУП

Станом на жовтень 2019 року каталог Open Source проектів Ohloh.net містить інформацію про більш ніж 550 000 проектів. Згідно з дослідженням компанії SAP зростання кількості Open Source проектів за період з січня 2015 року по грудень 2018 року набули експонентного характеру.

На сьогоднішній день існує ряд міжнародних організацій (Apache Software Foundation, Open Source Initiative), що сприяють розвитку та поширенню Open Source проектів.

Серед проектів, створених за підтримки Apache Software Foundation слід зазначити Apache HTTP Server, Subversion, Cassandra, Hadoop і ін. Повний список доступний за посиланням.

Використання пропрієтарного програмного забезпечення передбачає наявність ряду обмежень, таких як неможливість вивчення, зміни вихідного коду, а також відсутність можливості вносити виправлення і модифікації.

У зв'язку зі значною кількістю існуючих Open Source проектів і експоненціальним характером зростання кількості нових проектів, а також з обмеженнями, які накладає пропрієтарних ПЗ, виникає необхідність оцінки поточного стану та перспектив розвитку Open Source проектів. Подібна оцінка може бути проведена на основі аналізу метрик, що характеризують історію роботи на проектами.

Аналіз метрик історії роботи над проектом може стати ефективним засобом оцінки поточного стану та перспектив розвитку Open Source проектів.

Об'єкт дослідження: процеси розробки проектів з відкритим кодом.

Предмет дослідження: методи збору інформації про історію розвитку Open Source проектів, методи оцінки стану проектів.

Методи дослідження – для аналізу метрик роботи над проектами будуть використані моделі і методи розрахунку метрик складності програмного забезпечення, статистичний аналіз.

Мета роботи – оцінити стан і перспективи розвитку Open Source проектів на основі оцінки метрик, що характеризують історію роботи над проектами.

Для досягнення поставленої мети вирішуються наступнізавдання:

- проаналізувати поточний стан репозиторіїв OS проектів;
- проаналізувати API даних сховищ;
- скласти набір розраховуються метрик історії розробки ПЗ;
- виконати планування експерименту зі збору та розрахунку метрик ПЗ;
- провести експеримент;
- виконати аналіз отриманих даних;
- сформулювати висновки на поточному етапі і перспективи розвитку з точки зору метрик роботи над проектом.

Отримали подальший розвиток методи аналізу стану проектів з відкритим кодом на основі аналізу метрик, які характеризують інтенсивність роботи над проектами, що дозволяє підвищити обґрунтованість прийнятих рішень з управління проектами з відкритим кодом, або по використанню результатів цих проектів.

Розроблений алгоритм прогнозу розвитку Open Source проектів, що дозволяє більш точно оцінювати поточний стан проекту, а також перспективи його розвитку.

Розроблені методи дозволяють керівникам розробки проектів більш ефективно планувати подальшу розробку, а кінцевим користувачам отримувати додаткову інформацію при вибору проекту.

1 АНАЛІЗ ПРОБЛЕМИ МОНІТОРИНГУ РЕПОЗИТОРІЇВ OPEN SOURSE ПРОЕКТІВ

1.1 Аналіз властивостей репозиторіїв OS проектів

Відкрите програмне забезпечення (Open source software) – програмне забезпеченняз відкритимвихіднимкодом. Вихідний код таких програм доступний для перегляду, вивчення та зміни, що дозволяє користувачеві взяти участь в доопрацюванні найбільш відкритою програми, використовувати код для створення нових програм і виправлення в них помилок – через запозичення вихідного коду, якщо це дозволяє сумісність ліцензій, або через вивчення використаних алгоритмів, структур даних, технологій, методик і інтерфейсів (оскільки вихідний код може істотно доповнювати документацію, а при відсутності такої сам служить документацією).

Відкрите програмне забезпечення чинить серйозний вплив на галузь програмного забезпечення в цілому, а також на його похідні процеси. Багато програмні продукти сьогодні містять принаймні, деякі компоненти з відкритим вихідним кодом. Деякі комерційні продукти є повністю відкритим програмним забезпеченням. На деяких ринках, наприклад, ринку веб-серверів, ПЗ з відкритим кодом має домінуючу частку ринку.

Згідно з дослідженням, проведеному компанією SAP кількість проектів з відкритим вихідним кодом за період з кінця січня 1995 до грудня 2006 досягло 4500.

Графік зростання кількість активних OS проектів згідно з даними сайту Ohloh.net вказано на рисунку 1.1.

Загальна кількість рядків вихідного коду за той же період часу перевищила один мільярд.

Графік зростання кількість рядків вихідного коду в проектах з відкритим вихідним кодом представлено на рисунку 1.2.

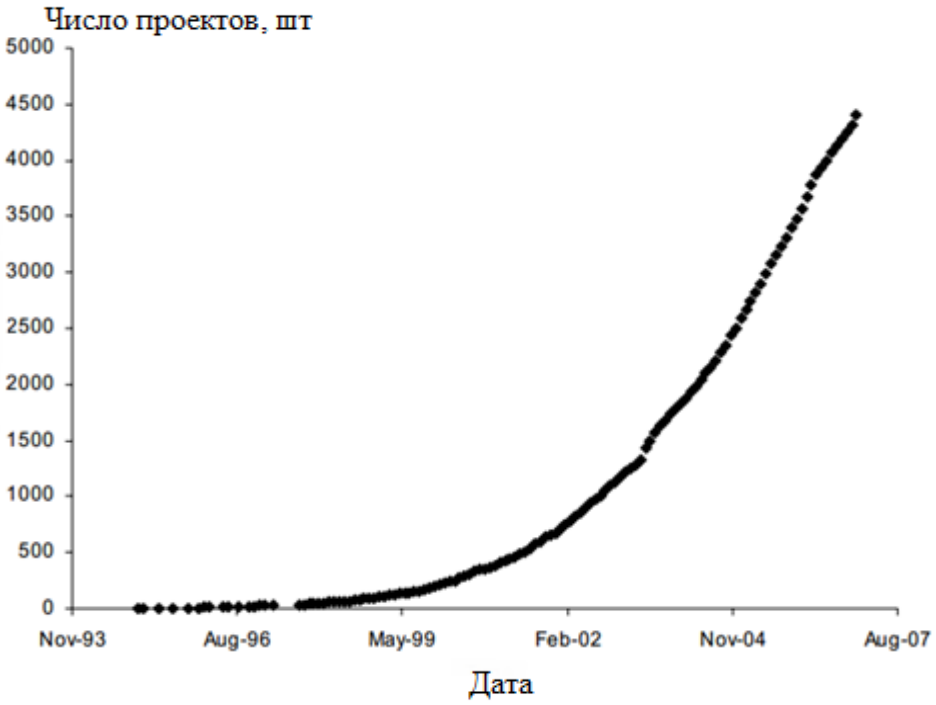

Рисунок 1.1 – Кількість OS-проектів

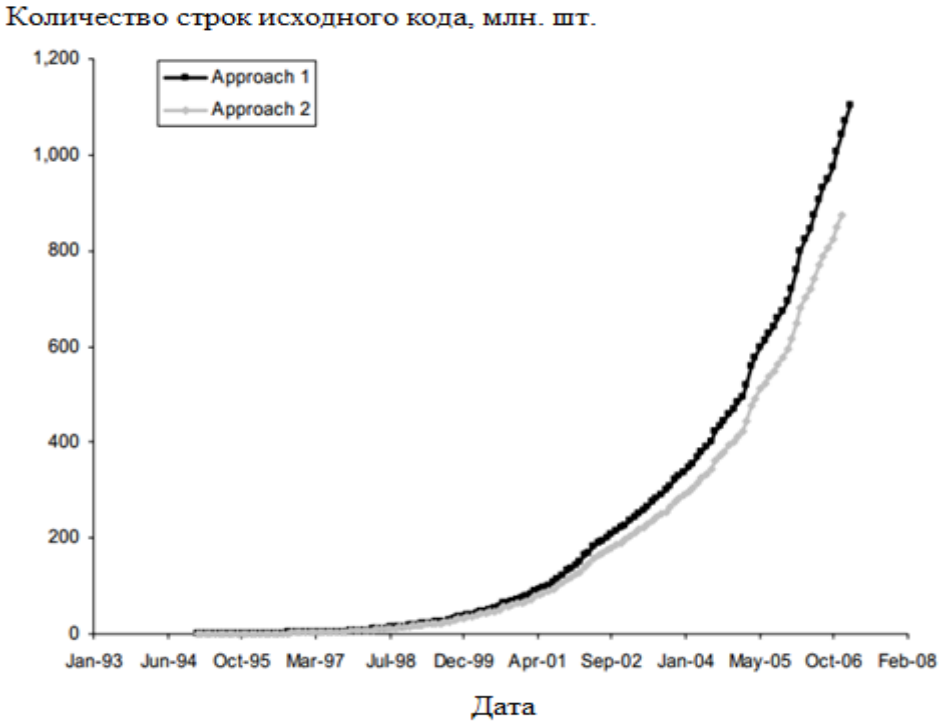

Рисунок 1.2 – Кількість рядків коду

Пропрієтарне програмне забезпечення – це програмне забезпечення, що є приватною власністю авторів або правовласників. Основною характеристикою

пропрієтарних ліцензій є те, що видавець ПЗ в ліцензії дає дозвіл її одержувачу використовувати одну або кілька копій програми, але при цьому сам залишається правовласником всіх цих копій. Один з наслідків такого підходу полягає в тому, що практично всі права на ПЗ залишаються за видавцем, а користувачеві передається лише дуже обмежений набір чітко окреслених прав. Для пропрієтарних ліцензій типово перерахування великої кількості умов, що забороняють певні варіанти використання ПЗ, навіть тих, які без цієї заборони були б дозволені законом про авторське право. Хорошим прикладом пропріє-тарної ліцензії може служити ліцензія на Microsoft Windows, яка включає великий список заборонених варіантів використання, таких як зворотна розробка, одночасна робота з системою декількох користувачів і поширення тестів її робочих характеристик.

Найбільш значним наслідком застосування пропрієтарної ліцензії є те, що кінцевий користувач зобов'язаний прийняти її, оскільки згідно із законом власником ПЗ є не він, а видавець програми. У разі відмови прийняти ліцензію користувач взагалі не може працювати з програмою.

Але більше іншого обмеження пропрієтарного програмного забезпечення позначаються на розробників програмного забезпечення, так як пропрієтарні ліцензії не тільки не надають доступу до вихідного коду ПЗ, але і забороняють повторно використовувати в інших продуктах навіть готові модулі і бібліотеки.

Очевидно, такий стан речей дає занадто багато незручностей IT-спільноти, а тому у нього існує альтернатива – вільне програмне забезпечення.

Вільне програмне забезпечення (ВПЗ) – широкий спектр програмних рішень, в яких права користувача («свободи») на необмежені установку, запуск, а також вільне використання, вивчення, поширення і зміна (вдосконалення) програм захищені юридично авторськими правами за допомогою вільних ліцензій. На практиці, для ВПО вихідні коди програмного забезпечення повинні бути доступні користувачам, поряд з виконуваними файлами і ліцензіями.

Рух ВПЗ зародився в 1983 році, коли Річард Столлман сформував ідею про необхідність дати програмну свободу (англ. Software freedom) користувачам. У

1985 році Столлман заснував Фонд вільного програмного забезпечення, щоб забезпечити організаційну структуру для просування своєї ідеї.

Для того щоб зберегти модель наукового співробітництва між розробниками, необхідно було забезпечити, щоб вихідні тексти програм, написаних розробниками, залишалися доступними для читання і критики всьому науковому співтовариству зі збереженням авторства творів. Для цього Річард Столлман сформулював поняття вільне програмне забезпечення, в якому відбилися принципи відкритої розробки програм в науковому співтоваристві, що склалося в американських університетах в 1970-і роки. Столлман явно сформулював ці принципи, вони ж – критерії вільного програмного забезпечення. Ці критерії обумовлюють ті права, які автори вільних програм передають будь-якому користувачеві:

− програму можна вільно використовувати з будь-якою метою («нульова свобода»);

− можна вивчати,як програма працює, і адаптувати її для своїх цілей («перша свобода»). Умовою цього є доступність вихідного тексту програми;

− можна поширювати копії програми – на допомогу товаришеві («друга свобода»);

− програму можна вільно покращувати і публікувати свою поліпшену версію – з тим, щоб принести користь всьому співтовариству («третя свобода»). Умовою цієї третьої свободи є доступність вихідного тексту програми і можливість внесення в нього модифікацій і виправлень.

Можливість виправлення помилок і покращення програм – найважливіша особливість вільного і відкритого програмного забезпечення, що просто неможливо для користувачів закритих приватних програм навіть при виявленні в них помилок і дефектів, кількість яких, як правило, невідомо нікому.

Поняттю вільного програмного забезпечення (freesoftware) близько поняття відкритого програмного забезпечення (opensourcesoftware).

Відкрите програмне забезпечення (англ. Open source software) – програмне забезпечення з відкритим вихідним кодом. Вихідний код таких програм

доступний для перегляду, вивчення та зміни,що дозволяє користувачеві взяти участь в доопрацюванні найбільш відкритою програми, використовувати код для створення нових програм і виправлення в них помилок – через запозичення вихідного коду, якщо це дозволяє ліцензія, або через вивчення використаних алгоритмів , структур даних, технологій, методик і інтерфейсів.

Термін «open source» створений в 1998 році Еріком Реймондом і Брюсом Перенс, які стверджували, що термін free software (безкоштовне ПЗ) в англійській мові неоднозначний і бентежить багатьох комерційних підприємців.

Переважна більшість відкритих програм є одночасно вільними. Визначення відкритого та вільного ПЗ в повному обсязі збігаються один з одним, але близькі, і більшість ліцензій відповідають обом.

Відмінність між рухами відкритого ПЗ і вільного ПЗ полягає в основному в пріоритетах. Прихильники терміна «opensource» роблять наголос на ефективність відкритих початкових кодів як методу розробки, модернізації та супроводу програм. Прихильники терміна «freesoftware» вважають, що саме права на вільне поширення, модифікацію і вивчення програм є головною перевагою вільного відкритого ПЗ.

1.2 Аналіз діяльності репозиторіїв проектів з відкритим вихідним кодом

Отже, відкрите ПЗ є багатим джерелом коду, відкритого для вивчення і повторного використання; воно надає можливість програмісту самому виправляти і вносити необхідні йому зміни в функціональність відкритих програм. Крім того, відкрите ПЗ зручно у використанні в освітніх і дослідницьких цілях, так як дозволяє вивчати підходи і методи, техніку і стиль написання коду іншими програмістами, а також дослідити розвиток і зміна програмного рішення протягом усього його часу розробки.

В зв'язку з необхідністю забезпечення публічного доступу до проектів з відкритим вихідним кодом, такі проекти зберігаються в публічних репозиторіях. Подібні репозиторії забезпечують загальний доступ вихідному коду, а також інформації про історію роботи і зміни, що вносяться в проект.

SourseForge.net – найбільший з існуючих на сьогоднішній день хостинг проектів з відкритим вихідним кодом. Станом на лютий 2009 року містив більш 230000 проектів. Однак значна частина проектів є неактивною.

Ohloh.net – безкоштовний публічний каталог програм з відкритим вихідним кодом,а також розробників

Ohloh побудований за принципом вики,будь-який бажаючий може приєднатися до спільноти і додавати нові проекти в каталог або робити виправлення в існуючих проектах. Станом на вересень 2019 року на сервісі зареєстровано 538117 проектів.

Ohloh – це не хостинг проектів з вихідним кодом, а служба, що дозволяє аналізувати ці проекти. Ohloh дозволяє аналізувати дані по окремим проектам – активність по проекту, кількість рядків вихідного коли, кількість розробників, інформація про активність кожного розробника, мови програмування, використовувані ліцензії і т.д. А також дозволяє порівнювати різні проекти між собою.

GitHub – веб-сервіс для хостингу проектів і їх спільної розробки. Заснований на системі контролю версій Git. GitHub є «соціальною мережею для розробників ». Крім розміщення коду, учасники можуть спілкуватися, коментувати правки один одного, стежити за новинами. За допомогою широких можливостей Git програмісти можуть об'єднувати свої репозиторії і GitHub пропонує зручний інтерфейс для цього і може відобразити внесок кожного учасника у вигляді дерева. Для проектів є особисті сторінки, невеликі Вікі, і система стеження завадами. Прямо на сервісі можна переглянути файли проектів з підсвічуванням синтаксису для більшості мов. На платних тарифних планах можна створювати репозиторії, доступні тільки обмеженому колу користувачів. Код проектів можна не тільки скопіювати через Git,а й скачати звичайний архів з

сайту. Крім Git, сервіс підтримує отримання і редагування коду через SVN і Mercurial.

Нижче наведено список вимог, який пред'являється до ліцензій на відкрите програмне забезпечення. Даний список був сформульований організацією Open Source Software Foundation.

Ліцензія не повинна обмежувати право будь-якого суб'єкта на продаж або безкоштовне розповсюдження програмного забезпечення як компонента сукупного набору програм, отриманих з різних джерел. Ліцензія не повинна вимагати відрахувань або інших виплат за подібне поширення.

Програма з відкритими кодами повинна включати вихідний код, і повинно допускатися її поширення в як у вигляді вихідного коду, так і в відкомпілювався вигляді. У тих випадках, коли який-небудь вид продукту поширюється без вихідного коду, повинні існувати широко оголошені способи отримання вихідного коду, вартість яких не перевищує обґрунтованих витрат на відтворення; прийнятний шлях є безкоштовне скачування з Internet. Вихідний код повинен бути кращою формою, в якій програмісти могли б модифікувати програму. Навмисне заплутування вихідного коду не допускається. Поширення проміжних форм представлення програм, такі як результати виводу препроцесора або транслятора, не допускаються.

Ліцензія повинна допускати створення модифікованих і похідних продуктів і повинна дозволяти їх розповсюдження на таких же умовах, що і оригінального програмного продукту.

Ліцензія може обмежувати розповсюдження вихідного коду в модифікованій формі, тільки якщо ліцензія допускає поширення разом з вихідним кодом «файлів-патчів» для зміни файлів в процесі створення виконуваної системи. Ліцензія повинна явно дозволяти розповсюдження програмного забезпечення, створеного на основі модифікованого вихідного коду. Ліцензія може вимагати, щоб назви або номера версій похідних продуктів відрізнялися від тих, які були у вихідного програмного забезпечення.

В ліцензії не повинні проявлятися пристрасті по відношенню до будь-яким особам або групам осіб.

