Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Програмної інженерії

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження моделей життєвого циклу програмного забезпечення з метою вибору ефективнішого з використанням машиного навчання для обробки великої кількості даних

Виконав: студент 2 курсу, групи ІПЗм-17-2
спеціальності 121 – Інженерія програмного забезпечення
Освітньо-наукової програми
Інженерія програмного забезпечення

Кравченко О.К.
Керівник к.т.н., доц. ПІ Голян В.В.

Допускається до захисту
Зав. кафедри, проф. _____ З.В.Дудар

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

освітньо-науково програма Програмне забезпечення систем

ЗАТВЕРДЖУЮ:

Зав. Кафедри_____

(підпис)

«____» _____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Кравченку Олександру Костянтиновичу

1. Тема роботи Дослідження моделей життєвого циклу програмного забезпечення з метою вибору ефективнішого з використанням машиного навчання для обробки великої кількості даних

2. Термін подання студентом роботи до екзаменаційної комісії 05 червня 2019 р.

3. Вихідні дані до роботи *моделі життєвих циклів програмного забезпечення, технології розробки веб-застосувань, UML-діаграми, алгоритми машиного навчання. Використовувати ОС Ubuntu, середовище розробки Rubymine 2019, СУБД PostgreSQL, мови об'єктно-орієнтованого програмування Ruby та JavaScript, фреймворки Ruby on Rails та Angular, сховище Hbase.*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної області, формування вимог до програмного продукту, архітектура та проектування програмного забезпечення, опис прийнятих проектних рішень, тестування, аналіз моделей життєвого циклу програмного забезпечення. Додатки: а) слайди презентації; б) приклади коду програми; в) наукові публікації; г) електронні матеріали до проекту на CD.*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *Модель водоспаду, V-модель, Інкрементаційна*

модель, Модель RAD, Ітеративна модель, Agile модель, Діаграма прецендентів (Use Case), Діаграма послідовності, Діаграма кооперації, Клієнт-серверна архітектура, Діаграма розгортання, Діаграма компонентів, Діаграма класів, Діаграма об'єктів, Структура системи, Схема БД, Інтерфейс планувальника задач, Структура проекту, Треновані моделі, Інтерфейс Swagger, Графіки розробки програмного продукту для моделей життєвих циклів програмного забезпечення, слайди презентації.

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	к.т.н., доц. ПІ Голян В.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Аналіз предметної галузі	19 квітня 2019 р.	
2.	Огляд існуючих методів	27 квітня 2019 р.	
3.	Моделі життєвих циклів програмного забезпечення	20 травня 2019 р.	
4.	Підготовка пояснювальної записки	25 травня 2019 р.	
5.	Спецчастина	26 травня 2019 р.	
6.	Підготовка презентації та доповіді	28 травня 2019 р.	
7.	Попередній захист	30 травня 2019 р.	
8.	Нормоконтроль, рецензування	02 червня 2019 р.	
9.	Занесення диплома в електронний архів	03 червня 2019 р.	
10.	Допуск до захисту у зав. кафедри	04 червня 2019 р.	
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання _____ 2019 р.

Студент гр. ІПЗм-17-2 _____ Кравченко О.К.

Керівник роботи _____ к.т.н., доц. ПІ Голян В.В.

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 128 с., 41 рис., 18 табл., 3 додатки, 20 джерел.

МОДЕЛІ, ЖИТТЄВИЙ ЦИКЛ, МАШИННЕ НАВЧАННЯ, RUBY, RUBY ON RAILS, ANGULAR, JAVASCRIPT, HBASE, POSTGRESQL, WEB.

Об'єктом дослідження є моделі життєвих циклів програмного забезпеченя.

Метою роботи є розробка програмного додатку, який має показувати ефективнішу модель життєвого циклу програмного забезпечення для продукту.

Методи розробки базуються на алгоритмах машиного навчання та роботи з великим набором даних та технологіях, пов'язаних з WEB розробкою на Ruby on Rails.

У результаті роботи були здійснена програмна реалізація для виявлення ефективнішої моделі та дослідженні переваги та недоліки існуючих моделей життєвого циклу програмного забезпечення.

MODELS, LIFE CYCLE, MACHINE LEARNING, RUBY, RUBY ON RAILS, ANGULAR, JAVASCRIPT, HBASE, POSTGRESQL, WEB.

The object of research is the model of life cycle software.

The goal of the work is to develop a software application that should display a more effective product life cycle software model.

Development methods are based on machine learning algorithms and work with a large set of data and technologies related to WEB development on Ruby on Rails.

Results – program implementation was implemented to identify a more efficient model and explore the advantages and disadvantages of existing life cycle software models.

ЗМІСТ

Вступ 6
1 Аналіз предметної області 8
1.1 Концепція життєвого циклу програмного забезпечення 8
1.2 Поняття моделі життєвого циклу програмного забезпечення 10
1.3 Аналіз моделей життєвого циклу програмного забезпечення 15
1.4 Постановка задачі 29
2 Формування вимог до програмного продукту 30
3 Архітектура та проектування програмного забезпечення 33
3.1 UML проектування програмного забезпечення 33
3.2 Проектування архітектури програмного забезпечення 36
3.3 Проектування бази даних 39
3.4 Створення UI / UX або іншого дизайну системи 46
4 Опис прийнятих програмних рішень 47
4.1 Серверна частина 47
4.2 Клієнтська частина 50
4.3 Модуль машиного навчання 51
5 Тестування розробленого програмного забезпечення 55
6 Аналіз ефективнішої моделі 60
Висновки 99
Перелік джерел посилання 102
Додаток А Слайди презентації 103
Додаток Б Лістинг коду програми 112
Додаток В Апробація результатів роботи 117

ВСТУП

Процес розробки програмного забезпечення є структурою, що накладається на розробку програмного продукту.

Процес є основоположним інструментом для досягнення консенсусу суспільства та сприяння дуже великої кількості людей для роботи над спільним проектом. Методичний підхід до розробки програмного забезпечення призводить до меншої кількості дефектів і, отже, у кінцевому рахунку забезпечує більш короткий термін випуску та кращу вартість. Необхідність вибору та слідування процесу розробки програмного забезпечення полягає в забезпеченні бажаної дисципліни для створення якісного продукту для успішного ведення бізнесу та для уникнення втрати часу, грошей, деморалізації в розробниках тощо.

Комп'ютери та програмне забезпечення є частиною нашого існування – освітнього, професійного та особистого, вони зробили наше життя легким і точним від малого ринку бізнесу до ракетної науки. Таким чином, програмне забезпечення більше не є програмуванням для індивідуальних інтересів або заради цього, програмне забезпечення – це не просто програма, яка повинна виконуватися для виконання завдання, а взаємодія програм, структури даних і документації і є складною структурою, яку потрібно розробляти, перевіряти і підтримувати.

Життєвий цикл розробки програмного забезпечення стикається з багатьма проблемами на кожному етапі [1]. Найгірші ситуації – це запуск проекту з новими співробітниками, які не мають досвіду в галузі, не мають доказових технологій, а також мають складний термін. Поряд з технічними виклики будь-якої ситуації можуть перешкодити розробці програмного забезпечення і поставити управління в ризиковану і страшну кризу, яка не вирішила добре цю ситуацію, може призвести до того, що – продукти перевищують як вартість, так і оцінку часу, але все ще закінчуються поганою якістю. Вони не відповідають технічним вимогам, визначеним споживачем, і, нарешті, призводять до відмови бізнесу.

Ціллю даної роботи є проведення глибокого аналізу та порівняння різноманітних моделей життєвого циклу програмного забезпечення, дослідження процесу вибору найефективнішою моделі в залежності від базових характеристик і ресурсів, яка має команда до початку створення продукту.

Дана робота допомагає проаналізувати питання, які стосуються вибору моделі життєвого циклу програмного забезпечення на початку, та допомагає актуально підібрати можливі альтернативи в підходах та організації циклу програмного забезпечення в процесі розробки.

Практичну частину роботи можна охарактеризувати актуальною для обміну файлами, а саме дипломними та науковими роботами, між викладачами та студентами. В подальшому дану роботу можна використовувати для наступних поколінь студентів.

Мета магістерської атестаційної роботи – проаналізувати деякі поширені моделі життєвого циклу програмного забезпечення, провести їх оцінку. Треба оцінити які моделі є найбільш ефективними для досягнення бізнес цілей програмного продукту. Треба виявити сильні та слабкі сторони кожної з моделей. Треба охарактеризувати за надійністю (кількостю дефектів), якістю, можливістю підтримки, наявністю актуальної документації, кінечної вартості та часом розробки кінцевого продукту. Треба спробувати знайти лідера серед обраних моделей, або виявити в яких ситуаціях слід надавати перевагу певній моделі, або поєднати найкращі сторони з усіх моделей для створення нової ефективної альтернативи.