Ліцензія не повинна накладати обмеження на застосування програми будь-якої області застосування. Наприклад, ліцензія не може обмежувати використання в області бізнесу або в області генетичних досліджень.

Права, приписані даній програмі, повинні поширюватися на всіх її одержувачів без потреби оформлення ними будь-якої додаткової ліцензії.

Права, приписувані даною програмою, не повинні залежати від того, чи є ця програма частиною будь-якого спеціального набору програм. Якщо програма витягується з цього набору та використовують або розповсюджують відповідно до умов її ліцензії, то всі суб'єкти, яким поширюється дана програма, повинні мати такі ж права, як і ті, які надаються разом з вихідним набором програм.

В ліцензії не повинні бути присутніми обмеження на інше програмне забезпечення, яке розповсюджується разом з ліцензованим програмним забезпеченням. Наприклад, в ліцензії не повинно вимагатися, щоб всі інші програми, поширювані на тому ж носії, що і дана програма, представляли собою програмне забезпечення з відкритими кодами.

Жодне з положень ліцензії не повинно грунтуватися на будь-якої індивідуальної технології або стилі інтерфейсу.

1.3 Огляд типів ліцензій відкритого ПЗ

Ліцензія Apache дає користувачеві право використовувати програмне забезпечення для будь-яких цілей, вільно поширювати, змінювати, і поширювати змінені копії, за винятком назви.

Дана ліцензія не ставить умовою незмінність ліцензії поширення програмного забезпечення, і не наполягає навіть на збереженні його

безкоштовного і відкритого статусу. Єдиною умовою, що накладається ліцензією Apache, є інформування одержувача про факт використання вихідного коду.

Ліцензія BSD – це ліцензійна угода, що вперше застосована для поширенняUNIX-подібнихопераційних систем BSD.

Існують дві основні версії ліцензії BSD, які необхідно розрізняти: «оригінальна» і так звана «модифікована» (другу в англомовній літературі часто називають New BSD License).

Права на вихідний дистрибутив BSD офіційно належать «піклувальникам університету Каліфорнії» – керуючому органу університету Каліфорнії. Причина цього полягає в тому, що BSD був розроблений в кампусі Берклі-університету Каліфорнії.

Ліцензія BSD допускає Власницьке комерційне використання ПЗ. Для ПЗ, випущеного під цією ліцензією, допускається вбудовування в пропрієтарні комерційні продукти. Роботи, засновані на такому ПЗ, навіть можуть поширюватися під пропрієтарними ліцензіями (але все ж повинні відповідати вимогам ліцензії). Найбільш помітні приклади таких програм – використання мережевого коду BSD в продуктах корпорації Microsoft, а також використання багатьох компонентів FreeBSDв операційній системі Mac OSX.

Ліцензія MIT (англ. MIT License) – ліцензія вільного програмного забезпечення розроблена Массачусетським технологічнимінститутом.Вона є дозвільної ліцензією, тобто дозволяє програмістам використовувати ліцензований код в закритому ПЗ за умови, що текст ліцензії надається разом з цим ПЗ. Ліцензія є GPL-сумісної, тобто дозволяє програмістам комбінувати і поширювати GPL продукти з софтом, який використовує MIT License.

GNU General Public- ліцензія на вільне програмне забезпечення, створена в рамках проекту GNU в 1988г. Друга версія цієї ліцензії була випущена в 1991році,третя версія, після багаторічної роботи і тривалої дискусії – в 2007році. GNU Lesser General Public License (LGPL) – це ослаблена версія GPL, призначена для деяких бібліотекПО.

GNU GPL – надає користувачеві права копіювати, модифікувати і поширювати (в тому числі на комерційній основі) програми, а також гарантувати, що і користувачі всіх похідних програм отримають перераховані вище права.

Microsoft Public License – це ліцензія Microsoft, роздільна поширення скомпільованої коду як для комерційного, так і для некомерційного використання під будь-який ліцензією, що підкоряється Ms-PL. Поширення самого вихідного коду можливо тільки під Ms-PL. Спочатку ліцензія називалася Microsoft Permissive License, проте була перейменована під час розгляду для схвалення OSI. На відміну від GPL гарантує свободу коду без права вимагати напрацювання на основі даного назад.

1.4 Постановка задач дослідження

Мета роботи – оцінити стан і перспективи розвитку Open Source проектів на основі оцінки метрик, що характеризують історію роботи над проектами.

Для досягнення поставленої мети вирішуються наступні завдання:

- проаналізувати поточний стан репозиторіїв OS проектів;
- проаналізувати API даних сховищ;
- скласти набір метрик історії розробки ПЗ, що розраховуються;
- виконати планування експерименту зі збору та розрахунку метрик ПЗ;
- провести експеримент;
- виконати аналіз отриманих даних;
- сформулювати висновки на поточному етапі і перспективи розвитку з точки зору метрик роботи над проектом;
- виконати економічне обґрунтування науково-дослідної роботи.

Програмне забезпечення з відкритим вихідним кодом поширюється на умовах відкритих ліцензій, які надають користувачам право використовувати, вивчати, поширювати і покращувати ці програми. Це робить відкрите ПЗ багатим

джерелом вихідного коду, придатного для застосування в освітніх і дослідницьких цілях, так як дозволяє вивчати підходи і методи, техніку і стиль написання коду іншими програмістами, а також дослідити розвиток і зміна програмного рішення протягом усього його часу розробки. Тому даний дослідження буде проводитися саме на базі відкритого ПЗ.

Серед основних недоліків ПЗ із закритим вихідним кодом слід зазначити неможливість аналізу його роботи без використання спеціалізованих програмних засобів, а також неможливість внесення змін і модифікацій в разі потреби.

Зазначені недоліки багато в чому зумовили збільшення кількості ПЗ з відкритим вихідним кодом. В даний час існує безліч фондів (FSF, ASF і ін.) Підтримують поширення і розвиток програмного забезпечення з відкритим вихідним кодом.

Зростання кількості проектів з відкритим вихідним кодом привів до виникнення репозиторіїв для подібних проектів (sourceforge.net, Ohloh.net і ін.), Які служать для упорядкування та організації таких проектів.

Зростання кількості проектів з відкритим вихідним кодом і як наслідок зростання числа репозиторіїв призводить до складності відстеження та порівняльного аналізу різноманітних проектів. Цим обумовлена необхідність розробки методу і ПЗ для моніторингу змін в репозиторіях проектів з відкритим вихідним кодом.

В рамках магістерської роботи буде досліджено метрики, що характеризують історію роботи над проектами. Як джерело вихідного коду для вивчення буде використано відкрите ПЗ.

2 АНАЛІЗ МЕТОДІВ ОЦІНЮВАННЯ ДИНАМІКИ ЗМІНИ РЕПОЗИТОРІЇВ OPEN SOURCE ПРОЕКТІВ

2.1 Аналіз метрик оцінки складності програм

Для оцінювання якості ПЗ існує множина різних методів. Якості програмного забезпечення (ЯПЗ) дають наступне визначення: "сукупність властивостей, що визначають корисність програми для користувачів відповідно до функціонального призначення й пред'явлених вимог" [9]. Оцінка ЯПЗ складається з характеристик, тобто факторів, що впливають на якість програм і піддаються виміру, і критеріїв якості, які показують ступінь відповідності програм оцінюваним характеристикам.

У даній роботі методи оцінки ЯПЗ застосовуються до програм, що написані за принципом структурного програмування. Структурне програмування це методологія розробки програмного забезпечення з розбивкою на керуючі блоки, що з'явилася через зрослу складність програм. При цьому вся програма будується на базі трьох керуючих структур: послідовності (then), розгалуження (if-then-else), циклу (while-do), без використання оператора goto.

Метрика програмного забезпечення (англ. software metric) – міра, що дозволяє одержати чисельне значення деякої властивості програмного забезпечення або його специфікацій . Метрики необхідні для того, щоб:

– показати, яким чином діяльність у деякій конкретній сфері функціонування підприємства вносить вклад у досягнення заздалегідь певних цілей;

– виявити зміни й істотні аномалії в процесах і прийняти обґрунтовані рішення по виправленню або поліпшенню процесів [11].

Таким чином, основною задачею введення метрик у процес функціонування компанії є необхідність додати потрібний напрямок вимірам, що відносяться до певної предметної області. В загальному випадку застосування метрик дозволяє

керівникам проектів і підприємств вивчити складність розробленого або навіть розроблювального проекту, оцінити обсяг робіт, стилістику розроблювальної програми й зусилля, витрачені кожним розроблювачем для реалізації того або іншого рішення .

Для виміру характеристик і критеріїв якості використаються метрики виміру якості програм. Як правило, метрики якості подають інформацію рекомендаційного характеру й використовуються для вивчення складності розроблювального ПЗ, оцінки затрат праці й обсягу робіт, необхідних для реалізації проекту, і дають оцінювання про можливості поліпшення програмного коду або оптимізації зусиль, затрачуваних розроблювачем на створення програми. Також, метрики якості можуть слугувати критерієм вибору варіантів розпаралелювання для програм, що функціонують у розподіленому режимі.

Самих метрик існує велика кількість. Є два основних напрямки їхнього дослідження:

– метрики для оцінки самого програмного забезпечення;
– метрики для оцінки умов розробки програм.

Метрики можуть давати оцінку про різні аспекти процесу розробки програми, такі як повнота технічних характеристик, якість розроблювальної програми й відповідність програми необхідним характеристикам.

Метрики оцінки якості програмного забезпечення надають уявлення про надійність програм, їхньої продуктивності, складності супроводу й зміни й т.д. У атстаційній роботі досліджуються метрики оцінки складності, які діляться на 3 основні групи:

– метрики розміру програм, або кількісні метрики;
– метрики складності потоку керування програми;
– метрики складності потоку даних програм.

При цьому метрики з однієї групи можуть використовуватися одночасно й у зв'язуванні з метриками з інших груп, для одержання більш точної оцінки певних характеристик або самого ПЗ в цілому.

Найпоширенішою й елементарною метрикою кількісної оцінки є тривіальна оцінка кількості рядків вихідного коду, що також позначається абревіатурою SLOC. Ця метрика дає гарні результати оцінки програми. Зрозуміло, у сучасних умовах оцінювати програми просто по обсягу тексту не представляється раціональним рішенням, тому що з'явилася величезна кількість мов програмування з різним синтаксисом і можливостями запису команд, не говорячи вже про порожні рядки й коментарі. Тому дана метрика розрізняє у вихідному коді логічні й фізичні рядки. Логічні рядки – це команди, або оператори програми, по них звичайно й виробляється оцінка.

Така оцінка, як правило, використовується для визначення затрат праці на створення ПЗ й незастосовна для серйозної оцінки складності програм, але її можна використати для класифікації програм по їхньому обсяги. Для оцінки ж програм зі схожим обсягом уже застосовуються оцінки інших факторів, наприклад, з інших груп метрик, як було сказано вище.

Одними з найпоширеніших метрик кількісної оцінки є метрики Холстеда [12]. У їхній основі лежить обчислення кількості операторів й операндів у вихідному коді програми. В основі даних метрик лежать наступні показники:

– *n1* – число унікальних операторів програми, включаючи символи-роздільники, імена процедур і знаки операцій (словник операторів);

– *n2* – число унікальних операндів програми (словник операндів);

– *N1* – загальне число операторів у програмі;

– *N2* – загальне число операндов у програмі.

Так, у прикладі кількість унікальних операторів програми дорівнює 9, число унікальних операндів програми 5, загальне число операторів у програмі – 20, загальне число операндів у програмі – 25.

Обсяг програми визначається в логічних одиницях програми – символах, операторах, операндах. Таким чином, у прикладі словник програми дорівнює 14, довжина програми дорівнює 45, а обсяг – 171,33075.

Також Холстед вводить теоретичні оцінки:

– n_1' – теоретичне число унікальних операторів,

– n_2' – теоретичне число унікальних операндів, по яких визначається теоретичний словник програми n' – функції, що необхідні для написання програми, з огляду на що вони вже реалізовані й робота програми зводиться до їхнього виклику,

$$N = N_1 + N_2 \tag{2.1}$$

Потім, з використанням n' визначається теоретичний обсяг програми, що відповідає мінімальному тексту програми:

$$V = N \times \log_2(n). \tag{2.2}$$

Теоретична довжина програми, у свою чергу, визначається як:

$$V = n' \times \log_2(n').$$

дає досить точне подання про довжину тексту програми, у коректно написаних програмах з точністю від 90%. За допомогою цих базових оцінок складності можна визначити також другорядні покажчики якості програми:

$$N' = n_1 \times \log_2(n_1) + n_2 \times \log_2(n_2) \tag{2.3}$$

Якість програмування, показує, наскільки грамотно написаний вихідний код програми, в ідеальній програмі L =1;

$L' = \frac{2n_2}{n_1 \times N_2}$ - якість програмування, що обчислюється, коли не відомий теоретичний обсяг програми;

$I = L' \times V = \frac{2n_2}{n_1 \times N_2} \times (N_1 + N_2) \times \log_2(n_1 + n_2)$ інтелектуальний зміст визначає складність написання програми;

$E = \frac{V}{L} = \frac{V^2}{V'}$ – оцінка розумових затрат праці, визначає кількість уявних рішень при написанні програми;

Переваги метрик Холстеда:

– не вимагають глибокого вивчення структури програми;
– прогнозують кількість помилок;
– прогнозують витрати на супровід;
– корисні при плануванні проектів;
– вимірюють загальну складність програм;
– легко обчислюються;
– можуть використовуватися для будь-яких мов програмування.

До недоліків слід віднести:

– залежать від закінченості вихідного коду;
– погано підходять для оцінки ПЗ на етапі проектування.

Ще одна метрика оцінки розміру програм – ABC метрика [13]. Вона записується як трійка значень $\{ A, B , C \}$, де A – кількість присвоювань значень змінним (Assignment), B – кількість викликів функцій (Branch) і кількість логічних перевірок C (Condition), і обчислюється по формулі:

$$ABC = \sqrt{A^2 + B^2 + C^2}.$$

2.2 Метрики складності потоку керування програми

Наступна група метрик це ті, які ґрунтуються на оцінці графа потоку керування (ГПК) програми замість її кількісних характеристик.

ГПК програми є орієнтований граф, що відображає керуючу структуру програми, графічне подання якого вперше було запропоновано Томасом Маккейбом в 1976 році.

Керуючий граф будується в такий спосіб (рисунок 2.1):

– вершинам графа відповідає лінійна ділянка коду, що не містить у собі керуючих крапок, які передають або приймають керування, але може включати один або декілька операторів програми;

– потоки керування програми в графі позначаються спрямованими дугами;

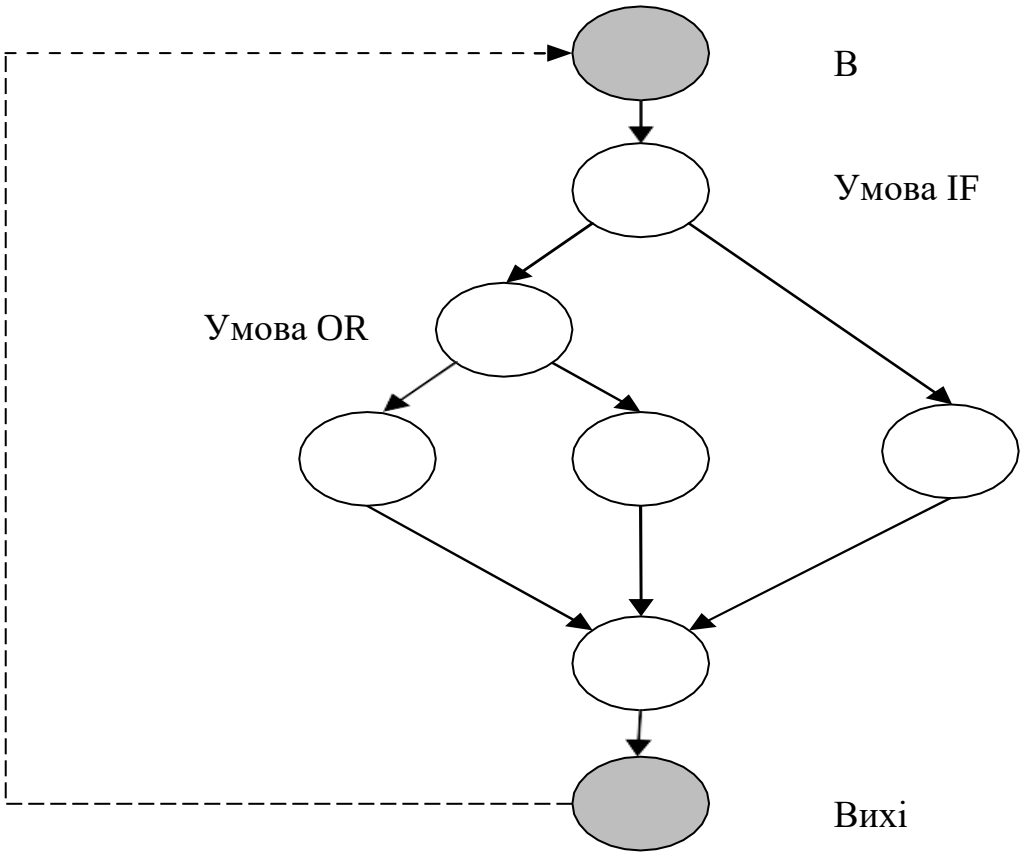

Рисунок 2.1 – Приклад графа потоку керування простої програми з однією умовою *if* і булевим оператором *or*

– вершини графа поділяються на два види: операторні – з яких виходить одна дуга, і предикатні – з яких виходять дві дуги, що відповідає умовам у програмі (наприклад, if). Треба відзначити, що в такий спосіб можна відображати тільки прості умови без додаткових операторів (and, or), умови з такими операторами називаються складовими й кожен оператор є окремою предикатною вершиною, керування в яку передається із предикатного вузла простої умови.

Як правило, у ГПК також присутні дві додаткові вершини – вхідна й вихідна. Кожна вершина графа повинна бути досяжна із вхідний, і вихідна вершина повинна бути досяжна з будь-якої іншої вершини.

Метрика цикломатичної складності програми й може бути обчислена по формулі [12]:

$$V = E - N + 2P$$

де E – число дуг у графі потоку керування програми,

N – число вершин,

P – число компонентів зв’язності графа.