В рамках магістерської атестаційної роботи необхідно обрати та проаналізувати поширені моделі, змоделювати та розробити програмний продукт, який мав би аналізувати та порівняти їх, а також провести аналіз програмних продуктів для знаходження найбільш ефективної моделі під певний набір характеристик та ресурсів програмного продукту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Життєвий цикл розробки систем, який також називають життєвим циклом розробки додатків, є терміном, що використовується в інженерних системах, інформаційних системах та інженерії програмного забезпечення для опису процесу планування, створення, тестування та розгортання інформаційної системи. Концепція життєвого циклу розробки систем застосовується до ряду апаратних і програмних конфігурацій, оскільки система може складатися тільки з апаратного забезпечення, програмного забезпечення або комбінації обох [2]. Зазвичай у цьому циклі є шість етапів: аналіз, розробка, тестування, впровадження, документація та оцінка.

1.1 Концепція життєвого циклу програмного забезпечення

Життєвий цикл розробки систем складається з низки чітко визначених та чітких фаз роботи, які використовуються системними інженерами та розробниками систем для планування, проектування, побудови, тестування та доставки інформаційних систем. Як і все, що виробляється на конвеєрі, ЖЦПЗ має на меті виробляти високоякісні системи, які відповідають або перевищують очікування клієнтів, виходячи з вимог замовника, надаючи системи, які переміщуються через кожну чітко визначену фазу, в межах запланованих часових рамок і кошторисів. Комп'ютерні системи є складними і часто (особливо з недавнім зростанням сервісно-орієнтованої архітектури) зв'язують декілька традиційних систем, потенційно поставляються різними постачальниками програмного забезпечення. Щоб керувати цим рівнем складності, було створено ряд моделей або методологій ЖЦПЗ, таких як водоспад, спіраль, розробка програмного забезпечення Agile, швидке створення прототипів, інкрементне та синхронізування та стабілізація.

ЖЦПЗ може бути описаний уздовж спектра гнучкої до ітеративної до послідовної методології. Agile методології, такі як XP і Scrum, зосереджуються на легких процесах, які дозволяють швидко змінюватися (без обов'язкового дотримання схеми підходу ЖЦПЗ) по циклу розробки. Ітеративні методології, такі як метод Rational Unified Process і метод розвитку динамічних систем, зосереджуються на обмеженій кількості проектів і розширюють або вдосконалюють продукти за допомогою декількох ітерацій. Моделі з послідовним або великим проектуванням (BDUF), такі як водоспад, зосереджуються на повному і правильному плануванні, щоб керувати великими проектами і ризиками для успішних і передбачуваних результатів. Інші моделі, такі як анаморфний розвиток, мають тенденцію зосередитися на формі розвитку, яка керується масштабами проекту та адаптивними ітераціями розвитку ознак.

В управлінні проектами проект може бути визначений як з життєвим циклом проекту (PLC), так і з ЖЦПЗ, під час якого відбуваються незначні дії. Згідно з Тейлором (2004), «життєвий цикл проекту охоплює всі види діяльності проекту, в той час як життєвий цикл розвитку систем зосереджується на реалізації вимог продукту».

ЖЦПЗ використовується під час розробки IT-проекту, він описує різні етапи, що беруть участь у проекті, з дошки креслення, через завершення проекту.

ЖЦПЗ – це не сама методологія, а опис етапів життєвого циклу програмного забезпечення. Ці етапи (в широкому сенсі) – це дослідження, аналіз, проектування, побудова, тестування, впровадження та технічне обслуговування та підтримка. Всі методики розробки програмного забезпечення (такі, як більш відомі методології водоспаду та скраму) слідують за етапами ЖЦПЗ, але спосіб їх виконання значно варіюється між методологіями. У методології Scrum, наприклад, можна сказати, що одна історія користувача проходить через всі етапи ЖЦПЗ протягом одного двотижневого спринту. Порівняйте це з методологією водоспаду, як інший приклад, де кожна вимога бізнесу (записана на етапі аналізу ЖЦПЗ в документі, що називається специфікацією бізнес-вимог) перекладається на характеристики / функціональні описи (записані на етапі проектування в документі Функціональна

специфікація), яка потім побудована за один раз як набір функцій рішення, як правило, протягом трьох-дев'яти місяців або більше. Ці методології, очевидно, зовсім інші підходи, але вони обидва містять етапи ЖЦПЗ, на яких виникає потреба, а потім проходить через фази життєвого циклу, що закінчуються в заключній фазі технічного обслуговування і підтримки, після чого (як правило) починається весь життєвий цикл. для подальшої версії програмного забезпечення.

1.2 Поняття моделі життєвого циклу програмного забезпечення

Життєвий цикл розробки програмного забезпечення, коротко кажучи, ЖЦПЗ – це чітко визначена, структурована послідовність етапів розробки програмного забезпечення для розробки призначеного програмного продукту.

ЖЦПЗ визначає повний цикл розвитку, тобто всі завдання, пов'язані зі збором вимоги щодо обслуговування продукту.

ЖЦПЗ надає ряд кроків, які необхідно дотримуватися для ефективного проектування та розробки програмного продукту. Структура ЖЦПЗ включає в себе етапи програмного забезпечення:

– спілкування;

– збір вимог;

– техніко-економічне обґрунтування;

– системний аналіз;

– дизайн програмного забезпечення;

– кодування;

– тестування;

– інтеграція;

– реалізація;

– експлуатація та обслуговування;

– диспозиція.

1.2.1 Спілкування

Це перший крок, коли користувач ініціює запит на потрібний програмний продукт. Він звертається до постачальника послуг і намагається обговорити умови. Він подає запит до організації, що надає послуги, у письмовій формі.

1.2.2 Збір вимог

На цьому етапі команда розробників програмного забезпечення працює над виконанням проекту. Команда проводить обговорення з різними зацікавленими сторонами з проблемних областей і намагається виявити якомога більше інформації щодо їх вимог. Вимоги розглядаються і розділені на вимоги користувачів, системні вимоги та функціональні вимоги. Вимоги збираються з використанням ряду практик, як зазначено:

– вивчення існуючої або застарілої системи та програмного забезпечення;

– проведення інтерв'ю користувачів і розробників;

– посилаючись на базу даних;

– збір відповідей з анкет.

1.2.3 Техніко-економічне обґрунтування

Після збору вимог команда висуває приблизний план програмного процесу. На цьому кроці команда аналізує, чи може бути зроблено програмне забезпечення для виконання всіх вимог користувача і якщо є можливість програмного забезпечення, що не є більш корисним. З'ясувалося, якщо проект фінансово,

практично і технологічно здійсненний для організації. Є багато алгоритмів, які допомагають розробникам зробити висновок про доцільність створення програмного проекту.

1.2.4 Системний аналіз

На цьому кроці розробники вирішують дорожню карту свого плану і намагаються вивести кращу модель програмного забезпечення, придатну для проекту. Системний аналіз включає в себе розуміння обмежень програмних продуктів, проблем, пов'язаних із системою навчання, або змін, які необхідно здійснити в існуючих системах заздалегідь, виявлення та вирішення впливу проекту на організацію та персонал тощо. відповідних ресурсів.

1.2.5 Дизайн програмного забезпечення

Наступний крок полягає в тому, щоб знизити всі знання вимог і аналіз на столі і розробити програмний продукт. Вхідні дані від користувачів та інформація, зібрана на етапі збору вимог, є входом цього кроку.

1.2.6 Кодування

Цей етап також відомий як етап програмування. Реалізація розробки програмного забезпечення починається з точки зору написання програмного коду

на відповідній мові програмування та ефективного розробки виконуваних програм без помилок.

1.2.7 Тестування

За оцінками, необхідно перевірити 65% всього процесу розробки програмного забезпечення. Помилки можуть зіпсувати програмне забезпечення від критичного рівня до його власного видалення. Тестування програмного забезпечення здійснюється під час кодування розробниками, а ретельне тестування проводиться експертами з тестування на різних рівнях коду, наприклад, тестування модулів, тестування програм, тестування продуктів, внутрішнє тестування та тестування продукту на кінці користувача. Раннє виявлення помилок і їх виправлення є ключем до надійного програмного забезпечення. Це дозволяє завчасно знайти та пофіксити помилки в документації та структурі програми.