Параметр P , як правило, приймається за число дуг, які необхідно додати до ГПК для перетворення його в сильно зв'язний граф, тобто граф, кожна вершина якого досяжна з будь-який іншої. Графи коректно написаних програм, що не мають недосяжних вершин із вхідної й не мають "висячих" точок входу й виходу перетворяться в сильно зв'язкові шляхом додавання дуги, що замикає вихідну й вхідну вершини, як показано на рис. 2.1 пунктирною лінією.

Тому для коректно написаних програм P приймається рівним 1, а формулу для розрахунку цикломатичної складності можна записати у вигляді:

$$V = E – N + 2$$

У ГПК програми, вихідний код якої показаний на рис. 2.1, число дуг цикломатична складність.

Спочатку Маккейбом пропонувалося використати оцінку цикломатичної складності для обмеження складності програм при їхній розробці. Він припустив, що складність розроблювальних модулів програми не повинна перевищувати 10, що згодом підтвердилося на практиці, однак, це є тільки рекомендацією, тому цикломатична складність у деяких випадках може перевищувати 10 [13].

Надалі в даній роботі саме ця метрика ляже як базис для оцінки складності програм, що здійснюють високоточні обчислення в паралельному режимі.

```
        A   =   function(n1,
   m1); S = 0;
        for (i = 0; i <100; i++)

        {
        if (A)
        {

        }
        else
        {

}
AfxMessageBox("Starting calculation");
AfxMessageBox("Error!");
```

```
            for (j = 0; j < m1; j++)
            {
                  S += S * i + n1 *
            j; if (S > A)
                  {
                        AfxMessageBox("Value found!");
                  }
            }
      }
```

Переваги оцінки цикломатичної складності:

– направляє процес тестування обмежуючи складність програмної логіки на етапі розробки;

– зручна в застосуванні.

Недоліки:

– це метрика виміру складності керування програми, а не її даних;

– при оцінці складності не враховується параметризація графів, тому програми зі схожими графами будуть оцінені однаково;

– при оцінці не відслідковується потік даних, що може також дати перекручене подання про складність програми.

Для виправлення цих недоліків існує кілька метрик, які є поліпшенням оцінки цикломатичної складності. Потрібно згадати їх для того, щоб одержати подання про можливості їхнього використання для оцінки високоточних паралельних обчислень.

Одна з таких метрик – метрика Г. Майерса, що запропонував для оцінки складності брати інтервал $[V(G), V(G_h)]$, у якому h для простих предикатів дорівнює нулю, а для n-місних предикатів $h = n - 1$. Це дозволяє розрізняти програми з однаковими ГПК й різними по складності предикатами, але на практиці така методика використається рідко.

Метрика Хансена [14] оцінює складність програм за допомогою пари {цикломатична складність, число операторів}. Дана метрика дуже чутлива до структурованості коду програми.

Топологічна метрика Чена оцінює складність програми по числу перетинань границь областей графа програми [14]. Метрика Чена може бути

застосована тільки у випадку структурованих програм з послідовним з'єднанням керуючих конструкцій, у зворотному випадку метрика істотно залежить від умовних і безумовних переходів. У такому випадку можна вказати верхню границю метрики, рівну числу логічних операторів при їхній взаємній вкладеності, збільшеному на 1, а нижня дорівнює 2. Коли керуючий граф програми має тільки один компонент зв’язності, метрика Чена збігається із цикломатичною метрикою Маккейба (рисунок 2.2).

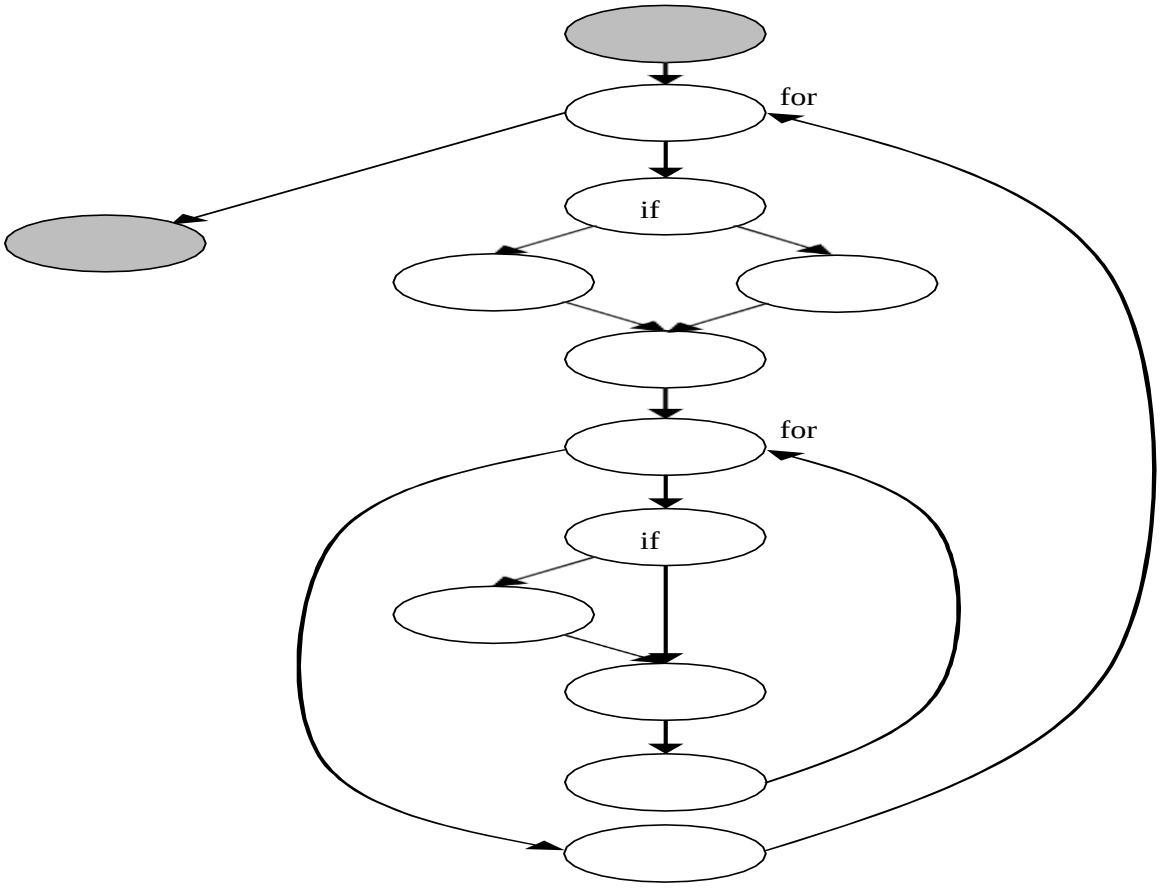

Рисунок 2.2 – ГПК програми для приклада

Для оцінки вкладеності й довжини програм використовуються метрики Харрисона, Мейджела [15]. З використанням будь-яких метрик оцінки складності фрагментів коду, описаних раніше, кожній вершині графа привласнюється її початкова складність. Для визначення сфери впливу предикатної вершини виділяється підграф, утворений вершинами, що перебувають на кінцях вихідних із предикатної вершини дуг, а також вершинами, досяжними з кожної такої

вершини (нижня границя підграфа), і вершинами, що лежать на шляхах із предикатної вершини в яку-небудь нижню границю.

Наведеною складністю предикатної вершини називається сума початкових або наведених складностей вершин, що входять у її сферу впливу, плюс первинна складність самої предикатної вершини.

Функціональна міра (SCOPE) програми – це сума наведених складностей всіх вершин керуючого графа.

Функціональним відношенням (SCORT) називається відношення числа вершин у керуючому графі до його функціональної складності, причому із числа вершин виключаються термінальні. SCORT може приймати різні значення для графів з однаковим цикломатичним числом.

Метрика Пивоварского є ще однією модифікацією метрики цикломатичної складності й дозволяє відслідковувати розходження не тільки між послідовними й вкладеними керуючими конструкціями, але й між структурованими й неструктурованими програмами. Вона розраховується як

$$N = V^* + \sum P_i,$$

де V * – модифікована цикломатична складність, обчислена так же, як й V .

Відмінність обчислення полягає: оператор CASE з n виходами розглядається як один логічний оператор, а не як n – 1 операторів, а Pi – глибина вкладеності i-ї предикатної вершини. Для підрахунку глибини вкладеності предикатних вершин використається число «сфер впливу». Під глибиною вкладеності розуміється число всіх «сфер впливу» предикатів, які або повністю втримуються в сфері розглянутої вершини, або перетинаються з нею. Глибина вкладеності збільшується за рахунок вкладеності не самих предикатів, а «сфер впливу». Міра Пивоварского зростає при переході від послідовних програм до вкладеного й далі до неструктурованих, що є її величезною перевагою перед багатьма іншими мірами даної групи.

Ще одна досить ефективна метрика – метрика Т. Джилба, що пропонує оцінювати складність програми по її насиченості вираженнями IF, THEN, ELSE

[7]. Ця метрика може використатися як для оцінки розміру програм, так і для оцінки ГПУ. Для цього використаються дві характеристики:

– CL – абсолютна складність програми, що характеризується кількістю операторів умови;

– *n* – загальне число операторів програми;

– *cl* – відносна складність програми, що характеризується насиченістю програми операторами умови, тобто *cl* визначається як відношення CL до загального числа операторів.

Додавання до метрики Джилба оцінки максимального рівня вкладеності дозволяє застосувати її до аналізу циклічних конструкцій.

Наступна розповсюджена й ефективна оцінка складності ПЗ – метрика граничних значень. Для даної метрики використовується орієнтований граф G (N, E). Вершини такого графа діляться на дві групи, залежно від кількості вхідних й вихідних з них дуг. Кількість вхідних дуг характеризує негативний ступінь вершини, а вихідних, відповідно – позитивний ступінь вершини. Одержано дві групи вершин:

– приймаючі вершини, ті, у яких позитивний ступінь менший або дорівнює 1;

– вершини відбору, у яких позитивний ступінь більше 1.

Для одержання оцінки складності граф G розбивається на максимально можливу кількість підграфів G', які повинні задовольняти умовам:

– вхідною вершиною підграфа є вершина відбору;

– у кожному підграфі повинна бути кінцева вершина – нижня границя підграфа, у яку можна потрапити з будь-якої іншої вершини підграфа.

Скоректована складність вершини відбору дорівнює кількості вершин, що утворять її підграф (не враховуючи вхідну вершину). Скоректована складність кожної вершини подграфа, крім кінцевої, дорівнює 1, а складність кінцевої вершини дорівнює 0. У прикладі на рис. 2.3 скоректована складність вершин показана в табл. 2.1.

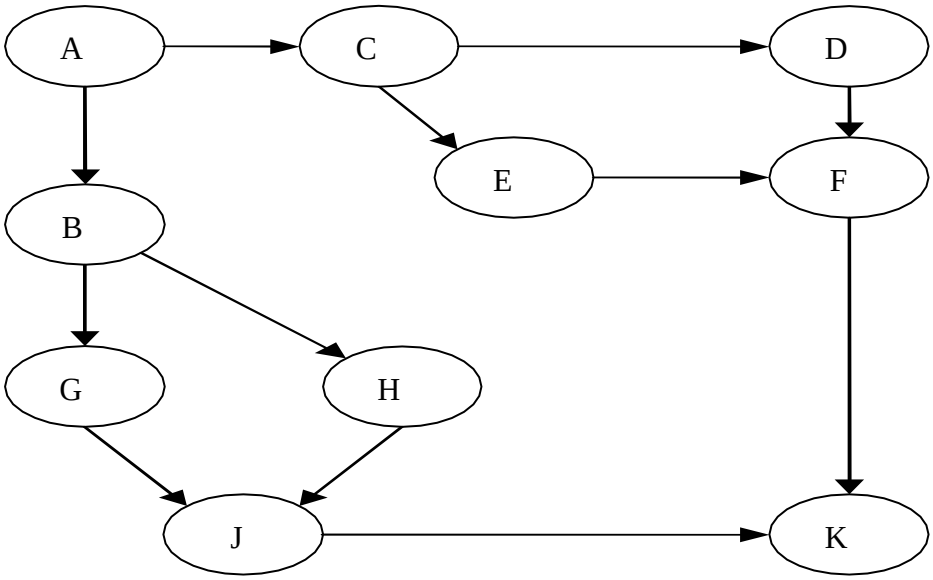

Рисунок 2.3 – Приклад графа для визначення складності по метриці граничних значень

Таблиця 2.1 – Скоректована складність вершин підграфа G

Характеристики підграфів програм	Вершини відбору		
	A	B	C
Вершини переходу	B, C	G, H	E, D
Скоректована складність вершини графа	9	3	3
Вершини підграфа		G, H	E, D
Нижня границя підграфа	K	J	F

Відносна гранична складність програми визначається за формулою:

$$S_0 = 1 - \frac{(N-1)}{S_a},$$

де N – загальне число вершин графу,

S_a – абсолютна гранична складність програми, одержувана додаванням скоректованих складностей всіх вершин графа G .

2.3 Аналіз факторів і висновків програмного експерименту

В термінології планування експериментів вхідні змінні і структурні допущення, що становлять модель, називаються факторами, а вихідні показники роботи – відгуками. Рішення про те, які параметри і структурні допущення вважати фіксованими показниками моделі, а які експериментальними чинниками, залежить від цілей дослідження.

У багатьох випадках фактори можуть носити не тільки кількісний, але і якісний характер. Тому значення факторів зазвичай називають рівнями. Якщо при проведенні експерименту можна змінювати рівень фактора, то експеримент називається активним, в іншому випадку – пасивним.

Факторами буде є інформація про історію змін вихідного коду різних Open Source проектами за деякий період часу. Джерелом даних є API сайту Ohloh.net. В ході експерименту ніяких змін в вихідні коди досліджуваних проектів вноситься не буде. Отже, експеримент носить пасивний характер.

Висновок програмного експерименту є набір числових характеристик, що описують історію роботи над проектом. Для кожного проекту будуть отримані такі метрики:

- кількість доданих рядків коду;
- кількість вилучених рядків коду;
- кількість доданих рядків коментарів;
- кількість вилучених рядків коментарів;
- кількість доданих порожніх рядків;
- кількість вилучених порожніх рядків;
- число розробників;
- число коммітов;
- число рядків коду;
- число коментарів;
- число порожніх рядків;

- число людино-місяцев;
- число користувачів;
- загальна кількість рядків коду;
- дата збору інформації.

2.4 Аналіз алгоритмів первинного перетворення даних

Після отримання вихідних даний необхідно провести їх перетворення до вигляду, придатного для подальшої обробки. Вихідні дані будуть перетворені до виду плоскою структури. Дана структура буде містити параметри, представлені в таблиці 2.2

Перетворення буде виконано наступним чином: для кожного з досліджуваних проектів будуть отримані вектори параметрів P1-Pn, описаних в таблиці 2.2. Дані вектори будуть впорядковані за датою створення. Для тих параметрів, які монотонно зростають, наприклад, число людино-місяців, крім абсолютних значень буде обчислена дельта змін, тобто різниця між поточним і попереднім значенням даного параметра.

В результаті для кожного з досліджуваних проектів буде отримана таблиця, структура якої представлена в таблиці 2.3.

Таблиця 2.2 – Параметри для подальшої обробки

Назва	Тип даних	Коментар
Code added	Integer	Кількість доданих рядків коду
Code removed	Integer	Кількість вилучених рядків коду
Comments added	Integer	Кількість доданих рядків коментарів
Comments removed	Integer	Кількість вилучених рядків коментарів

Продовження таблиці 2.2

Blanks added	Integer	Кількість доданих порожніх рядків
Blanks removed	Integer	Кількість віддалених порожніх рядків
Contributors	Integer	Число розробників
Commits	Integer	Число коммітов
Code lines	Integer	Число рядків коду
Comments	Integer	Число коментарів
Blanks	Integer	Число порожніх рядків
ManMonths	Integer	Число людино-місяців
ManMonthsDelta	Integer	Дельта числа людино-місяців
Total Code lines	Integer	Загальна кількість рядків коду
Name	String	Назва
Date	Date(ISO 8601)	Дата збору інформації

В таблиці 2.3 P_1-P_n – параметри, описані в таблиці 2.1; T_1-T_m дата збору інформації, N [i, j] – значення параметра.

Таблиця 2.3 – Представлення даних в плоскому вигляді

Дата	P1	...	Pn
T1	N [1,1]	...	N [1, n]
...	...	...	...
Tm	N [m, 1]	...	N [m, n]

2.5 Методика обробки результатів вимірювань

В результаті виконання експерименту будуть отримані чисельні значення набору метрик для кожного досліджуваного проекту. Зважаючи на велику

кількість отриманих даних за доцільне виглядає застосувати класичні методи статистики для їх обробки.

Для кожного набору значень по одній метриці отримаємо наступні статистичні характеристики:

- мінімальне значення метрики;
- максимальне значення метрики;
- математичне очікування – середнє значення випадкової величини:

$$x = \frac{1}{n}\sum_{i=1}^{n} x_i ;$$

- середньоквадратичне відхилення – показник розсіювання значень випадкової величини щодо її математичного очікування:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 .$$

Кореляція – статистичний взаємозв'язок двох або кількох випадкових величин (або величин, які можна з деякою допустимої ступенем точності вважати такими). Зміни значень однієї або декількох з цих величин супроводжують систематичного зміни значень іншої або інших величин. Математичної мірою кореляції двох випадкових величин служить коефіцієнт кореляції.

Деякі види коефіцієнтів кореляції можуть бути позитивними або негативними. У разі позитивного коефіцієнта кореляції передбачається, що можна оцінити лише сам факт наявності або відсутності взаємозв'язку випадкових величин, в разі негативного можна також визначити її напрямок. Збільшення значення першої випадково величини в такому випадку веде до зменшення значення другої. Відповідно, позитивне значення коефіцієнта кореляції в таких умовах вказує на такий зв'язок, при якій збільшення першої змінної пов'язано зі збільшенням і другий. Крім того, можливо ситуація, коли коефіцієнт кореляції

близький до нуля, – це говорить про відсутність статистичного взаємозв'язку між аналізованими випадковими величинами. Метод обчислення коефіцієнта кореляції залежить від виду шкали, до якої відносяться змінні. Так, для вимірювання змінних з інтервального і кількісної шкалами необхідно використовувати коефіцієнт кореляції Пірсона.

Коефіцієнт лінійної кореляції Пірсона – найбільш часто використовуваний коефіцієнт кореляції. Призначений для розрахунку сили і напряму лінійної залежності між змінними дослідження. Передбачається, що змінні виміряні в інтервального шкалою або в шкалі відносин.

Якщо досліджувані пари змінних розмістити на координатному полі, і значення змінних розглядати як координати точки, то коефіцієнт кореляції буде відображати середня відстань від точок до усередненої прямий: чим ближче точки з усередненою прямий, тим вище кореляція (див. рисунок 2.2).

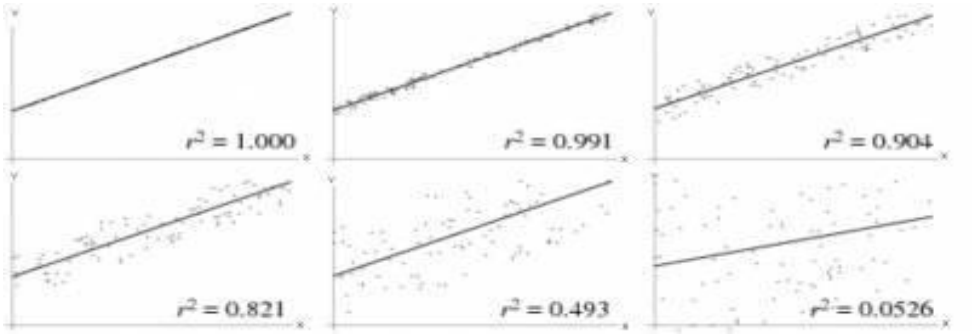

Рисунок 2.2 – Коефіцієнт кореляції Персона

Загальна формула для обчислення коефіцієнта лінійної кореляції Пірсона:

$$r_{xy} = \frac{\Sigma(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\Sigma(x_i - \bar{x})^2 \cdot \Sigma(y_i - \bar{y})^2}},$$

де x_i – значення, що приймаються змінною X,

y_i - значення, що приймаються зміною Y,

$\bar{x}$ – середня по X,

Розрахунок коефіцієнта кореляції Пірсона припускає, що змінні X і Y розподілені нормально.

3 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

3.1 Аналіз методів перевірки гіпотез

Статистична гіпотеза являє собою деяке припущення про закон розподілу випадкової величини або про параметри цього закону, що формулюється на основі вибірки. Прикладами статистичних гіпотез є припущення: генеральна сукупність розподілена за експоненціальним законом; математичні очікування двох експоненціально розподілених вибірок рівні один одному. У першій з них висловлено припущення про вид закону розподілу, а в другій – про параметрах двох розподілів. Гіпотези, в основі яких немає ніяких припущень про конкретний вид закону розподілу, називають непараметричних,в іншому випадку – параметричними.

Гіпотезу, яка стверджує, що різниця між порівнюваними характеристиками відсутня, а спостережувані відхилення пояснюються лише випадковими коливаннями в вибірках, на підставі яких проводиться порівняння, називають нульовою (основний) гіпотезою і позначають H_0. Поряд з основною гіпотезою розглядають і альтернативну (конкуруючу, що суперечить) їй гіпотезу H_1. І якщо нульова гіпотеза буде відкинута, то буде мати місце альтернативна гіпотеза.

Розрізняють прості і складні гіпотези. гіпотезуназивають простий,якщо вона однозначно характеризує параметр розподілу випадкової величини. Наприклад, якщо l є параметром експоненціального розподілу, то гіпотеза H_0 про рівність l =10-проста гіпотеза. Складною називають гіпотезу, яка складається з кінцевого або нескінченного безлічі простих гіпотез. Складна гіпотеза H_0 про нерівність l >10 складається з нескінченної кількості простих гіпотез H_0 про рівність $l = b_i$, де b_i – будь-яке число, більше 10. Гіпотеза H_0 про те, що математичне очікування нормального розподілу дорівнює двом при невідомій дисперсії , теж є складною. Складною гіпотезою буде припущення про розподіл випадкової величини X за нормальним законом, якщо не фіксуються конкретні значення математичного очікування і дисперсії.

Перевірка гіпотези грунтується на обчисленні деякої випадкової величини – критерію, точне або наближене розподіл якого відомо. Позначимо цю величину через z,її значення є функцією від елементів вибірки $z=z(x_1,x_2,..., x_n)$.Процедура перевірки гіпотези наказує кожному значенню критерію одне з двох рішень – прийняти або відкинути гіпотезу. Тим самим всі вибіркове простір і відповідно безліч значень критерію діляться на два непересічних підмножини S_0 і S_1.Якщо значення критерію z потрапляє в область S_0,то гіпотеза приймається, а якщо в область S_1,- гіпотеза відхиляється. Безліч S_0 називається областю прийняття гіпотези або областю допустимих значень, а безліч S_1 – областю відхилення гіпотези або критичною областю. Вибір однієї області однозначно визначає і іншу область.

Критерій Фішера (F-критерій) є параметричним критерієм і використовується для порівняння дисперсій двох варіаційних рядів. Емпіричне значення критерію обчислюється за формулою:

$$F = \frac{\sigma_1^2}{\sigma_2^2},$$

де σ_2^2 – велика і меньная дисперсія розглянутих варіаційних рядів.

Якщо обчислене значення критерію F більше критичного для певного рівня значущості і відповідних чисел ступенів свободи для чисельника і знаменника, то дисперсії вважаються різними.

Критичне значення критерію Фішера слід визначати за спеціальною таблицею, виходячи з рівня значущості α і ступенів свободи чисельника (n_1-1) і знаменника (n_2-1). Рівень значущості - ймовірність не прийняти гіпотезу за умови, що вона вірна. Як правило рівень значущості приймається рівним 0,05 або 0,01.

Для виконання даного дослідження необхідно ПЗ, що мають наступні функції:

– збір вихідних даних про проекти за допомогою періодичного опитування API;

− перетворення отриманих даних до придатного для подальшої обробки увазі;

− виконання необхідних розрахунків (виняток надлишкових параметрів, перевірка гіпотез).

Для виконання перших двох пунктів написана необхідна програма, розрахунок метрик буде виконаний за допомогою розрахунків.

Таким чином, визначені параметри експерименту. Фактором є інформація про історію змін вихідного коду різних Open Source проектів. Відгуками є набір числових характеристик, що описують історію роботи над проектом. Експеримент має пасивний характер.

Виконано планування обробки та інтерпретації отриманих даних. Для вирішення даного завдання будуть використані класичні методи статистичної обробки: знайдені мінімальні і максимальні показники метрик, математичне очікування, середньоквадратичне відхилення. Для кожної пари метрик буде розрахована кореляція, що дозволить зменшити розмірність простору. Після цього буде виконана перевірка гіпотез.

Визначено інструменти, необхідні для проведення дослідження.

3.2 Алгоритми методів обробки аналізу динаміки зміни репозиторіїв

Коефіцієнт кореляції Пірсона застосовується для дослідження взаємозв'язку двох змінних, виміряних в метричних шкалах на одній і тій же вибірці. Він дозволяє визначити, наскільки пропорційна мінливість двох змінних. Алгоритм зменшення розмірності простору застосовується для пошуку пар метрик, які сильно корелюють між собою з метою виключення надлишкових метрик. Юля визначення ступеня кореляції пар метрик між собою буде застосовуватися кореляція Пірсона

Коефіцієнт кореляції Пірсона застосовується для дослідження взаємозв'язку двох змінних, виміряних в метричних шкалах на одній і тій же вибірці. Він дозволяє визначити, наскільки пропорційна мінливість двох змінних.

Вхідними даними даного алгоритму є два векторів метрик: M_1 і M_2 однакової довжини. Оскільки коефіцієнт кореляції Пірсона нестійкий до викидів, перш ніж виконувати розрахунок необхідно виключити крайні значення для кожного з векторів, вхідних даних. Далі необхідно отримати довжину векторів, кількість ступенів свободи, а також середні значення для M_1 і M_2. Після чого, обчислити суму різниць, суму різниць квадратів і корінь з суми різниць квадратів.

Далі необхідно обчислити значення кореляції Пірсона. Якщо дане значення перевищує критичне значення для поточної кількості ступенів свободи, а також перевищує значення S, то можна зробити висновок про те, що між M_1 і M_2 існує суттєва кореляція і вектор M_2 можна виключити.

Розрахунок продуктивності розробника призначений для визначення середньої кількості рядків коду, коментарів, а також порожніх рядків, доданих або видалених одним розробником за один місяць.

Дана метрика розраховується за такою формулою:

$$\text{codeLines} = \frac{CodeAdd + CodeRem + ComAdd + ComRem + BlankAdd + BlankRem}{ManMonthDelta},$$

де CodeAdd – кількість доданих рядків коду,

CodeRem – кількість вилучених рядків коду,

ComAdd – кількість доданих коментарів,

ComRem – кількість віддалених коментарів,

BlankAdd – кількість доданих порожніх рядків,

BlankRem – кількість віддалених порожніх рядків,

ManMonthDelta – різниця між кількістю человекомесяцев в поточному і попередньому місяцях.

Розрахунок відсотка рядків коментарів – дана метрика вказує ступінь покриття вихідного коду коментарями і є важливим показником самодокументіруемості коду. Дана метрика розраховується за такою формулою:

$$Comments = \frac{CommentLines}{CodeLines + CommentLines + BlankLines},$$

де CommentLines – загальна кількість рядків коментарів в проекті,

CodeLines – загальна кількість рядків вихідного коду,

BlankLines – загальна кількість порожніх рядків.

Код, який містить менше 10% коментарів, вважається слабо документованим, що може викликати труднощі при подальшій роботі з ним. З іншого боку, код, що містить більше 40% коментарів, вважається перенасиченим ними, тому що велика кількість коментарів може перешкодити прочитання власне команд мови.

На рисунку 3.1 зображена схема алгоритму збору даних про історію роботи над проектом.

Вхідним параметром даного алгоритму є ID проекту. На першому кроці проводиться запит до API сховища Ohloh.net для отримання даних про історію роботи над проектом. Після чого перевіряється статус відповіді. Якщо запит виконаний успішно (статус 200), то з отриманих даних витягуються дані про метриках розміру, складності, а також дата обчислення метрик. Ці дані перетворюються в формат JSON і записуються в БД. У разі якщо запит виконаний невдало, виводиться повідомлення про помилку.

Вхідними даними для даного алгоритму є список метрик, що характеризує роботу над проектом.

На першому кроці в результуючий файл записуються заголовки стовпців. Після чого, в циклі з БД вичітаваются метрики проекту.

Дані з кожною прочитаною метрики перетворюються в формат CSV, після чого отриманий рядок записується в файл результату.

Цикл повторюється до тих пір, поки не будуть прочитані всі елементи.

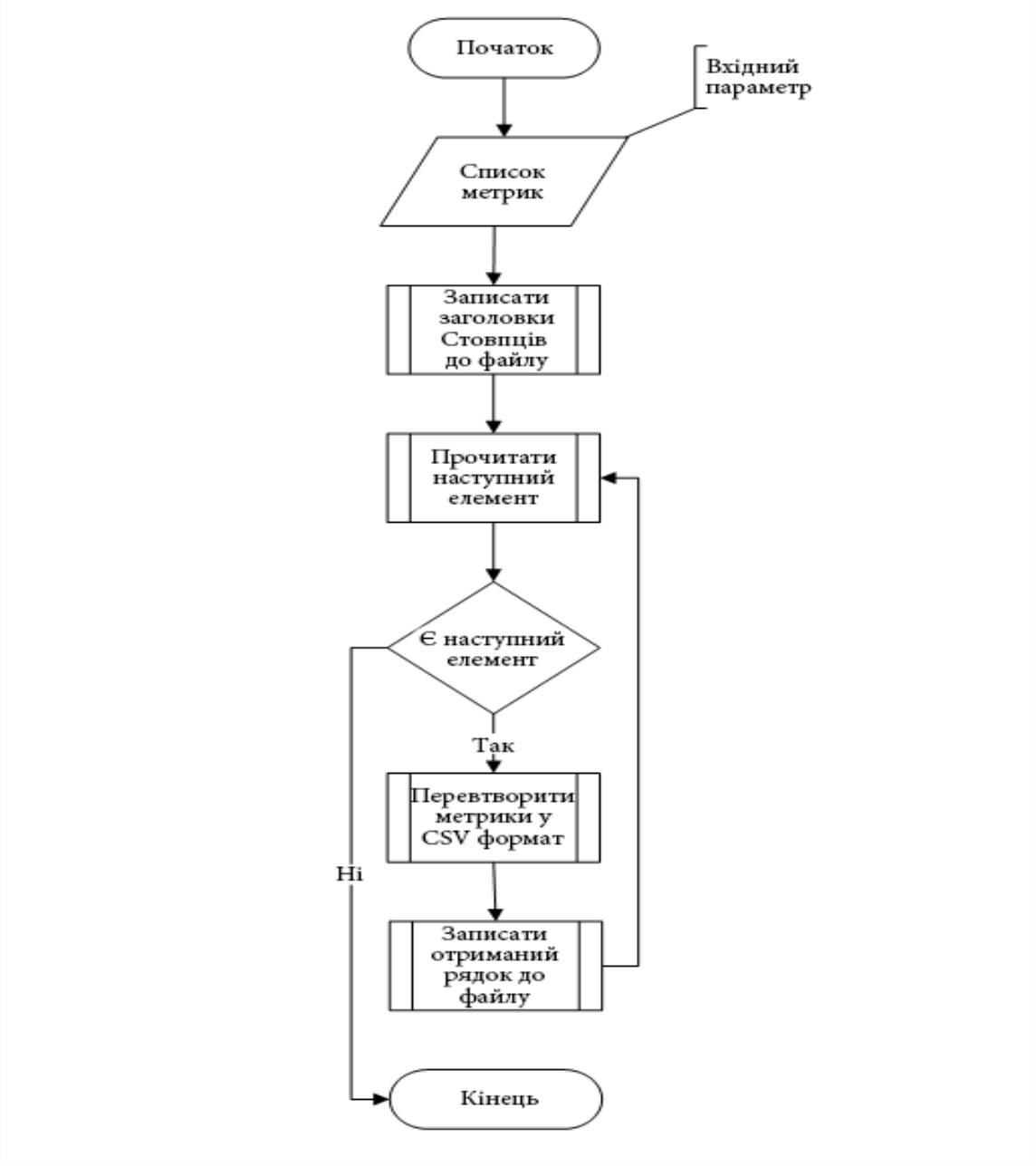

Рисунок 3.1 – Алгоритм збору даних про проект

3.3 Алгорити обчислення параметрів законів розподілу метрик класів

Після того, як вихідні дані були зібрані, їх необхідно перетворити до плоского виду для подальшої обробки. Алгоритм перетворення представлений на рис. 3.2.

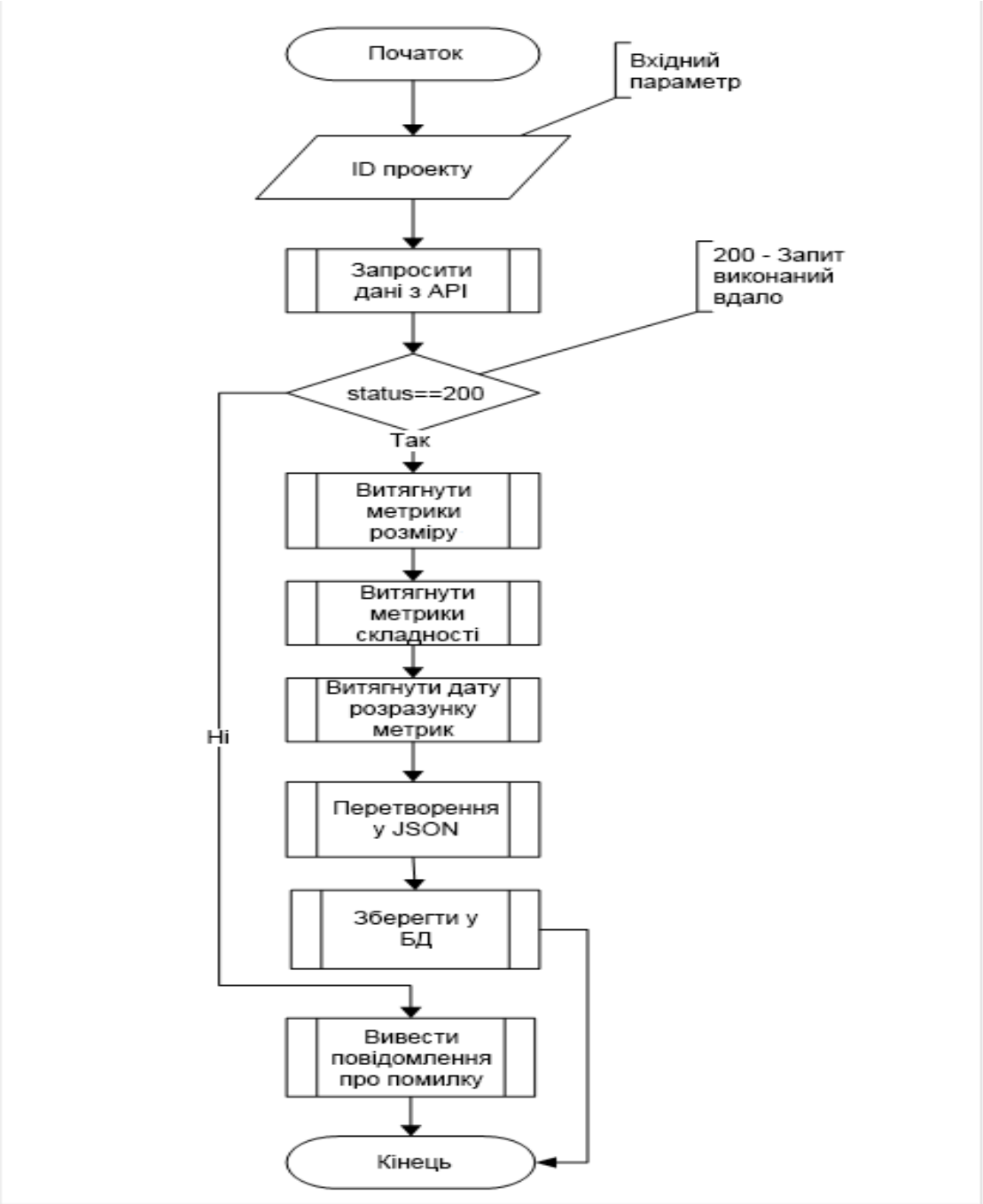

Рисунок 3.2 – Перетворення даних до плоского виду

В даному розділі буде розглянуто характер розподілу вимірюваних метрик. Як проект для розгляду буде використаний Apache Subversion та розглянуті метрики, що характеризують додавання нових рядків у проект, це:

- кількість доданих рядків коду;
- кількість доданих рядків коментарів;
- кількість доданих порожніх рядків.