1.2.8 Інтеграція

Програмне забезпечення може потребувати інтеграції з бібліотеками, базами даних та іншими програмами. Цей етап ЖЦПЗ бере участь у інтеграції програмного забезпечення з об'єктами зовнішнього світу. Вихід цього кроку приходить у вигляді двох конструкцій; логічний дизайн і фізичний дизайн. Інженери виробляють метадані та словники даних, логічні діаграми, діаграми потоків даних, а в деяких випадках псевдокоди. Це один з найважливіших етапів ЖЦПЗ, який відповідає за взаємодію з бібліотеками.

1.2.9 Реалізація

Це означає встановлення програмного забезпечення на машинах користувачів. Іноді програмне забезпечення потребує конфігурації після встановлення на кінці користувача. Програмне забезпечення перевіряється на переносимість, адаптивність і питання інтеграції вирішуються під час впровадження.

1.2.10 Експлуатація та обслуговування

Цей етап підтверджує роботу програмного забезпечення з точки зору більшої ефективності та меншої кількості помилок. Якщо потрібно, користувач навчається або допомагає документацією про те, як працювати з програмним забезпеченням, і як підтримувати роботу програмного забезпечення. Програмне забезпечення підтримується вчасно, оновлення коду відповідно до змін, що відбуваються в середовищі або технології користувача. Цей етап може зіткнутися з проблемами, пов'язаними з прихованими помилками та невідомими проблемами реального світу.

1.2.11 Диспозиція

З плином часу програмне забезпечення може зменшитися на фронті продуктивності. Це може бути повністю застарілим або може знадобитися інтенсивне оновлення. Отже, виникає нагальна потреба у ліквідації більшої частини системи. Ця фаза включає в себе архівування даних і необхідні компоненти

програмного забезпечення, закриття системи, планування діяльності з ліквідації та завершення системи у відповідний час кінця системи.

1.3 Аналіз моделей життєвого циклу програмного забезпечення

ЖЦПЗ – це абревіатура, яка використовується для опису життєвих циклів розробки програмного забезпечення або систем. Незважаючи на те, що поняття між цими двома є однаковими, одна відноситься до життєвого циклу програмного забезпечення, коли інша – до системи, яка охоплює розвиток програмного забезпечення. У цій статті підкреслюється розробка програмного забезпечення, хоча ті ж принципи можуть бути перетворені в системи. Крім того, більшість інновацій та лідерства в розумінні моделей та концепцій прийшли з розробки програмного забезпечення, і розвиток систем значною мірою запозичив розробку програмного забезпечення.

ЖЦПЗ, таким чином, є концептуальною структурою або процесом, який розглядає структуру етапів, що беруть участь у розробці програми від її початкового техніко-економічного обґрунтування до його розгортання на місцях та технічного обслуговування. Існує кілька моделей, що описують різні підходи до процесу ЖЦПЗ. Модель ЖЦПЗ, як правило, використовується для опису кроків, які виконуються в рамках життєвого циклу.

Необхідно мати на увазі, що модель відрізняється від методології в тому сенсі, що перша описує, що робити, тоді як остання, крім того, описує, як це зробити. Таким чином, модель є описовою, в той час як методологія є нормативною. В результаті розглядаються моделі ЖЦПЗ в цій статті з точки зору їх відповідності конкретним типам програмних проектів. Цей підхід переосмислює контекст, в якому використовується модель ЖЦПЗ. Наприклад, модель водоспаду може бути найкращою моделлю для використання при розробці корпоративної реляційної бази даних, але вона не може бути оптимальною моделлю при розробці

веб-додатків. Тому розглянемо моделі для трьох різних випадків використання або категорій програмних продуктів, щоб забезпечити контекст для їх застосування, а саме:

Категорія 1. Програмне забезпечення, яке надає функціональні можливості на зворотній лінії. Як правило, це програмне забезпечення, яке надає послугу іншим додаткам.

Категорія 2. Програмне забезпечення, яке надає послугу кінцевому користувачеві або додатку кінцевого користувача. Як правило, це програмне забезпечення, яке інкапсулює бізнес-логіку або формати даних, щоб зробити їх більш зрозумілими для кінцевого користувача.

Категорія 3. Програмне забезпечення, яке надає візуальний інтерфейс кінцевому користувачеві. Зазвичай це інтерфейсний додаток, що є графічним інтерфейсом користувача (GUI).

Моделі ЖЦПЗ також можуть бути класифіковані як такі, що підпадають під три широкі групи: лінійні, ітеративні та комбінації лінійних і ітераційних моделей. Лінійна модель є послідовною, коли один етап, після його завершення, безповоротно призводить до ініціювання наступного етапу. З іншого боку, ітераційна модель забезпечує перегляд усіх етапів моделі в майбутньому; ідея полягає в тому, що розвиток залишається постійним зусиллям для вдосконалення протягом всього життєвого циклу. Комбінована модель визнає, що ітераційний процес розробки може бути зупинений на певному етапі. Хоча існує велика кількість моделей ЖЦПЗ, ми розглянемо найбільш важливі або ті, що набули популярності.

1.3.1 Модель водоспаду

Модель водоспаду, також відома як каскадна модель, була вперше задокументована Бенінгтоном в 1956 році і модифікована Уінстоном Ройсом в 1970

році. Вона підкріпила всі інші моделі, оскільки вона створила міцну основу для вимог, які повинні бути визначені. аналізували до будь-якої конструкції або розробки.

Оригінальна каскадна модель Беннінгтона рекомендувала розвивати програмне забезпечення поступово: операційний аналіз, експлуатаційні специфікації, специфікації проектування і кодування, розробка, тестування, розгортання, оцінка. Визнаючи, що при створенні базової лінії в кінці кожного етапу можуть виникнути непередбачувані труднощі при проектуванні, Ройс розширив цю модель, надавши петлю зворотного зв'язку, щоб кожен попередній етап міг бути переглянуто. Ця ітерація показана далі, використовуючи червоні стрілки для потоку процесу; це дозволило перекривати етапи на додаток до попередньої стадії. Але Ройс також вважав, що ця організація може виявитися неадекватною, оскільки ітерація може знадобитися для того, щоб перевершити ітерацію наступної пари. Це відбувається в кінці етапу оцінювання (або тестування), коли вам може знадобитися перейти на стадію проектування в обхід етапу розробки або наприкінці етапу розробки, де вам може знадобитися перейти до етапу визначення вимог в обхід стадія аналізу. Таким чином, дизайн і вимоги, відповідно, перевизначаються, якщо тестування або проектування воєн-гуркіт. Це показано далі пунктирними стрілками, які показують більш складну петлю зворотного зв'язку. Далі, Ройс запропонував, щоб попередній етап проектування міг бути включений між вимогами та етапами аналізу [3]. Оригінальна каскадна модель Беннінгтона рекомендувала розвивати програмне забезпечення поступово: операційний аналіз, експлуатаційні специфікації, специфікації проектування і кодування, розробка, тестування, розгортання, оцінка. Він вважав, що це стосуватиметься центральної ролі, яку фаза проектування відіграє у мінімізації ризиків шляхом повторного відвідування декількох разів у вигляді комплексної петлі зворотного зв'язку. На рисунку 1.1 зображена модель водоспаду. Одним з аспектів моделі водоспаду є її вимога до документування.