Перша метрика – кількість доданих рядків коду. Графік зміни цієї метрики наведено на рис. 3.3

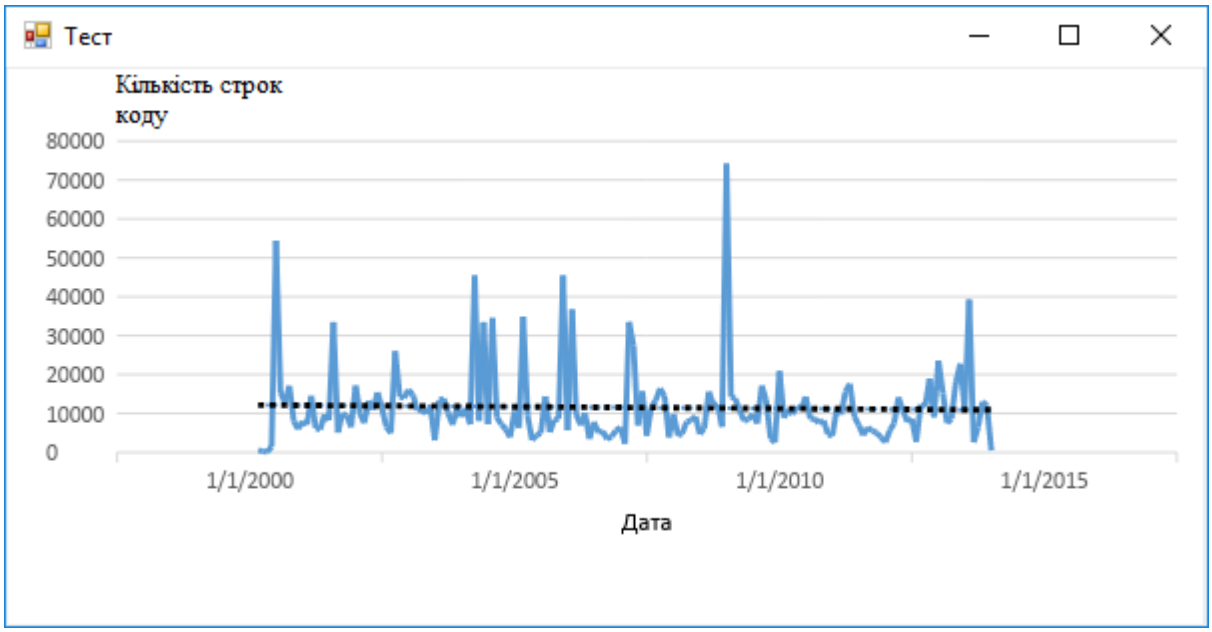

Рисунок 3.3 – Кількість доданих рядків коду

З графіка випливає, що за винятком окремих піків, кількість доданих рядків коду лежить в невеликих межах протягом усього часу розвитку проекту.

Можна припустити, що додавання значної кількості нових рядків вихідного коду пов'язане з реалізацією нової функціональності, а більшу частину часу в проект вносилися тільки незначні правки та модифікації.

Графік зміни метрики «кількість доданих коментарів» наведена на рис. 3.4.

З графіка можна зроби висновок, що протягом більшої частини часу роботи над проектом кількість доданих коментарів залишається стабільноюм.

Можна припустити, що окремі піки пов'язані з додаванням значної кількості вихідного коду.

Графік зміни метрики «кількість доданих порожніх рядків» приведена на рис. 3.5.

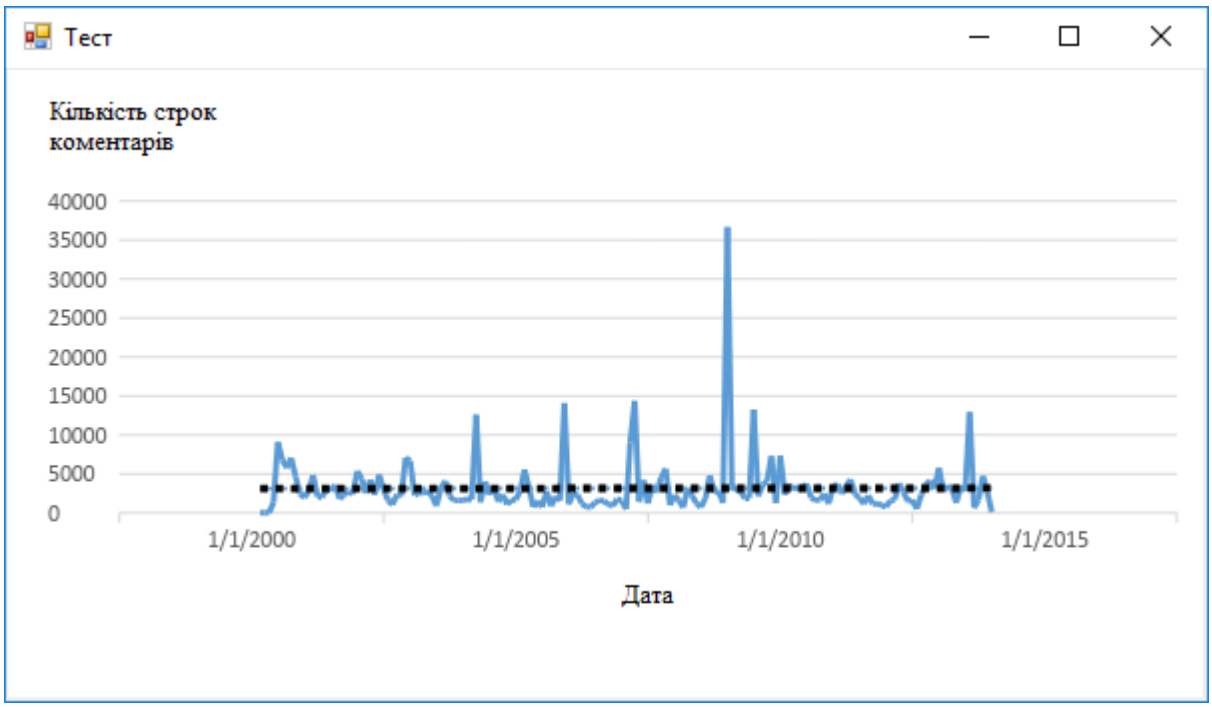

Рисунок 3.4 – Кількість доданих коментарів

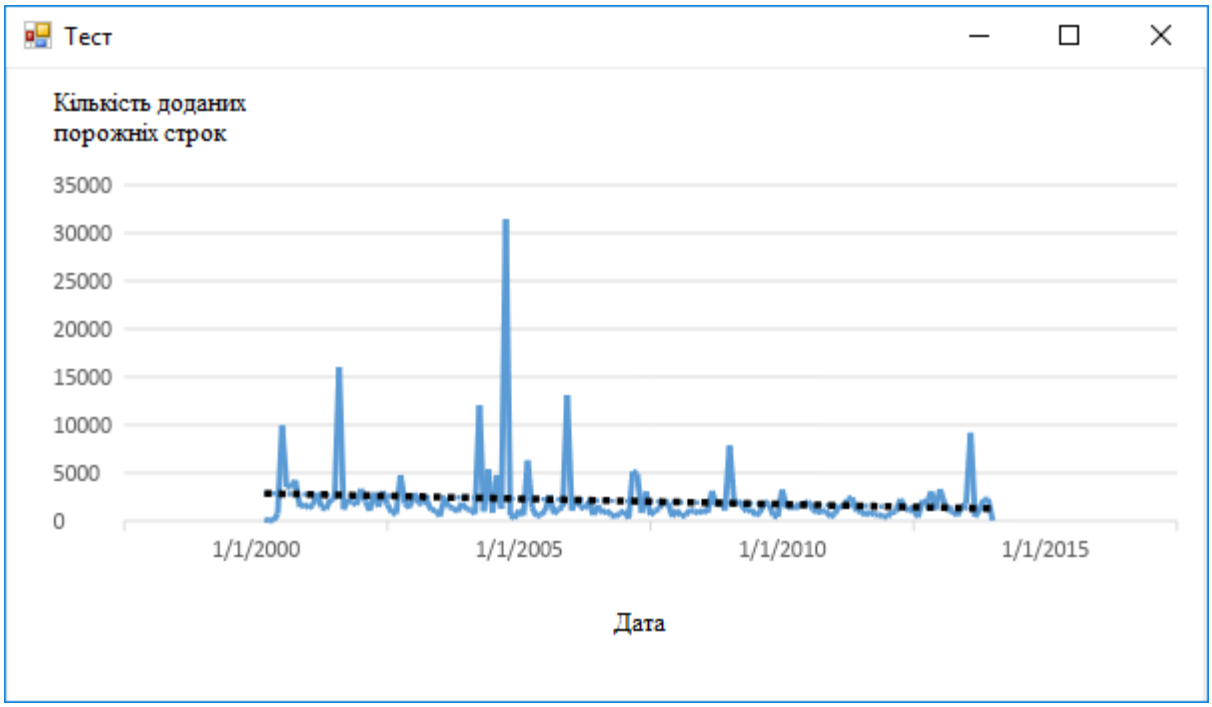

Рисунок 3.5 – Кількість доданих порожніх рядків

З графіка виливає, що кількість доданих порожніх рядків повільно знижується з часом. Можна припустити, що окремі піки пов'язані з додаванням нових функцій в проект.

4 ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Опис утіліти оцінювання метрік

Для виконання даного дослідження необхідно ПЗ, що має наступні функції:

− збір вихідних даних про проекти за допомогою періодичного опитування API;

− перетворення отриманих даних до придатного для подальшої обробки увазі;

− виконання необхідних розрахунків (виняток надлишкових параметрів, перевірка гіпотез).

Для виконання перших двох пунктів написана необхідна програма, а також проведений розрахунок метрик.

Таким чином, визначені параметри експерименту. Фактором інформації про історію змін вихідного коду різних Open Source проектів і висновками є набір числових характеристик, що описують історію роботи над проектом. Експеримент має пасивний характер.

Виконано планування обробки та інтерпретації отриманих даних. Для вирішення даного завдання будуть використані класичні методи статистичної обробки: знайдені мінімальні і максимальні показники метрик, математичне очікування, середньоквадратичне відхилення. Для кожної пари метрик буде розрахована кореляція, що дозволить зменшити розмірність простору. Після цього буде виконана перевірка гіпотез.

Визначено інструменти, необхідні для проведення дослідження.

В подальшому необхідно розглянути метрики, що характеризують видалення рядків з проекту, це:

− кількість вилучених рядків коду;

− кількість вилучених рядків коментарів;

− кількість вилучених порожніх рядків.

Графік зміни метрика «кількість вилучених рядків коду» показана на рисунку 4.1

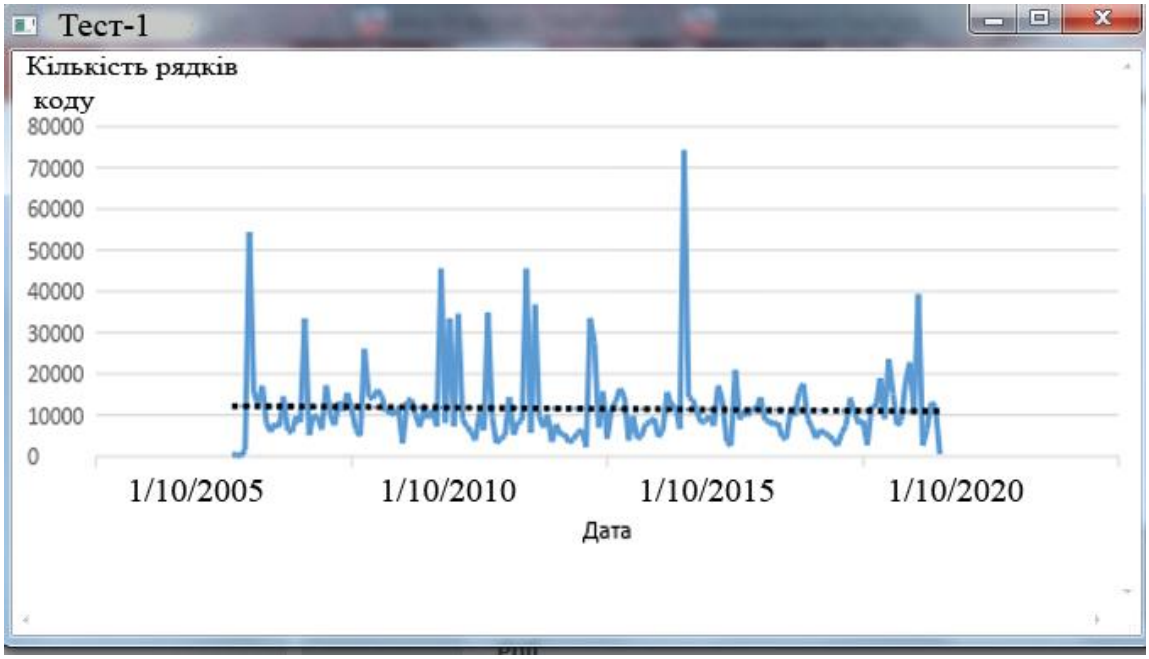

Рисунок 4.1 – Кількість вилучених рядків коду

З графіка витікає, що кількість вилучених рядків коду, за винятком окремих піків, залишається стабільних протягом усього часу роботи на проектом. Ймовірно, видалення великої кількості рядків коду пов'язано з проведенням рефакторінга або видаленням застарілої функціональності.

Графік зміни метрики «кількість вилучених рядків коментарів» показана на рис. 4.2. З графіка випливає, що кількість вилучених рядків коментарів залишається стабільним протягом більшої частини часу роботи над проектом. Можна припустити, що окремі піки відповідають часу проведення рефакторінгу і видалення старої функціональності.

Графік зміни метрики «кількість віддалених порожніх рядків» приведена на рисунку 4.3.

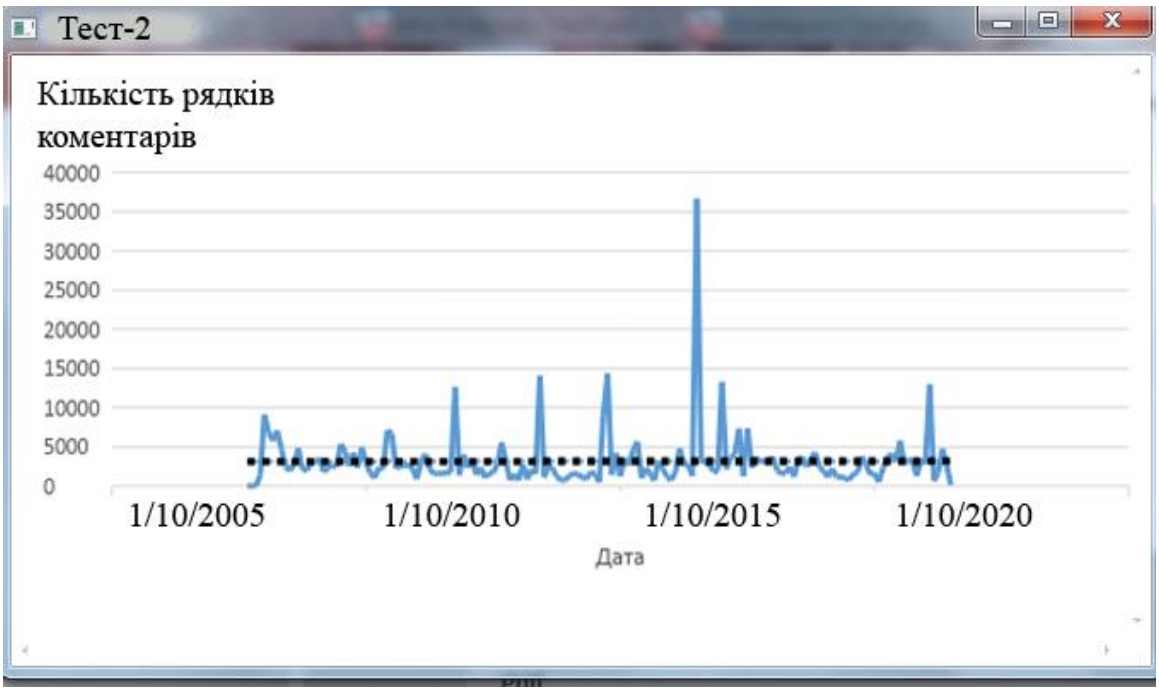

Рисунок 4.2 – Кількість вилучених рядків коментарів

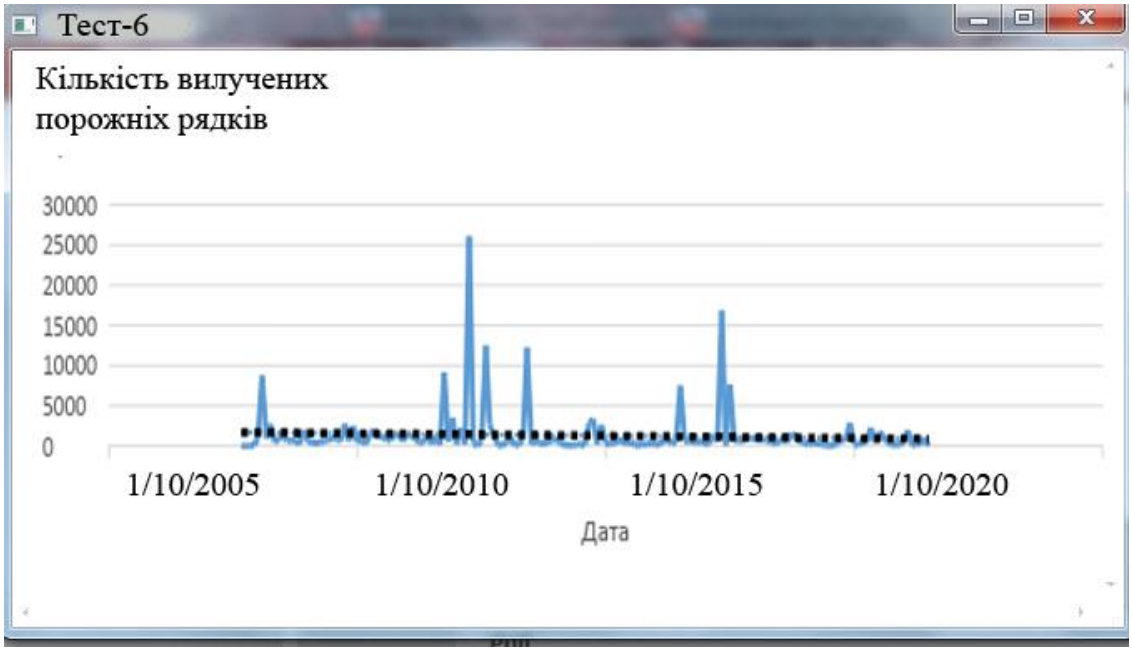

Рисунок 4.3 – Кількість вилучених порожніх рядків

З графіка слід що кількість віддалених порожніх рядків залишається вкрай низьким протягом усього часу роботи над проектом, за винятком окремих піків, які ймовірно пов'язані з видаленням застарілої функціональності і рефакторінгом.