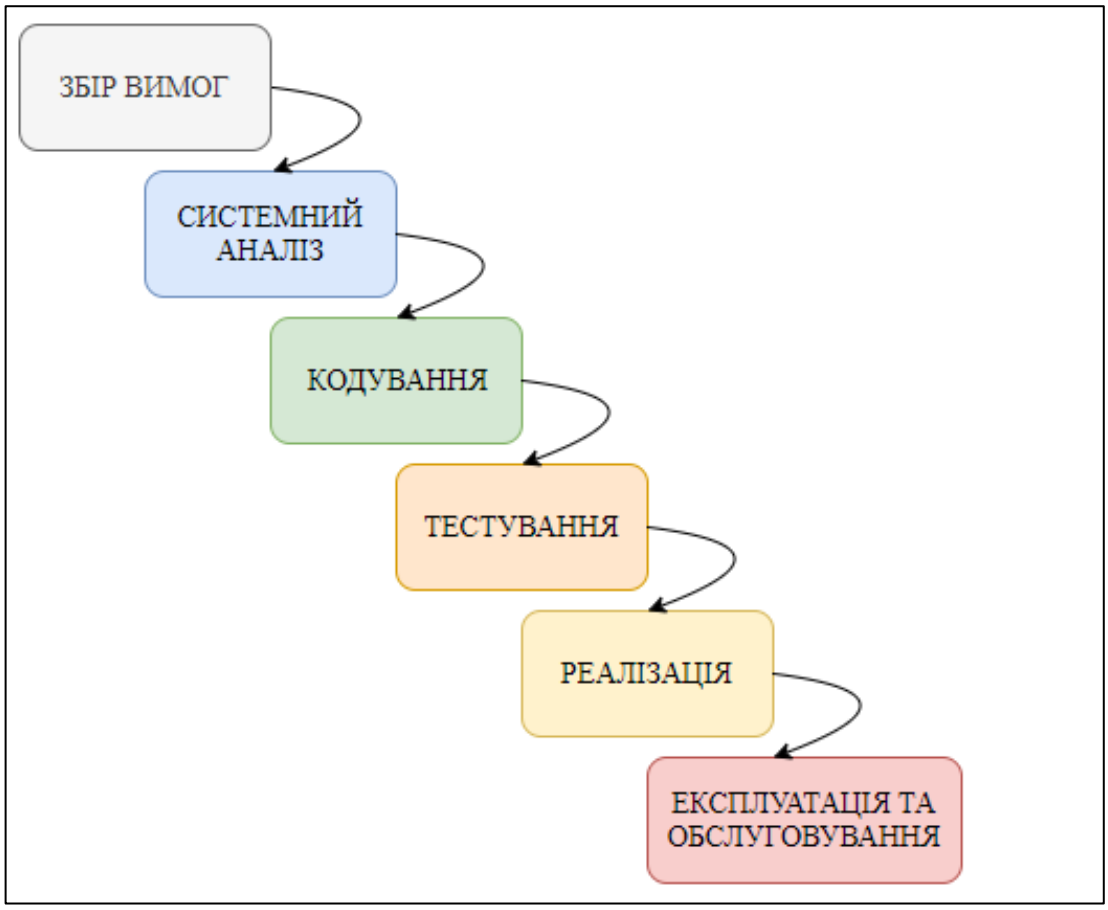

Рисунок 1.1 – Модель водоспаду з ітеративним зворотним зв'язком Ройса

Одним з аспектів моделі водоспаду є її вимога до документування. Ройс запропонував створити принаймні шість різних типів документів:

– вимоги до документа на етапі вимог;

– попередня проектна специфікація на етапі попереднього проектування;

– специфікація дизайну інтерфейсу на етапі проектування;

– кінцева специфікація проекту, що активно переглядається та оновлюється протягом кожного візиту етапу проектування; це додатково підвищується під час етапів розробки та валідації;

– план випробувань на етапі проектування; це пізніше оновлюється результатами тестування під час етапу перевірки або тестування;

– керівництво з експлуатації або інструкції на етапі розгортання.

Забезпечення якості вбудовано в модель водоспаду шляхом розділення кожної стадії на дві частини: одна частина виконує роботу, як це називається на сцені, а інша перевіряє чи перевіряє її. Наприклад, стадія проектування включає в

себе перевірку (для оцінки того, чи підходить вона до мети), на стадії розробки є тестування на одиницю та інтеграцію, а етап перевірки містить системне тестування як його частину.

1.3.2 B-модель

У 1988 році Біррелл і Улд обговорили розширення моделі водоспаду, яку вони назвали b-моделлю. Розширюючи операційний життєвий цикл (позначений як цикл технічного обслуговування), а потім прикріплюючи його до моделі водоспаду, вони розробили b-модель.

У певному сенсі, b-модель була спробою модифікувати водоспад, створивши процес еволюційного вдосконалення. На рисунку 1.2 зображена B-модель.

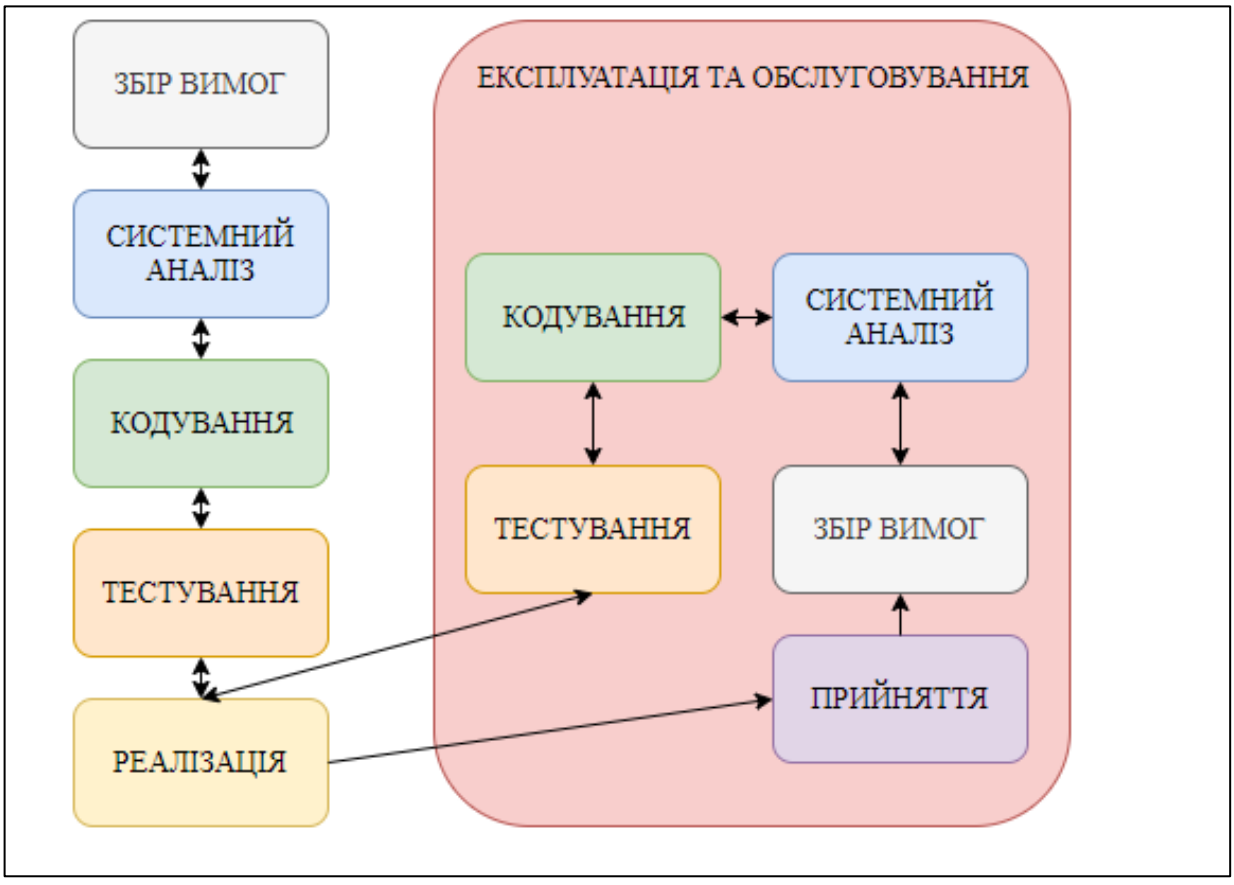

Рисунок 1.2 – B-модель як розширення моделі водоспаду

Це було зроблено для забезпечення постійного вдосконалення програмного забезпечення або система стала б частиною етапів розвитку. Крім того, вони вважали, що треба захопити альтернативу застарілості, щоб розширити або навіть нові системи могли бути розроблені як побічні від початкової системи.

B-модель, однак, більше підходить для розробки програмного забезпечення категорії 1, як його двоюрідного брата, водоспаду.

1.3.3 V-модель

V-модель, також відома як модель vee, вперше представлена на симпозіумі NCOSE в Чаттанузі, штат Теннессі 1991 року, була розроблена NASA. Модель є варіацією моделі водоспаду у V-формі, складеної навпіл на найнижчому рівні розкладання.. На рисунку 1.3 зображена V-модель.