Обчисленя метрики «Зміна кількості людино-місяців» характеризує те, наскільки зростала складність проекту, виражена в человекомесяцах за минулий місяць. Графік зміни даної метрики приведена на рисунку 4.4.

Рисунок 4.4 – Дельта людино-місяців

На даному графіку можна виділити три етапи розробки проекту. На початковому етапі розробки зростання складності проекту лінійно зростає. Після чого настає етап, коли зростання складності стабілізується. Можна припустити, що саме на цьому етапі відбувається розробка основної функціональності проекту. Після чого настає етап, на якому зростання складності сповільнюється.

Дана метрика має поліноміальний характер зміни.

Метрика «Зміна продуктивності розробника» відображає зміну середньої продуктивності розробника, що працює на проекті. Під середньою продуктивністю розуміється результат ділення кількості, змінених термін на кількість людино-місяців за минулий місяць роботи.

Графік зміни даної метрики наведено на рис. 4.5.

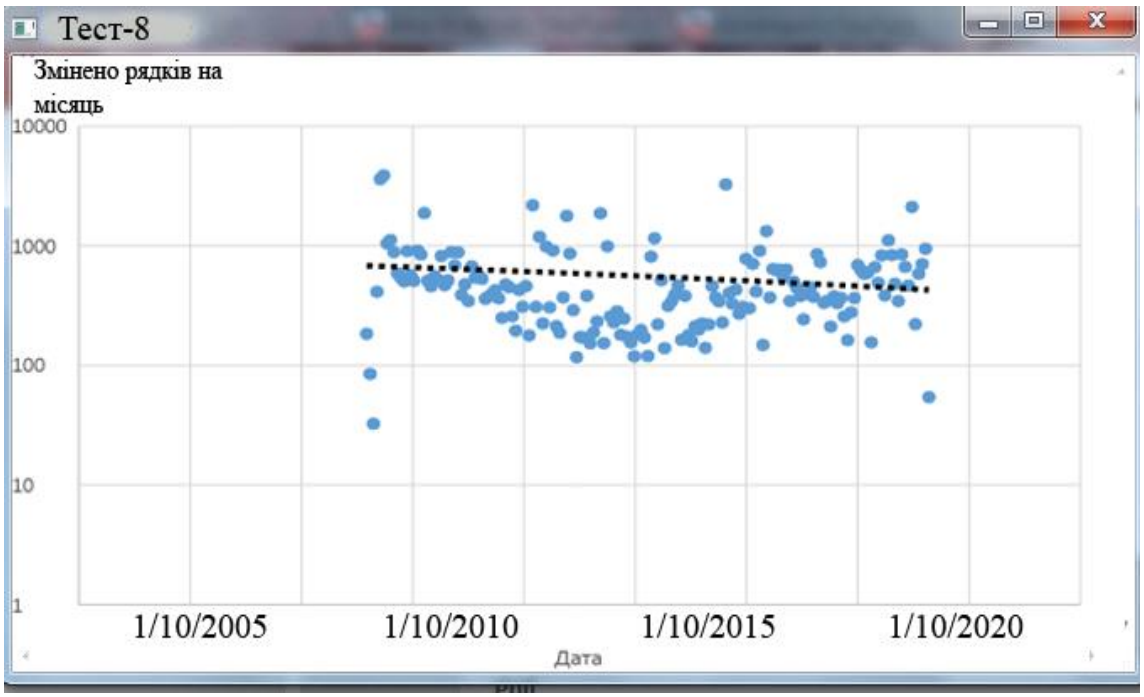

Рисунок 4.5 – Зміна продуктивності розробника

З даного графіка випливає, що продуктивності розробника повільно зменшується з ростом проекту. Можна припустити, що це пов'язано зі збільшенням складності проекту і необхідністю підтримки існуючого коду.

Метрика «Відсоток рядків коментарів» відображає рівень документованості коду. Проект вважається досить документованим, якщо містить від 20% до 40% рядків коментарів. Графік зміни даної метрики показаний на рис. 4.6.

Рисунок 4.6 – Відсоток рядків коментарів

Як випливає з графіка, на початку розробки проекту відсоток коментарів зростає дуже швидко, однак після стабілізується. Також можна відзначити скачки відсотка коментарів, які ймовірно викликані необхідністю поліпшити рівень документованості коду і спростити підтримку проекту.

4.2 Аналіз кореляції між метриками проектів

Проведений аналіз кореляції між метриками, що описано вище. Для обчислення кореляції буде використовуватися коефіцієнт кореляції Пірсона.

Як проектів для розгляду будуть використовуватися такі проекти:

- Apache Subversion;
- Ruby;
- PostgreSQL Database Server;
- Firefox.

Результати розрахунку кореляції наведені в таблицях 4.1 – 4.4.

Таблиця 4.1 – Кореляція між метриками для проекту

	odeLines Added	omments Added	Blanks Added	deLines Removed	mments Removed	Blanks Removed	ManMonth Delta	% Comments	eloper Productivity
CodeLines Added	1	0.60327	0.7844	0.63702	0.55185	0.29235	0.21789	-0.0497	0.762904
Comments Added		1	0.66789	0.75367	0.27013	0.62445	0.21819	0.04937	0.697933
Blanks Added			1	0.89344	0.40358	0.37909	0.16282	0.04465	0.699963
CodeLines Removed				1	0.40062	0.49958	0.17719	0.00595	0.638644

Продовження таблиці 4.1

Comments Removed					1	0.69637	0.14089	-0.0952	0.536144
Blanks Removed						1	0.20703	-0.0057	0.482466
ManMonth Delta							1	-0.1356	-0.1416
% Comments								1	0.068649
Developer Productivity									1

Таблиця 4.2 – Кореляція між метриками для проекту

	CodeLines Added	Comments Added	Blanks Added	CodeLines Removed	Comments Removed	Blanks Removed	ManMonth Delta	% Comments	Developer Productivity
CodeLines Added	1	0.654534	0.6985	0.50556	0.6872	0.6013	0.4426	0.23552	0.68372
Comments Added		1	0.5182	0.56547	0.33874	0.86381	0.38558	0.20482	0.18228
Blanks Added			1	0.66302	0.67807	0.40579	0.49134	0.34522	0.124447
CodeLines Removed				1	0.48673	0.75753	0.59256	0.35897	0.10189
Comments Removed					1	0.45770	0.39351	0.23268	0.232021
Blanks Removed						1	0.40023	0.20868	0.130639
ManMonth Delta							1	0.67116	-0.06952
% Comments								1	-0.103

Таблиця 4.3 – Кореляція між метриками для проекту

	CodeLines	Comments	Blanks Added	CodeLines	Comments	Blanks	ManMonth	% Comments	Developer
CodeLines Added		0.816324	0.91099	0.538397	0.967222	0.729602	0.573548	0.169974	0.50336
Comments Added		1	0.61685	0.618725	0.724734	0.941464	0.530607	0.159372	0.454077
Blanks Added			1	0.497609	0.955565	0.591038	0.492328	0.256622	0.408793
CodeLines Removed				1	0.469357	0.582481	0.392108	0.108156	0.266763
Comments Removed					1	0.691631	0.53034	0.181462	0.382965
Blanks Removed						1	0.448731	0.187644	0.546805
ManMonth Delta							1	0.114596	-0.10819
% Comments								1	0.275923
Developer Productivity									1

Осередки з високим показником кореляції, характерним для всіх розглянутих проектів, та який вказує на наявність сильного взаємозв'язку між парами метрик. Розглянемо такі пари метрик докладніше і проаналізуємо причини їх високої кореляції.

Кількість доданих рядків коду, кількість доданих порожніх рядків і рядків коментарів. Дані метрики тісто пов'язані між собою. Додавання рядків вихідного коду неминуче тягне за собою додавання певної кількості порожніх рядків.

Таблиця 4.4 – Кореляція між метриками для проекту PostgreSQL

	nes Added	ents Added	Added	nes Removed	ents Removed	Removed	onth Delta	ments	per Productivity
CodeLines Added		5482	409	0.41197	0.74532	0.20858	0.069741	4045	0.850923
Comments Added			009	0.58973	0.13382	0.61779	0.210139	1968	0.409379
Blanks Added				0.48844	0.84737	0.11658	0.058772	1607	0.868518
CodeLines Removed					0.12492	0.47631	0.286076	7028	0.308067
Comments Removed					1	0.159 6	-0.03103	0.305 977	0.874252
Blanks Removed						1	0.238647	0.101 636	0.210592
ManMonth Delta							1	0.114 596	-0.10819
% Comments								1	0.275923
Developer Productivity									1

Зв'язок з кількістю коментарів слабкіше, але також простежується. Ступінь покриття коду коментарями залежить він культури написання коду, а також, від стандартів прийнятих на проекту.

Кількість доданих рядків коду і продуктивність розробника. Ці метрики тісно пов'язані між собою. Збільшення кількості доданих рядків коду тягне за собою збільшення показника середньої продуктивності. Кількість доданих коментарів і кількість доданих порожніх рядків також досить сильно корелює зі збільшенням продуктивності, з аналогічних причин.

Кількість доданих порожніх рядків, кількість доданих коментарів і кількість вилучених рядків коду. Можна припустити, що висока залежність між даними метриками обумовлена тим, що в процесі рефакторінга або видалення застарілої функціональності непотрібний код замінюється порожніми рядками або коментується.

4.3 Реалізація алгоритму перевірки гіпотез

Для виконання завдань роботи необхідно розглянути ряд гіпотез про характер зміни різних метрик роботи над проектом.

Будуть розглянуті такі гіпотези:

- зростання документованості від залежно від розміру проекту;
- зростання середньої продуктивності в залежності від розміру проекту;
- активність роботи над проектом з плином часу.

Можна припустити, що з ростом кількості вихідного коду проекту має також до певного рівня зростати і кількість коментарів. Оптимальним вважається кількість коментарів від 20% до 40% по відношенню до загального обсягу проекту. Проект, що містить менше 10% коментарів є слабо документованим і погано піддається підтримки. Характер залежності кількості коментарів від розміру проекту для проекту Apache Subversion наведено на рис. 4.7.

Характер залежності, рівняння прямої та значення R^2 для всіх розглянутих проектів наведені в табл.. 4.5.

Таблиця 4.5 – Результат перевірки гіпотези

Назва проекту	Тип залежності	Рівняння	R^2
Apache Subversion	лінійна	y = 3E-07x + 0.0178	0.7765
Ruby	лінійна	y = 1E-07x + 0.0193	0.6438

Продовження таблиці 4.5

PostgreSQL	лінійна	y = 3E-07x + 0.0323	0.7077
Firefox	лінійна	y = 4E-09x + 0.1435	0.3905

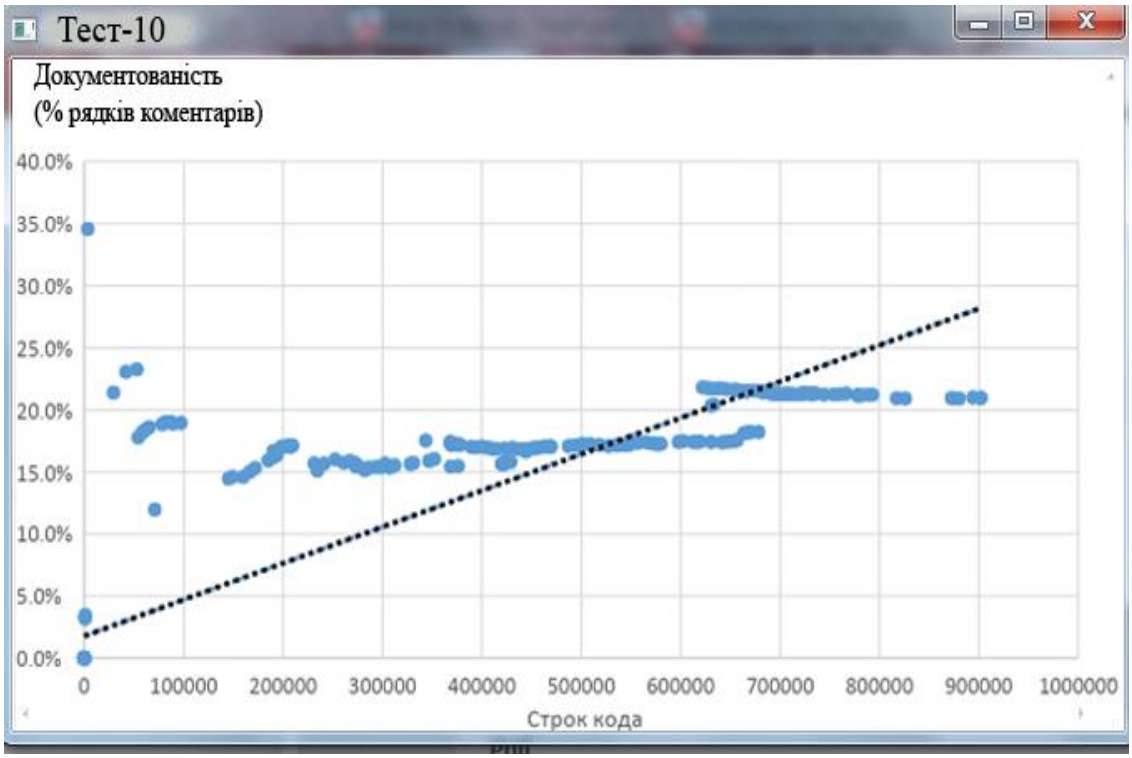

Рисунок 4.7 – Залежність кількості коментарів від розміру проекту

Можна припустити, що продуктивність розробника повинна збільшуватися в міру роботи над проектом. Цьому сприяє збільшення кількості вихідного коду, який можна повторно використовувати, поступове поліпшення розуміння проекту, а також збільшення кількості коментарів, розглянуте вище.

Графік залежності продуктивності від розміру проекту для проекту Apache Subversion наведено на рисунку 4.8.

Характер залежності, рівняння прямої та значення R^2 для всіх розглянутих проектів наведено в таблиці 4.6.

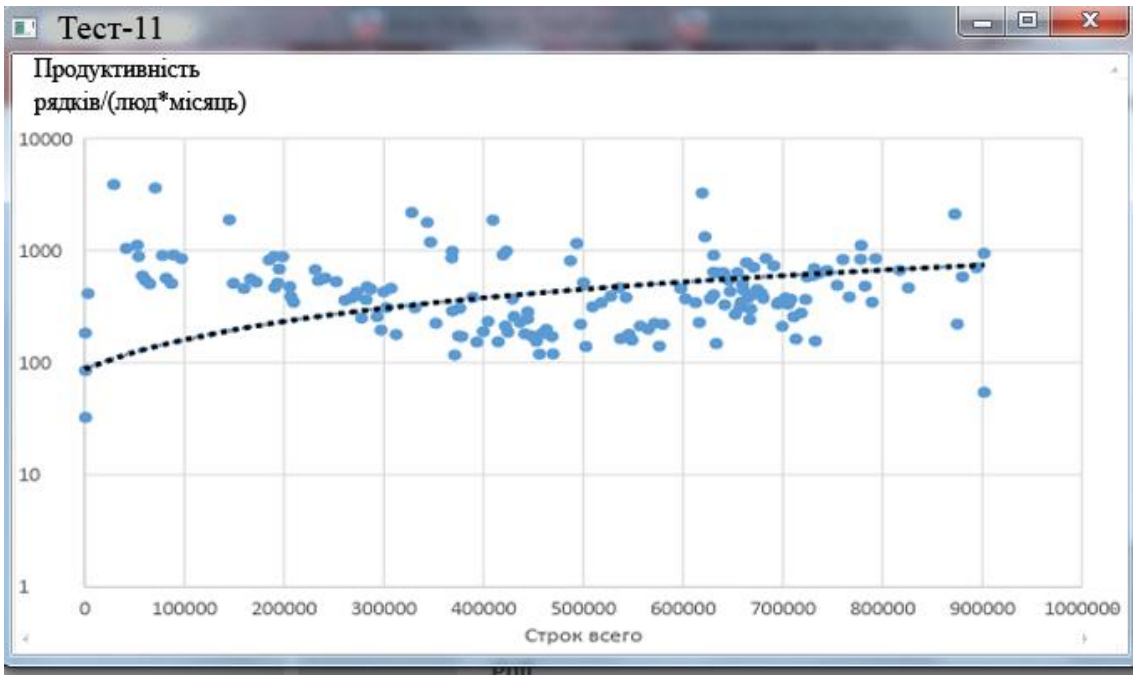

Рисунок 4.8 – Залежність продуктивності від розміру проекту

Таблиця 4.6 -Результат перевірки гіпотези

Назва проекту	Тип залежності	Рівняння	R^2
Apache Subversion	лінійна	y = 0.0007x + 87.414	0.2016
Ruby	лінійна	y = 0.0004x + 261.22	0.3019
PostgreSQL	лінійна	y = 0.0007x + 380.06	0.1458
Firefox	лінійна	y = 0.0011x + 2139.1	0.1405

Результати аналізу дозволяють зробити наступні загальні висновки про історію досліджуваних проектів.

Метрика «Кількість доданих рядків коду» сильно корелює з метриками «кількість доданих порожніх рядків» і «активність розробника»

Метрики «кількість вилучених рядків коду» сильно корелює з метриками «кількість доданих коментарів» і «кількість порожніх доданих рядків»

5 ОПИС МОЖЛИВОСТІ ВИКОРИСТАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

5.1 Алгоритм розрахунку активності роботи над проектом з плином часу

На основі раніше розглянутих даних можна зробити висновки про те, що історія роботи над проектом повинна включати в себе фазу швидкого зростання, коли проекту активно розвивається, фазу більш стабільного зростання, коли реалізується значна частина функцій проекту і фазу спаду активності, коли активність роботи над проектом зменшується і зводиться до виправлення помилок і реалізації дрібних функцій.

Графік залежності активності роботи над проектом від часу для проекту Apache Subversion наведено на рис. 5.1.