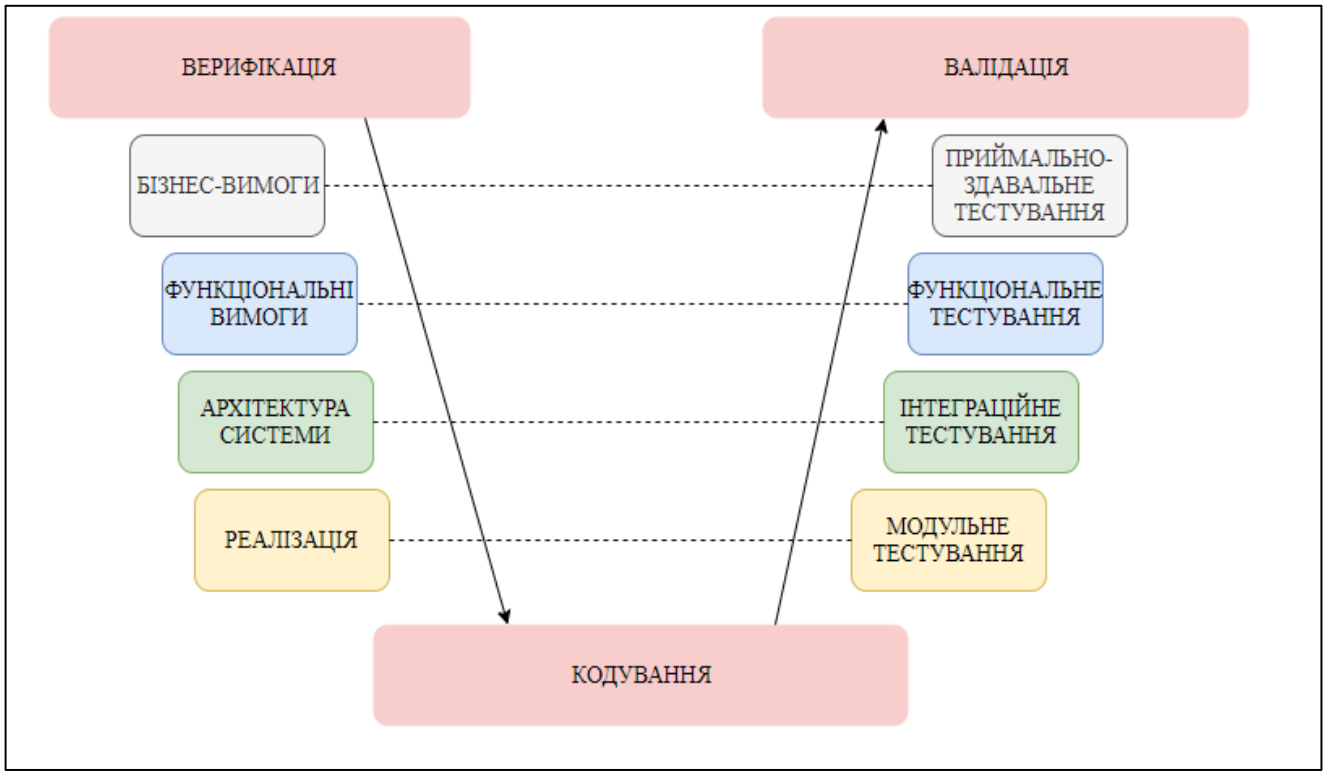

Рисунок 1.3 – V-модель

Ліва нога V-форми представляє еволюцію вимог користувача до все більш дрібних компонентів через процес декомпозиції і визначення; Права нога являє інтеграцію та перевірку системних компонентів у послідовні рівні впровадження та складання. Вертикальна вісь зображує рівень розкладання від системного рівня, у верхній частині, до найнижчого рівня деталізації на рівні компонентів внизу. Таким чином, чим складнішою є система, тим глибшою є форма V з відповідно більшим числом етапів. V-модель, будучи тривимірною, також має вісь, яка є нормальною до площини, осі z. Це являє собою компоненти, пов'язані з множинними поставками. Тому час представляється двома осями: зліва направо і в площину.

Модель v є симетричною по двох ногах, так що процедури правильності і гарантії якості визначаються для правої ноги під час відповідної стадії лівої ноги. Це гарантує те, що вимоги та дизайн можуть бути перевірені у вигляді SMART (конкретного, вимірюваного, досяжного, реалістичного та обмеженого в часі), що дозволяє уникнути таких заяв, як "користувач, який є користувачем", який є дійсним вимоги, що не підлягають перевірці.

Варіацією v-моделі є модель vee +. Це додає залученості користувачів, ризики та можливості до осі z-v-моделі. Таким чином, користувачі залишаються залученими до тих пір, поки їх не будуть цікаві декомпозиції. Застосування цього полягає в тому, що будь-які аномалії, ідентифіковані під час стадій інтеграції та складання, подаються користувачеві для прийняття або, у разі відхилення, повинні бути вирішені на відповідних рівнях розкладання; аномалії можуть бути вирішені як помилки або прийняті як конструктивні особливості. У контрасті, модель спіралі додає участь користувача під час його діяльності зі зменшення ризику, тоді як v та моделі водоспаду включають її під час етапу визначення початкових вимог. Крім того, особливості ризику та можливостей vee + означають, що певні стаги v на нижчих рівнях декомпозиції можуть бути скорочені для інтеграції пакетів COTS (комерційних, готових програм). Це означає, що різні рівні продуктів, на різних рівнях розкладання, можуть бути додані до життєвого циклу безперервно.

Іншою варіацією v-моделі є модель vee ++, яка додає процеси перетину до моделі vee +; Процес декомпозиційного аналізу і розв'язання додають до лівої ноги v-форми, а процес правильного аналізу і декомпозиції додають до правого.

1.3.4 Інкрементальна модель

Інкрементальну модель можна розглядати як тривимірне представлення моделі водоспаду. На рисунку 1.4 зображена інкрементальна модель.

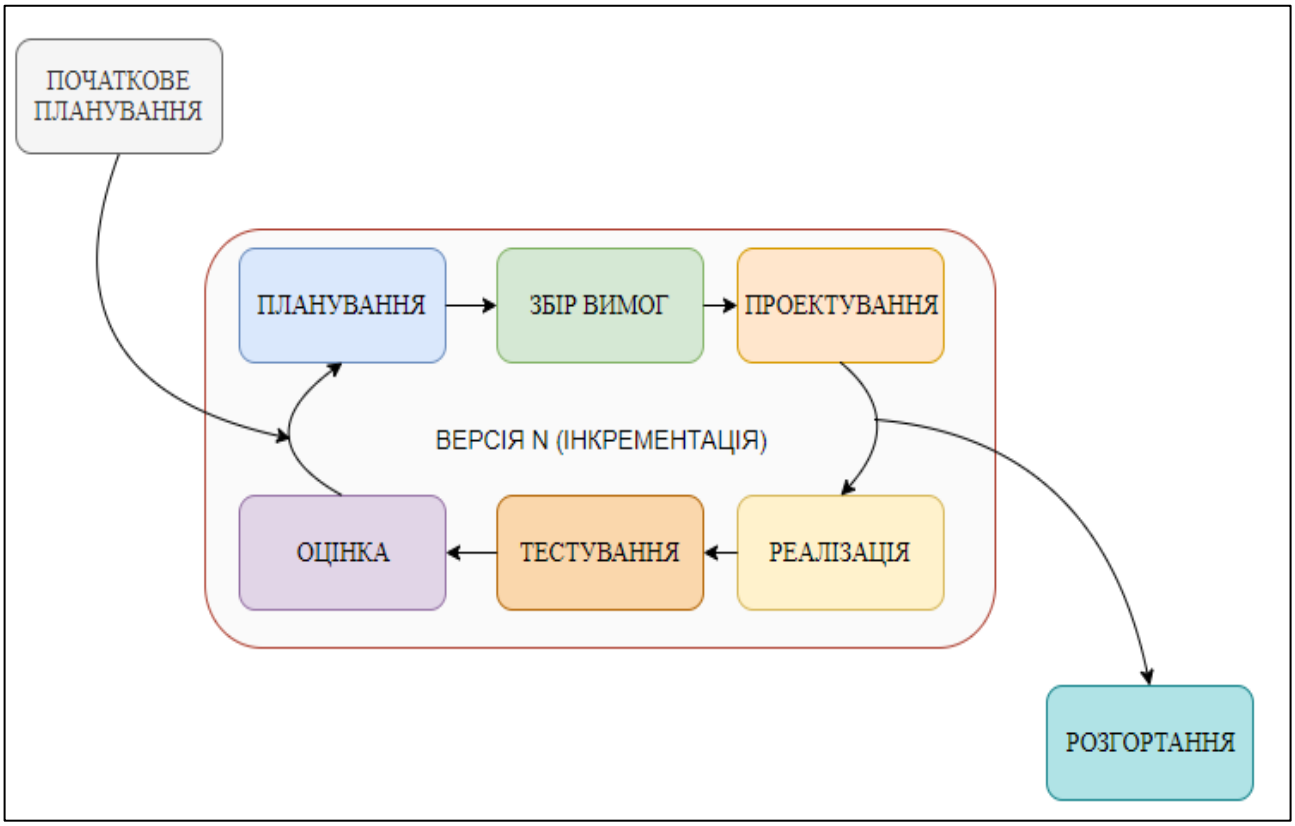

Рисунок 1.4 – Інкрементна модель

Вісь z містить серію моделей водоспаду для представлення кількості ітерацій, які будуть зроблені для підвищення функціональності кінцевого продукту. Інкрементальна модель, у цьому сенсі, є модифікацією моделі водоспаду, що наближається до спіральної моделі [4].