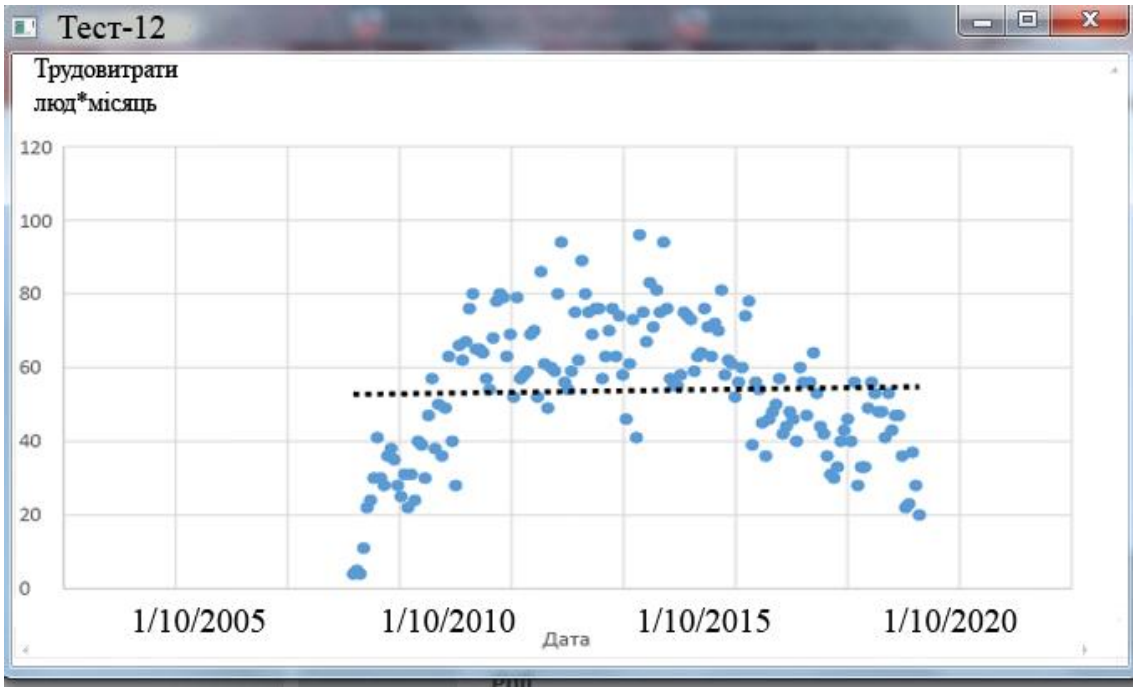

Рисунок 5.1 – Залежність активності роботи над проектом від часу

Характер залежності, рівняння прямої та значення R^2 для всіх розглянутих проектів наведено на рис.5.2.

Результат перевірки гіпотези

Назва проекту	Тип залежності	Рівняння	R^2
Apache Subversion	поліноміальна	$y = -8E\text{-}06x^2 + 0.5895x - 11467$	0.6035
Ruby	поліноміальна	$y = -5E\text{-}07x^2 + 0.051x - 1122$	0.7310
PostgreSQL	поліноміальна	$y = -3E\text{-}07x^2 + 0.0229x - 439.21$	0.1385
Firefox	поліноміальна	$y = 3E\text{-}05x^2 - 2.2151x + 41126$	0.6016

Рисунок 5.2 – Результат перевірки гіпотези

Кількість коментарів зростає з ростом проекту, до моменту досягнення коефіцієнта покриття коду коментарями до рівня близько 20%, після чого зростання кількості коментарів припиняється

5.2 Аналіз метрік проектів Open Source

Метрики роботи над проектом «FIREFOX».

Залежність трудовитрат від часу показана на рис. 5.3

Продуктивність розробника повільно зростає т зростанням проекту.

В історії роботи над проектами можна виділити три етапи: етап активного розвитку, етап стабільного зростання і етап загасання.