Система вводиться у виробництво, коли доставляється перший приріст. Перший приріст часто є основним продуктом, на якому розглядаються основні вимоги, а додаткові функції додаються в наступних кроках. Після того, як клієнт проаналізував основний продукт, є розробка плану для наступного збільшення.

Основними силами цієї моделі є:

– відгуки від попередніх ітерацій можуть бути включені в поточну ітерацію;

– зацікавлені сторони можуть бути залучені через ітерації, і тому допомагають визначити архітектурні ризики раніше;

– полегшує доставку продукту з ранніми, додатковими випусками, які розвиваються до повного набору функцій з кожною ітерацією;

– інкрементна реалізація дозволяє здійснювати моніторинг змін, а також вирішувати проблеми , які необхідно вирішувати для зменшення ризиків.

1.3.5 Модель RAD

Модель RAD – модель швидкого розробки додатків. Це тип інкрементної моделі. В моделі RAD компоненти або функції розробляються паралельно, як якщо б вони були міні-проектами. Розробки є коробкою часу, доставляються і потім збираються в робочий прототип [5].

Це може швидко дати клієнту щось побачити і використовувати, а також забезпечити зворотний зв'язок щодо доставки та їх вимог.

На відміну від традиційного підходу, при якому використовувалися специфічні засоби прототипування, не призначені для побудови реальних додатків, а прототипи викидалися після того, як виконували завдання усунення неясностей у проекті, в підході RAD кожен прототип розвивається в частину майбутньої системи. Таким чином, на наступну фазу передається більш повна і корисна інформація. На рисунку 1.5 зображена модель RAD.

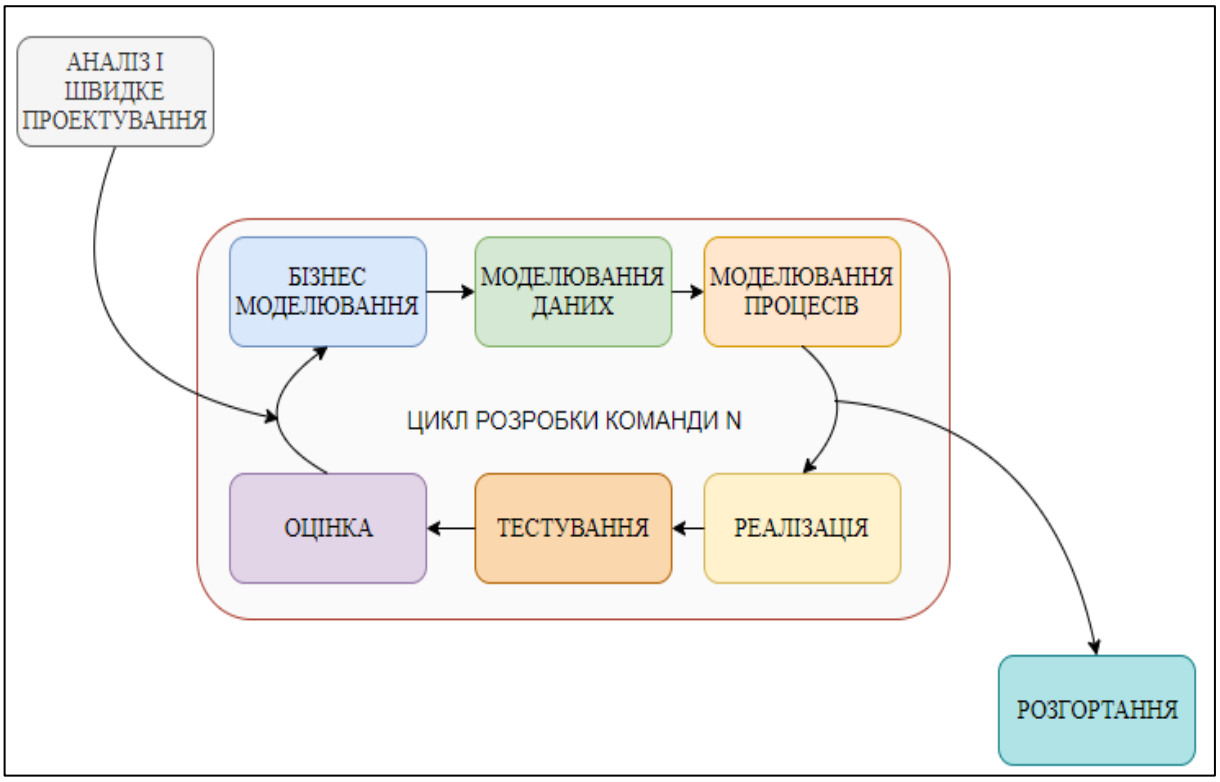

Рисунок 1.5 – Модель RAD

Після закінчення робіт кожної окремої команди розробників проводиться поступова інтеграція даної частини системи з іншими, формується повний програмний код, виконується тестування спільної роботи даної частини програми з іншими, а потім тестування системи в цілому.

Фазами моделі швидкої розробки додатків (RAD) є:

– бізнес-моделювання: інформаційний потік ідентифікується між різними бізнес-функціями;

– моделювання даних: інформація, отримана в результаті бізнес-моделювання, використовується для визначення об'єктів даних, необхідних для бізнесу;

– моделювання процесів: об'єкти даних, визначені в моделюванні даних, перетворюються для досягнення потоку бізнес-інформації для досягнення певної бізнес-мети;

– генерація додатків: автоматизовані інструменти використовуються для перетворення моделей процесів у код і фактичну систему;

– тестування і оборот: тестуйте нові компоненти та всі інтерфейси.

1.3.6 Модель Agile

Модель розробки програмного забезпечення Agile була призначена головним чином для того, щоб допомогти розробникам створити проект, який може швидко адаптуватися до перетворення запитів. Отже, найважливішим завданням для розробки моделі Agile є спрощення та швидке досягнення проекту [6]. Для досягнення цього завдання розробникам необхідно зберегти спритність під час розробки. Спритність може бути досягнута шляхом коригування прогресу в проекті шляхом усунення діяльності, яка може не мати вирішального значення для цього конкретного проекту.

На рисунку 1.6 зображена модель Agile.