Залежність відсотка коментарів від кількості рядків коду показана на рисунку 5.4

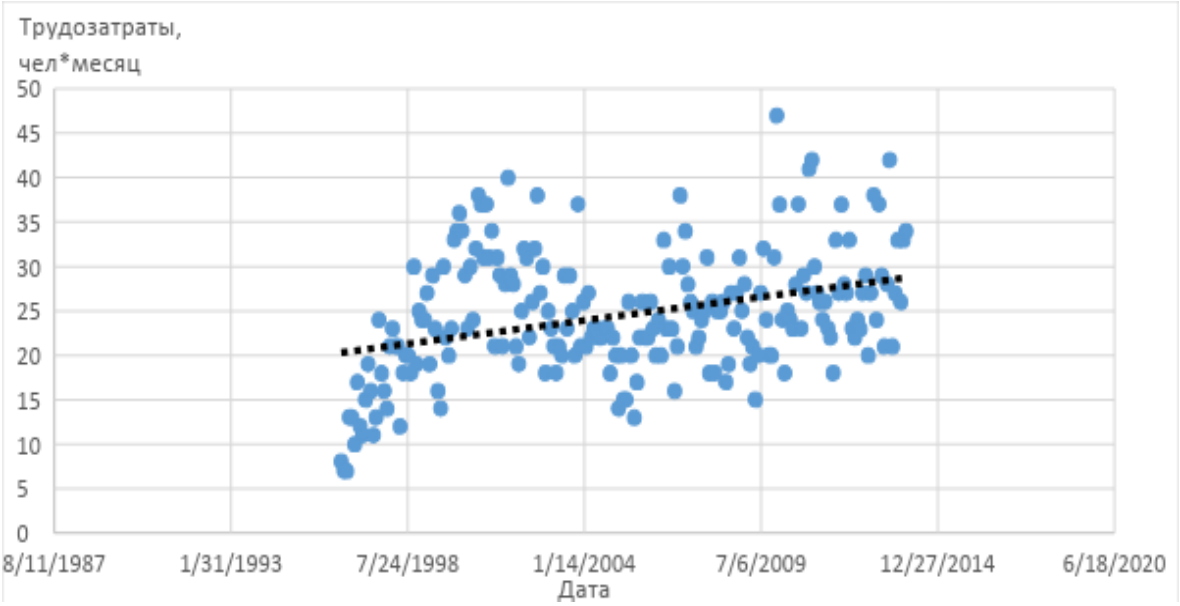

Рисунок 5.3 – Залежність працезатрат від часу

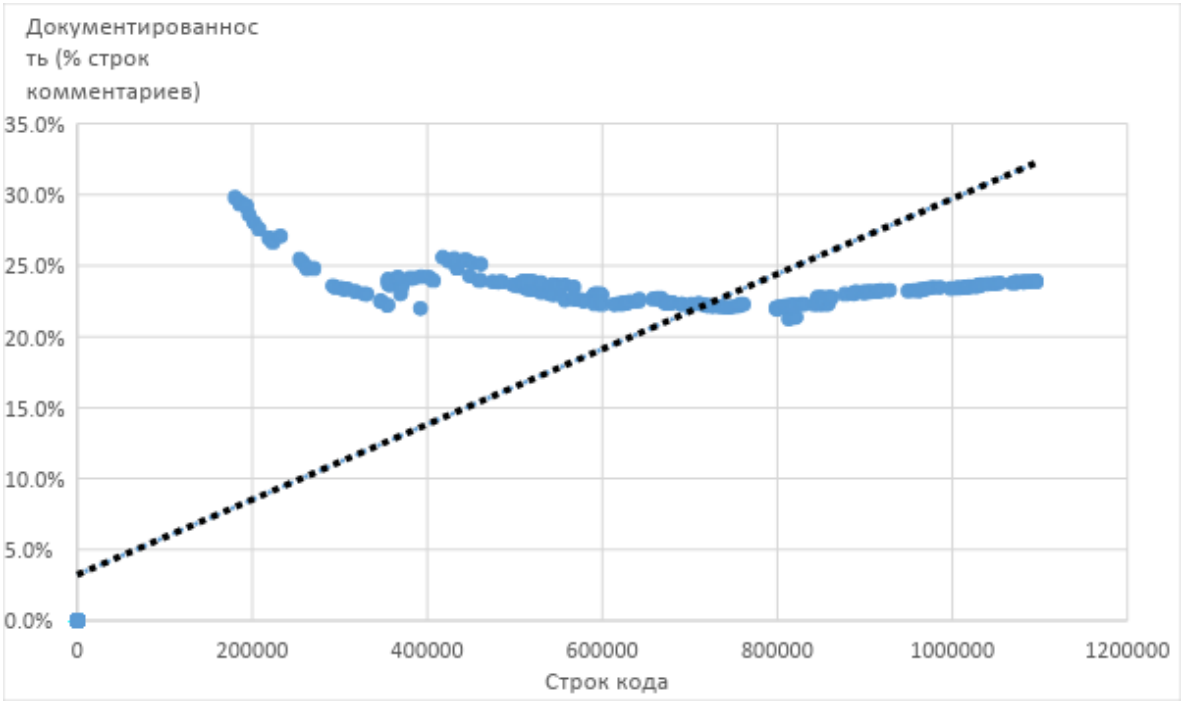

Рисунок 5 .4 – Залежність відсотка коментарів від кількості рядків коду

Залежність продуктивності від розміру проекту показана на рисунку 5.5

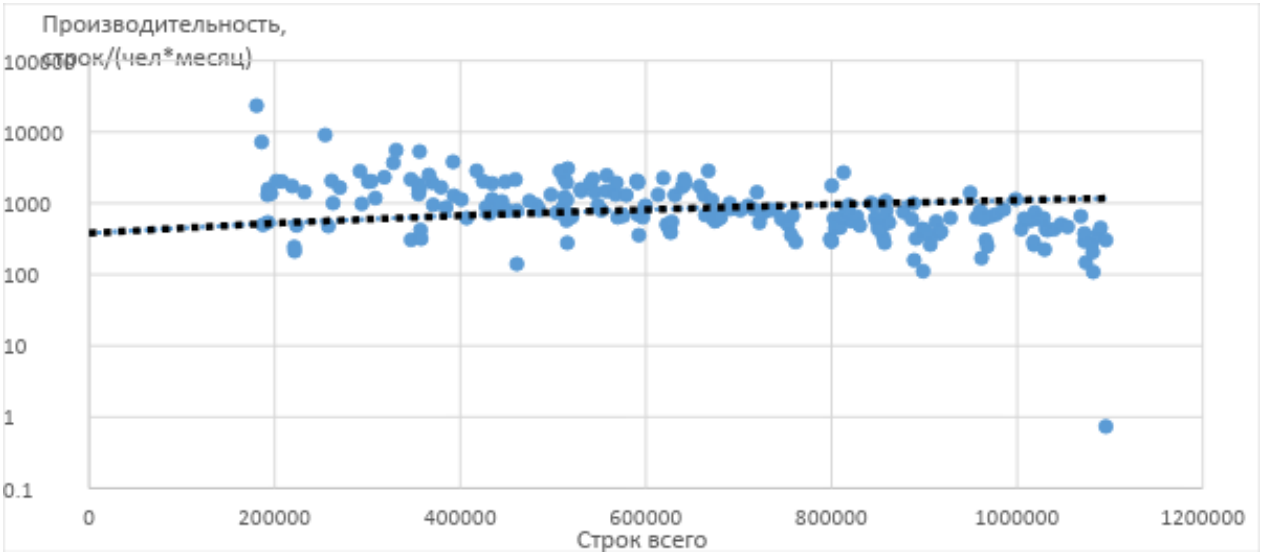

Рисунок 5.5 – Залежність продуктивності від розміру проекту

Метрікі роботи над проектом RUBY

Залежність трудовитрат від часу показана на рисунку 5.6

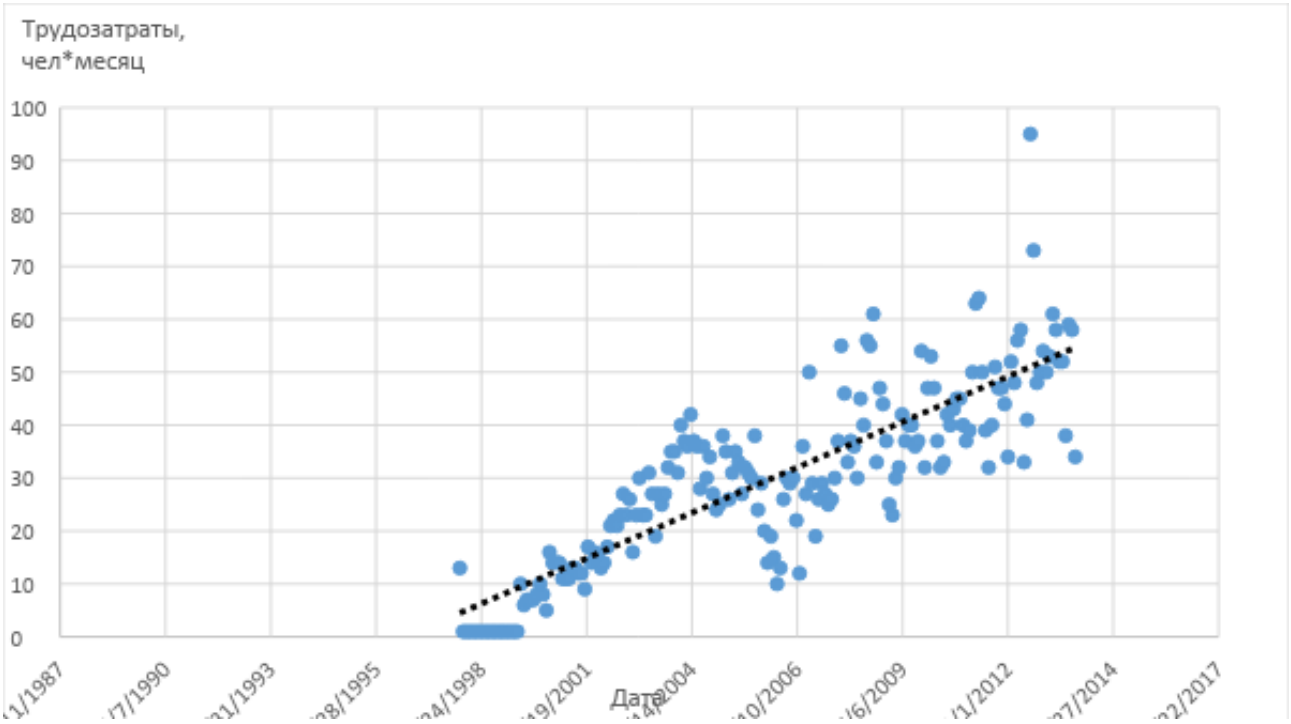

Рисунок 5.6 – Залежність трудовитрат часу

Залежність відсотка коментарів від кількості рядків коду показана на рисунку 5.7

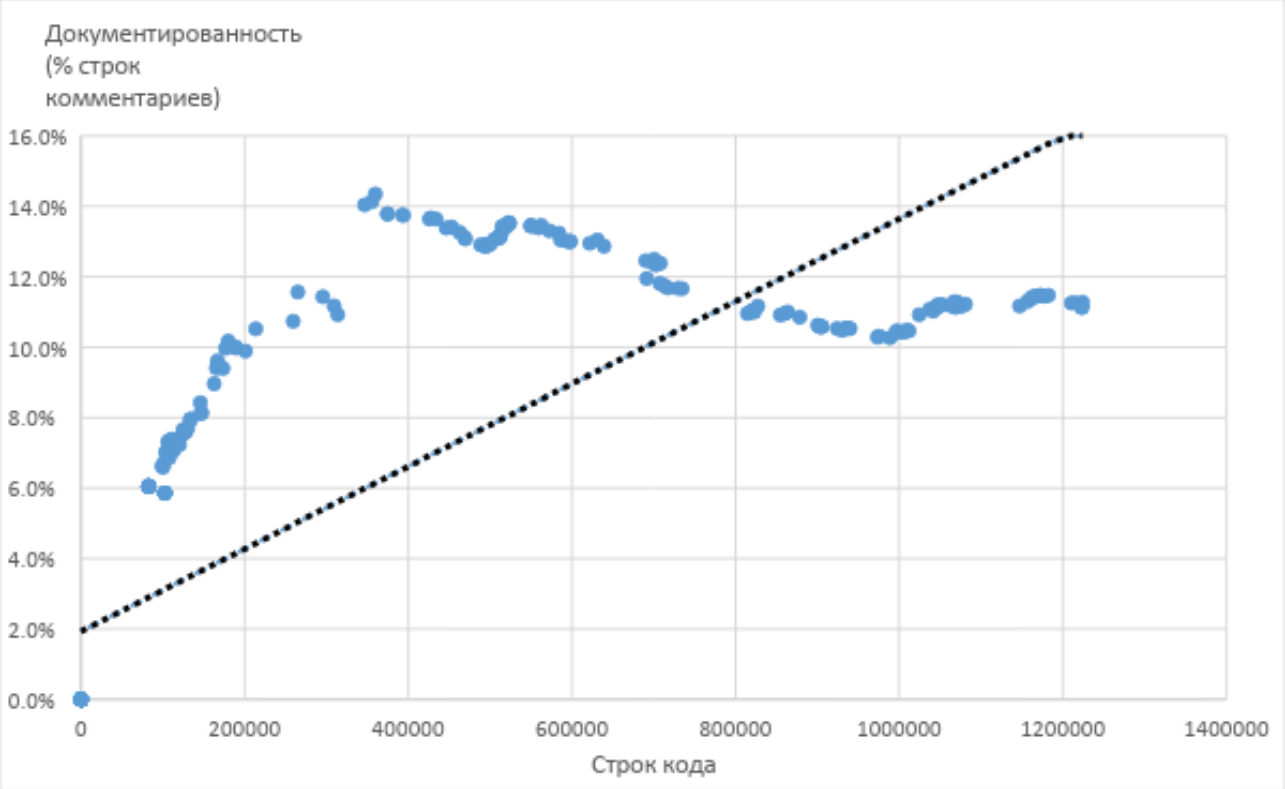

Рисунок 5.7 – Залежність відсотка коментарів від кількості рядків коду

Залежність продуктивності від розміру проекту показана на рисунку 5.8

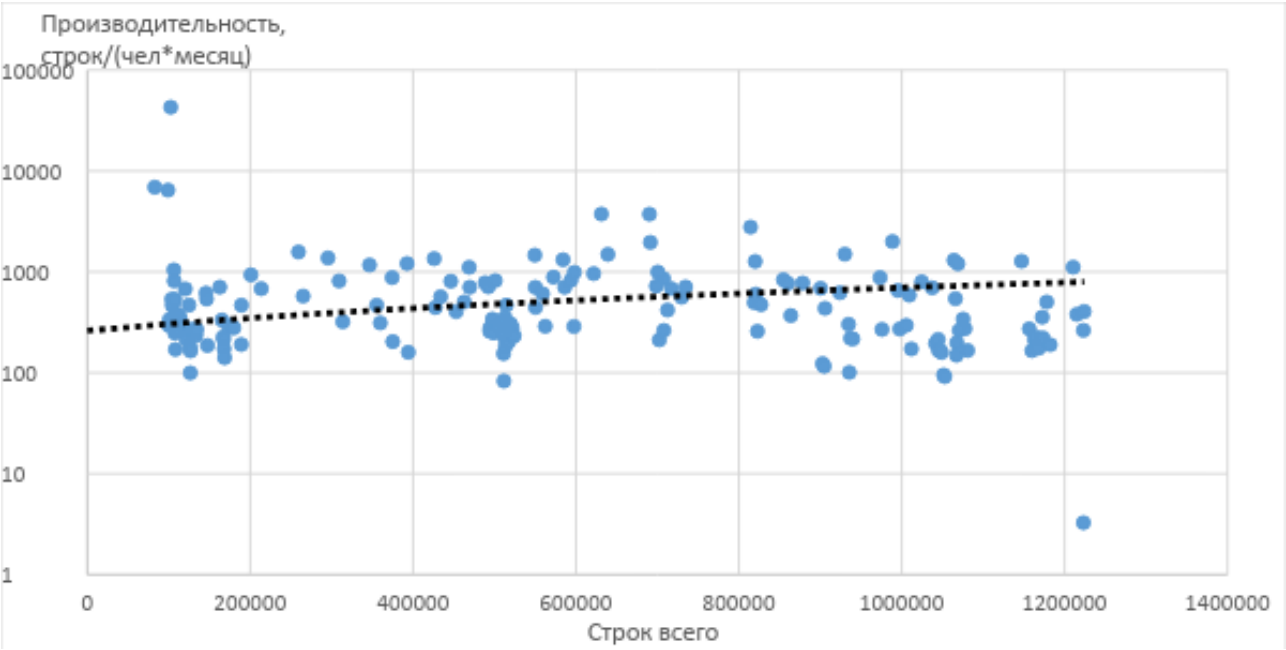

Рисунок 5.8 – Залежність продуктивності від розміру проекту

Метрики роботи над проектом Postgresql Database Server

Залежність трудовитрат від часу показана на рисунку 5.9

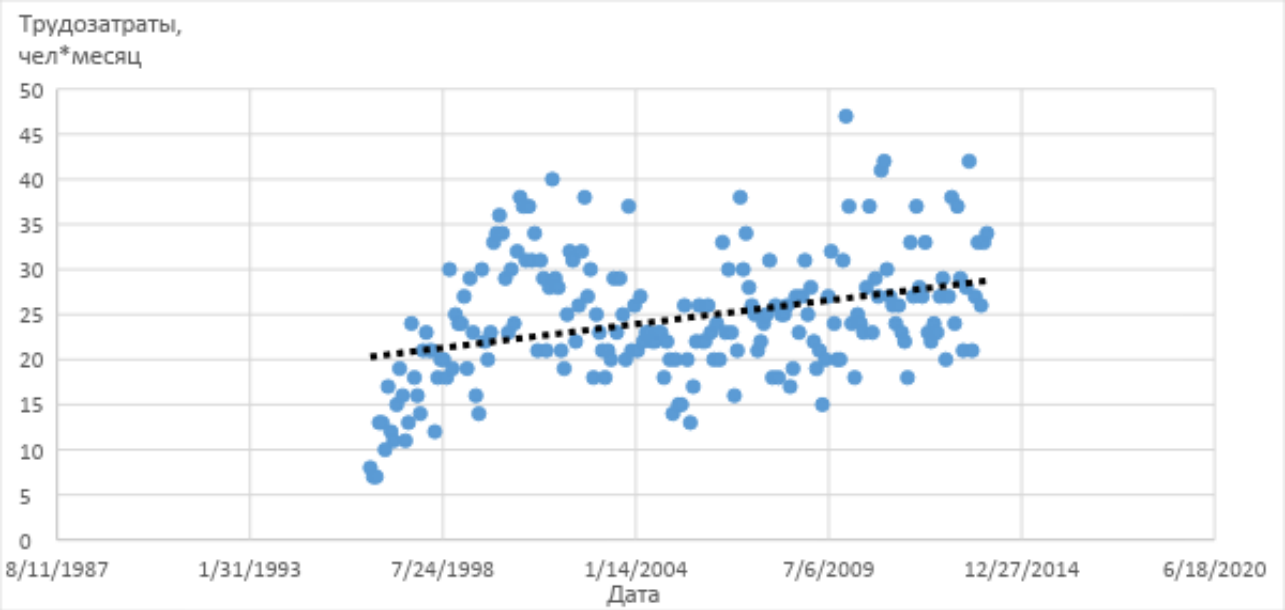

Рисунок 5.9 – Залежність трудовитрат часу

Залежність відсотка коментарів від кількості рядків коду показана на рисунку 5.10.

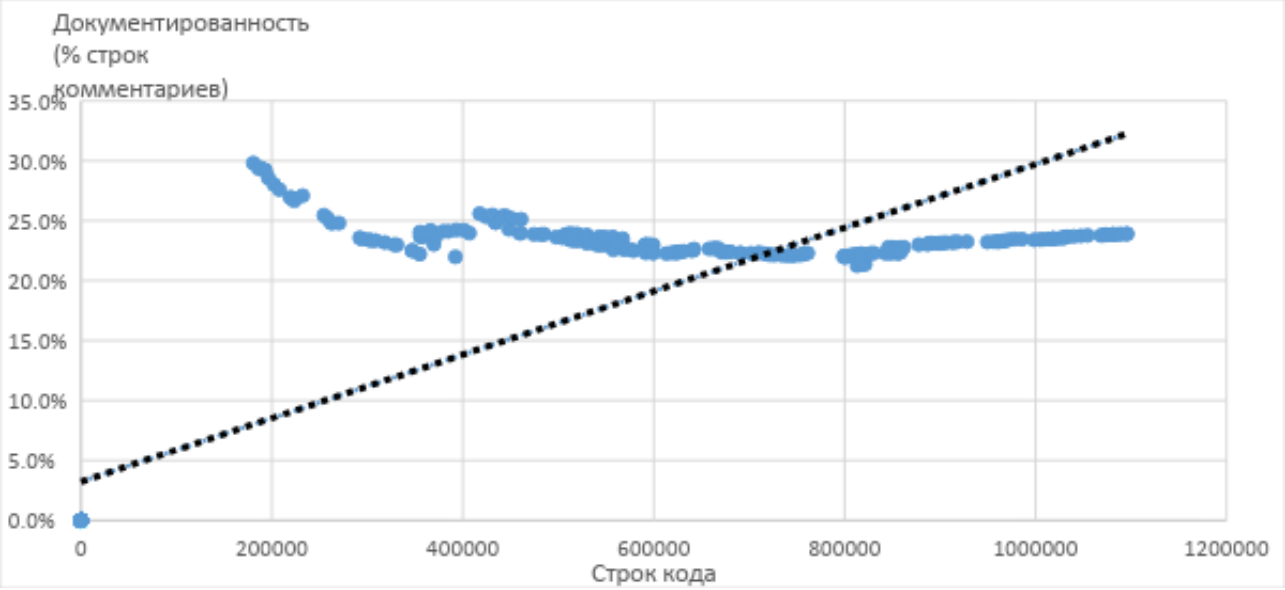

Рисунок 5. 10 – Залежність відсотка коментарів від кількості рядків коду

Залежність продуктивності від розміру проекту показана на рисунку 5.11

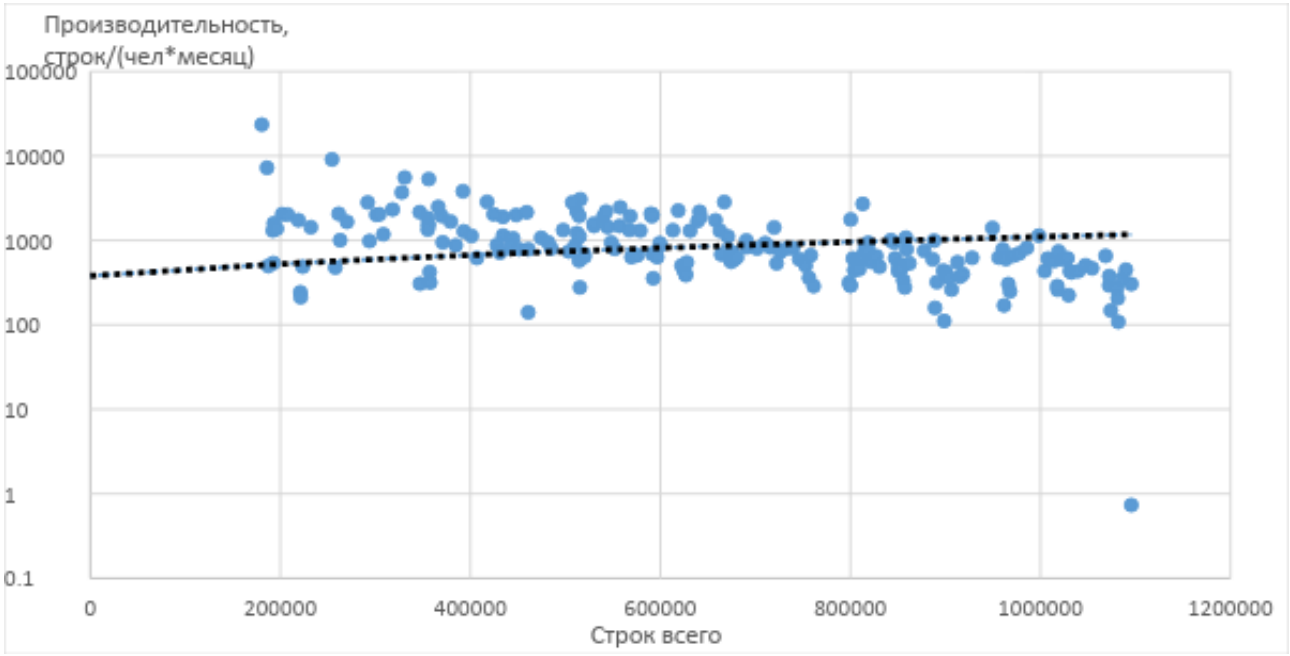

Рисунок 5.11 – Залежність продуктивності від розміру проекту

Результати аналізу дозволяють зробити наступні загальні висновки про історію досліджуваних проектів.

Метрика «Кількість доданих рядків коду» сильно корелює з метриками «кількість доданих порожніх рядків» і «активність розробника»

Метрики «кількість вилучених рядків коду» сильно корелює з метриками «кількість доданих коментарів» і «кількість порожніх доданих рядків»

На основі раніше розглянутих даних можна зробити припущення про те, що історія роботи над проектом повинна включати в себе фазу швидкого зростання, коли проекту активно розвивається, фазу більш стабільного зростання, коли реалізується значна частина функцій проекту і фазу спаду активності, коли активність роботи над проектом зменшується і зводиться до виправлення помилок і реалізації дрібних функцій.

ВИСНОВКИ

Відкрите програмне забезпечення відіграє важливу роль в IT індустрії. Зростання кількості нових проектів з відритим вихідним кодом має експонентний характер. У зв'язку з цим виникає необхідність в розробці методів оцінки поточного стану та перспектив розвитку проектів з відкритим вихідним кодом. Для вирішення вищезазначеної проблеми була сформульована мета випускної роботи магістра – оцінка стану і перспектив розвитку Open Source проектів на основі оцінки метрик, що характеризують історію роботи над проектами.

На підставі мети були сформовані завдання, які необхідно вирішити для досягнення поставленої мети.

Першим завданням став аналіз репозиторіїв Open Source проектів і вибір найбільш підходящого. Було прийнято рішення використовувати API сайту Ohloh.net, так як саме він надає найбільш повну інформацію про історії роботи над проектами. Був проаналізований API даного сайту і розроблено програмне забезпечення яке дозволяє запитувати необхідну інформацію.

На основі даних, отриманих в процесі аналізу проблеми, було виконано планування експерименту. Вхідними даними експерименту є набір метрик, що характеризує історію роботи над проектом Обробка отриманих даних включає перетворення отриманих даних до плоского вигляду, застосування методів математичної статистики для аналізу отриманих результатів.

На основі плану експерименту була виконана алгоритмизация методів обробки отриманих даних. Були розглянуті алгоритми розрахунку метрік вихідного коду, підібрані алгоритми для статистичного аналізу отриманих показників метрик. Також були обраний набір інструментів для проведення дослідження.

Вихідні дані експерименту були проаналізовані відповідно до розробленого плану. Було виконано статистичний аналіз метрик та розглянуто їх динаміка. Розрахована кореляція між метриками, знайдені пари сильно залежних метрік,

обгрунтовані причини наявності залежностей. Були висунуті гіпотези про різні аспекти розвитку проектів і виконана перевірка даних гіпотез. На підставі цих відомостей були сформульовані висновки як про поточний стан і перспективи розвитку проектів.

В ході атестаційної роботи були виконані всі етапи експериментального дослідження, що дозволяє зробити висновок про повну відповідність роботи завданню, що було поставлене.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Space product assurance. Software metrication programme definition and implementation. ECSS Secretariat, Requirements & Standards Division, 2013

2 Метрика программного обеспечения [http://ru.wikipedia.org/] от 4/12/2013.

3 Вовк О.Б. Аналіз та оцінювання якості програмного продукту, Національний університет «Львівська політехніка», 2018

4 Макконелл С., Совершенный Код, 2е издание, Microsoft Press, 2010, 268с.

5 McCabe T. J.. A Complexity Measure IEEE Transactions On Software Engineering, Vol. SE-2, NO.4, December 2016, pp 308-320

6 Коэффициент корреляции Пирсона – Методы математической статистики – URL: http://psystat.at.ua/ (дата звернення: 23.04.2020)

7 Колчков В.И. Метрология, стандартизация и сертификация. М.:Уч. пособие, 2011

8 Пархоменко О.М., Пудовкина Л.Ф., Ошевский А.Г. и др. Бизнес – план создания программного продукта: Учебн. Пособие. – Х.: Нац. аэрокосм. ун-т «Харьк. авиац. ин-т», 2018. – 198с.

9 Кобрин В.Н., Кузнецова Н.В., Кириенко П.Г. Охрана труда в отрасли: Учеб. Пособие. – Харьков: Нац. аэрокосм. ун-т «Харьк. авиац. ин-т», 2018. – 216 с.

10 Л. Константайн Л. Локвуд. Разработка программного обеспечения. – СПб.: Питер, 2014, – 592с.

11 Бейкер Р. Educational data mining and learning analytics/Р. Бейкер, Г. Сіменс – The Cambridge handbook of the learning sciences, 2014. – 274 с.

12 Ржеуцька С. Опыт применения методов кластеризации для анализа результатов дистанционного обучения // Информатизация инженерного образования: материалы международной научн.-практ. конф. 2016. № 56. С. 617 – 620.

13 Expectation-maximization algorithm // Wikipedia. URL: https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm (дата звернення: 23.04.2020).

14 K-means clustering // Wikipedia. URL: https://en.wikipedia.org/wiki/K-means_clustering (дата звернення: 25.02.2020).

15 Кукіер К. Big Data: A Revolution That Will Transform How We Live, Work, and Think/К. Кукіер, В. Штойнберг, 2018. – 236 с.

16 Лам Ч. Hadoop in Action/Ч. Лам, К. БХВ.– 2018. – 365 с.

17 Дашкевич О. Аналіз можливостей Apache Kafka в рамках забезпечення стримінгу Big Data // Информационные системы и технологии: материалы 7-й Международ. науч.-техн. конф. 2019. № 12. С. 34-35.

18 Data Mining with WEKA MOOC – Material // Machine Learning at Waikato University. URL: https://www.cs.waikato.ac.nz/ml/WEKA/mooc/dataminingwithWEKA (дата звернення: 22.04.2020).

19 Space product assurance. Software metrication programme definition and implementation. ECSS Secretariat, Requirements & Standards Division, 2013

20 Вовк О.Б. Аналіз та оцінювання якості програмного продукту, Національний університет «Львівська політехніка», 2003

21 Bailey, D. H. Experimental Mathematics: Examples, Methods and Implications / D. H. Bailey, J. M. Borwein // Notices of the AMS. – 2005. – Vol. 52. – P. 502–514.

22 Анализ вычислительной сложности крупноблочного итерационного процесса в условиях повышенной точности арифметических операций с плавающей точкой / С. С. Толстых, С. В.Мищенко, В. Е. Подольский, С. Г. Толстых // Научно-методический журнал «Информатизация образования и науки». – 2016. – №2(30). – С. 159-169.

23 Hurwitz J. What Is Platform as a Service (PaaS) in Cloud Computing? [Text] / J. Hurwitz, M. Kaufman, F. Halper, D. Kirsh // Hybrid Cloud For Dummies, Hoboken. – NJ: John Wiley & Sons, 2012. – 360 p.

24 Thomas E. B. Computer Concepts and Terminology: Types of Computers /

E. B. Thomas. – Electronic text data. – Los Alamos : University of New Mexico, 2012. – URL: http://www.unm.edu/~tbeach/terms/ types.html. – (дата звернення: 23.04.2020).

25 Howard A. R. The Surprising Technology Economics of Mainframe vs. Distributed Servers: Understanding the Impact of Your Strategy in Real Business Terms / A. R. Howard. – Electronic text data. – US : Rubin Worldwide, 2011. – 7 p. URL http://www-03.ibm.com/systems/es/ resources/ZSL03135USEN.pdf. (дата звернення: 23.04.2020)

26 Chapman, B. Using OpenMP: portable shared memory parallel programming Scientific and Engineering Computation [Text] / B. Chapman, G. Jost, Ruud van der Pas // Cambridge, Massachusetts: The MIT Press., 2008. - 353 pp.

27 Лєсна Н.С., Фоменко О.А. Дослідження методів і моделей асинхронного оновлення даних для підвищення продуктивності web-систем /Наука онлайн: міжнародний електронний науковий журнал. - 2019. - №1..-С.6-10

28 Брукс П. Метрики для управления ИТ-услугами – М.: Альпина Бизнес Брукс, 2008. – 25-31, 49-58, 99-115 с.