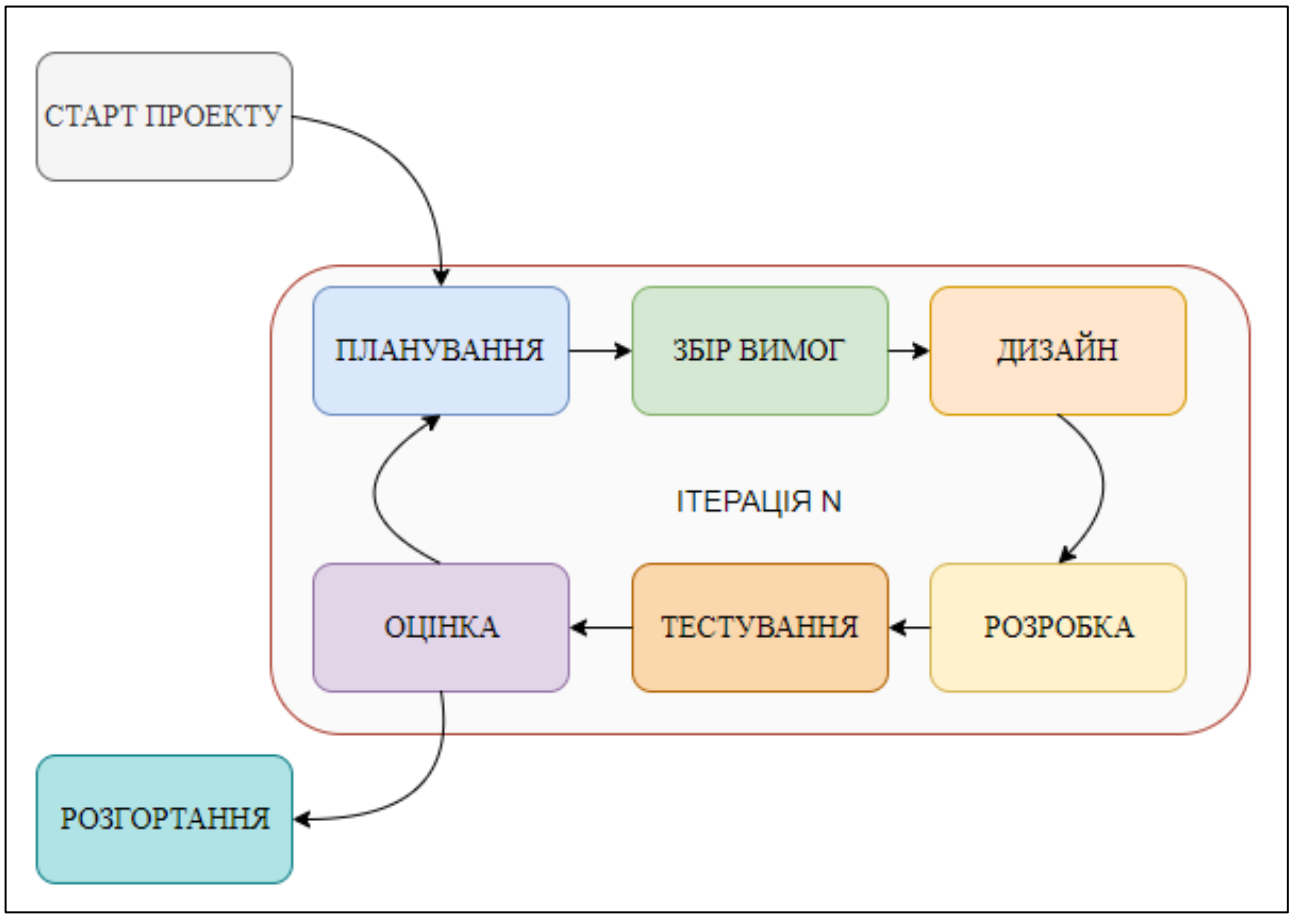

Рисунок 1.6 – Модель Agile

Для кожної ітерації залучається міжфункціональна група розробників, що працюють одночасно в різних областях розробки продукту, таких як:

– планування;

– аналіз вимог;

– дизайн;

– розвиток;

– тестування одиниць;

– розгортання.

Після завершення всіх ітерацій, що призводять до успішного продукту, воно відображається клієнтам та іншим зацікавленим особам, які схвалюють та приймають кінцевий продукт.

По суті, Agile модель – це колективна ітерація, згрупована для того, щоб зробити практику розвитку продукту. Ці процедури поділяють деякі суттєві якості, але впевнені в тому, що між кожною з них не існує різниці.

Успіх Agile в просуванні більш швидкої, більш орієнтованою на споживача і рентабельною розробки ПЗ привернув до неї увагу практично всіх інших галузей. В останні роки найрізноманітніші дисципліни, такі як маркетинг, бухгалтерський облік, виробництво та багато інших, були успішно переглянуті в рамках Agile.

1.3.7 Ітеративна модель

У ітеративній моделі ітераційний процес починається з простої реалізації невеликого набору програмних вимог і ітераційно посилює нові версії, поки не буде реалізована і готова до розгортання повна система [7].

Ітеративна модель життєвого циклу не намагається почати з повної специфікації вимог. Натомість, розробка починається з визначення та реалізації лише частини програмного забезпечення, яке потім переглядається для визначення подальших вимог. Потім цей процес повторюється, створюючи нову версію

програмного забезпечення в кінці кожної ітерації моделі. На рисунку 1.7 зображена ітеративна модель.

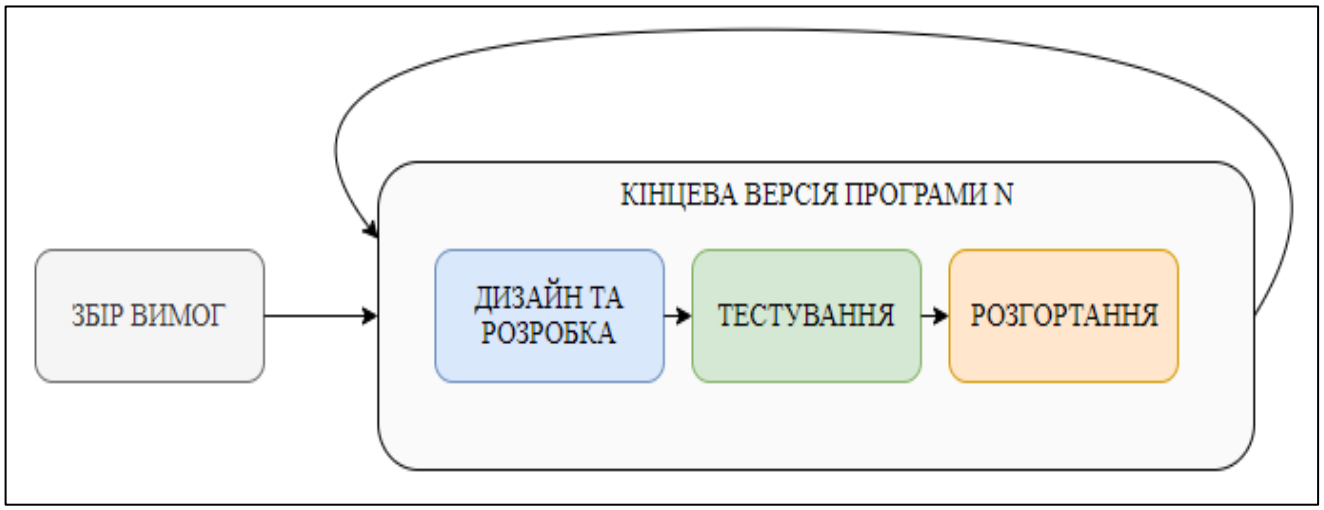

Рисунок 1.7 – Ітеративна модель

Ітераційний процес починається з простої реалізації підмножини програмних вимог і ітеративно посилює розвиваються версії, поки не буде реалізована повна система. На кожній ітерації виробляються модифікації дизайну і додаються нові функціональні можливості. Основна ідея цього методу полягає в розробці системи через повторні цикли (ітеративні) і меншими порціями за один раз (інкрементні).

1.3.8 Спіральна модель

Бем модифікував модель водоспаду в 1986 році, впровадивши кілька ітерацій, що виходять із малих почав. Часто цитований ідіом, що описує філософію, що лежить в основі спірального методу, починається з малого, великого [8]. На рисунку 1.8 зображена спіральна модель.

Модель спіралі являє собою комбінацію як ітеративної моделі, так і однієї моделі ЖЦПЗ. Це можна побачити, як якщо б ви обрали одну модель ЖЦПЗ і об'єднали її з циклічним процесом (ітеративна модель).

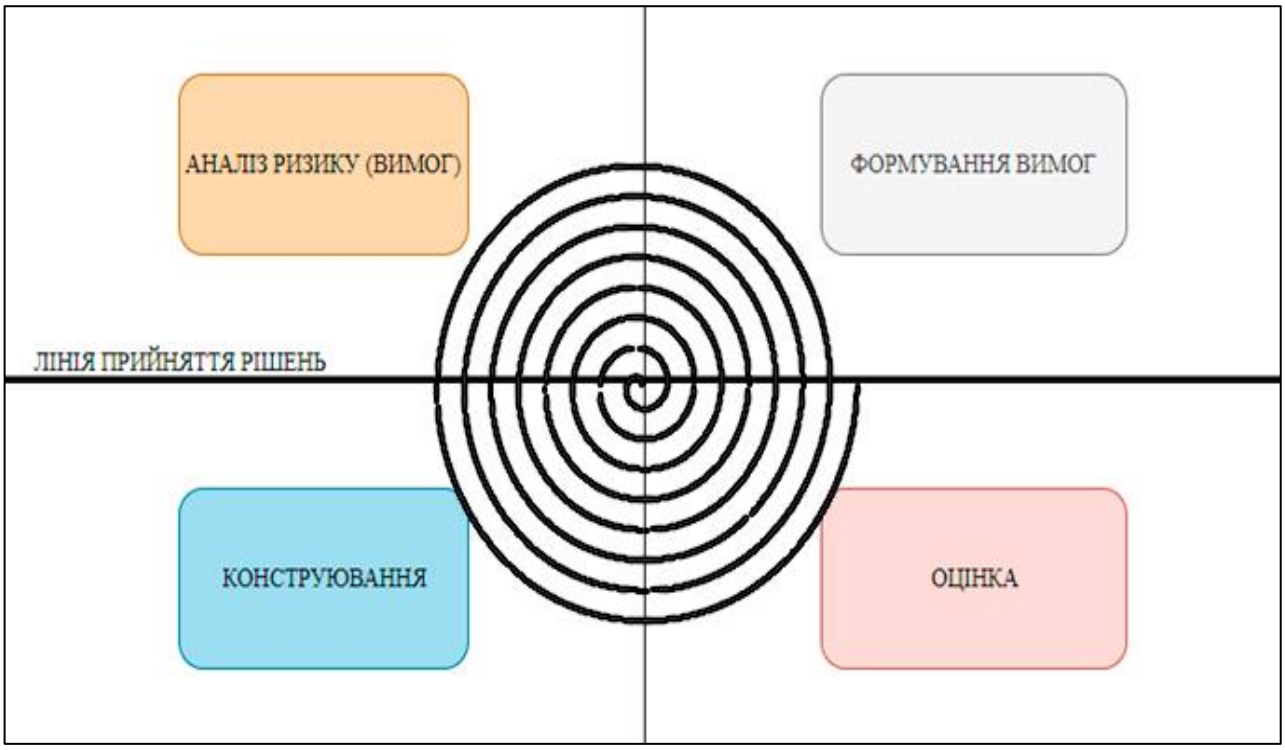

Рисунок 1.8 – Спіральна модель

Ця модель розглядає ризик, який часто не помічається більшістю інших моделей. Модель починається з визначення цілей і обмежень програмного забезпечення на початку однієї ітерації. Наступним етапом є створення прототипів програмного забезпечення. Це включає аналіз ризиків. Тоді для побудови програмного забезпечення використовується одна стандартна модель ЖЦПЗ. На четвертому етапі готується план наступної ітерації.

1.4 Постановка задачі

Відповідно до аналізу предметної галузі, вибір моделі життєвогго циклу програмного забезпечення є найвижливішим кроком у створенні програмного продукту, саме тому аналіз цих моделей щодо вибору найбільш ефективної є актуальним.

Таким чином, необхідно вирішити наступні задачі:

– провести детальний аналіз предметної галузі, що пов’язаний з дослідженням моделей життєвого циклу програмного забезпечення;

– дослідити існуючі моделі життєвого циклу програмного забезпечення та проаналізувати, за яких умов треба використовувати ту чи іншу модель;

– створити програмний додаток, який має обробляти велику вхідну кількість даних про ведення програмного забезпечення та за допомогою методів машиного вивчення аналізувати и показувати найбільш ефективну модель для конкретної ситуації.

Програмний додаток буде націлен на ринок ведення розробки програмного продукту.

Даний програмний продукт буде конкурентоспроможним за допомогою зручного та простого інтерфейсу, матиме цікавий і корисний функціонал, використовуватиме актуальніші технології розробки.

Даний продукт буде вирішувати наступні проблеми:

– відсутність можливості обробки великої кількості входних даних про ведення програмного забезпечення;

– відсутність аналізування даних за допомогою методів машиного вивчення.

Головними та практично унікальними функціями будуть:

– реєстрація та авторизація користувачів;

– сворення та редагування історії ведення продукту (інформація про штат робітників, стартовий та накопичувальний капітал, збір вимог, виявлення дефектів, розробку плану, вимог, кодування та проектування);

– аналіз найфектевніших моделей життєвого циклу програмного забезпечення;

– виявлення моделі життєвого циклу для кожного проекту за використування даних та інформації, яка отримаються від дій користувача в системі;

– широка область застосування за рахунок величезного функціоналу.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОГО ПРОДУКТУ

Даний програмний продукт повинен бути розроблений, як вже зазначалося вище, на базі клієнт-серверної архітектури. У якості ядра повинен виступати головний сервер з Ruby on Rails додатком [9], який приймає запити та має логіку подальшої роботи з діями користувача.

У якості базового набору необхідно створити два сервери бази даних, які мають однакову бізнес-логіку, але виконуються одночасно для вирішування проблеми малої швидкодії. Є головний сервер бази даних, який вирішує, до якої дочірньої бази треба звернутися. Є два постійних сервера обробки запитів користувача, які запущені одночасно та теж вирішують проблеми швидкодії. З ними взаємодіє головний сервер, який отримує запити користувача та вирішує, до якого дочірнього серверу звернутися.

База даних повинна зберігати інформацію про користувачів, сесії авторизації, списки проектів, інформацію для кожного проекту, як штаб розробників, наявний капітал, збір вимог, дефекти, зроблені задачі та інше. Для можливості працювати з основним функціоналом для кожного користувача необхідна система авторизації.

Фреймворк Ruby on Rails реалізує досить популярний шаблон Model-View-Controller, який дозволяє представити роботу серверної частини у вигляді окремих модулів, кожен з яких відповідає за окрему функцію та має свою мету. У якості модулю View повинен бути обран фреймворк для клієнтської взаємодії Angular.

На сайті необхідно мати наступні інформаційні сторінки:

– початкова сторінка з базовою інформацією про програмний продукт;

– сторінка реєстрації користувачів;

– сторінка автентифікації, за допомогою якої користувач може зайти в систему;

– сторінка зі списком поточних проектів;

– сторінка створення та редагування основної інформації про проект;

– сторінка створення та редагування ресурсу проекту, як поточні задачі, збір вимог, планування, тестування, штаб розробників, дефекти;

– сторінка з аналізом проекту щодо вибору найефективнішої моделі життєвого циклу програмного забезпечення;

– сторінка з аналізом проекту щодо поточної моделі життєвого циклу програмного забезпечення.

Сайт повинен безпосередньо працювати з сервером API, постійно звертаючись до нього й отримуючи необхідну інформацію. База даних винесена на окремий сервер з метою зменшення навантаження на основний веб-сервер та покращення продуктивності. Також на сервер баз даних винесена основна бізнес-логіка системи, що дозволить відокремити її від відображення та моделі та покращить масштабованість.

Даний програмний продукт позбавить від надлишкового пошуку однієї й тієї ж інформації на декількох сайтах.

За допомогої cron-задач сервер з API повинен відправляти великий потік даних, згрупований по кожному проекту [10], до серверу, на якому буде знаходитись Big Data обробник та за допомогою моделей машиного вивчення буде створюватися аналіз для кожного проекту щодо його поточної моделі життєвого циклу програмного забезпечення та найефективнішої моделі життєвого циклу програмного забезпечення.

Моделі машиного вивчення повинні бути ініційовані на основі коректних початкових даних з точною інформацією про кожну модель життєвого циклу програмного забезпечення та використовувати алгоритми регресійного аналізу в навчанні з учителем та класифікаційні задачі з учителем.

Класифікаційні задачі повинні використовуватись для виявлення поточної моделі життєвого циклу програмного забезпечення. Регресійний аналіз повинен використовуватись для виявлення найфективнішої моделі життєвого циклу програмного забезпечення.

До даного програмного забезпечення висуваються наступні системні вимоги:

– для розгортання веб-сайту необхідна машина зі встановленою операційною системою Ubuntu 18.04 зі встановленим Ruby та Ruby on Rails [11];

– на машині повинен бути встановлений сервер nginx;

– для розгортання баз даних необхідний встановлена СУБД PostgreSQL;

– обсяг оперативної пам’яті повинен бути не меншим ніж 2 Гб;

– обсяг дискового простору повинен бути не меншим ніж 25 Гб.

До даного продукту висуваються наступні функціональні вимоги: продукт повинен зберігати та опрацьовувати персональні дані користувачів, дані, які стосуються інформації про групи фахівців, чати з повідомленнями та документами для доступу всіх фахівців та календар подій. Продукт повинен опрацьовувати й аналізувати збережені дані для того, щоб представляти основні функції та можливості системи. Також продукт повинен гарантувати цілісність і надійність збереження користувальницьких даних, захищати їх авторське право та інтелектуальну власність. Також продукт повинен представляти можливість здійснення контролю за веденою інформацією, за його правильністю, актуальністю та змістом, але не повинен робити це автоматично. Продукт повинен дозволяти виконувати користувачам свої основні операції та функції, не викликаючи при цьому якісь проблеми, підозри або недовіри зі сторони користувачів.

Серед нефункціональних вимог до даного проекту варто віднести наступне: безпека системи, надійність, швидкодія, зручність у використанні, масштабування, відновлюваність. Система повинна представляти та гарантувати безпеку персональних даних користувачів та гарантувати захист авторського права та інтелектуальної цілісності. Також система повинна мати досить високу швидкодію та мати невисокий термін відклику на якісь дії користувачів. Продукт повинен бути зручним у використанні та відповідати принципам «Usability» інтерфейсів.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проектування програмного забезпечення

UML (Unified Modeling Language – уніфікована мова моделювання) – це мова візуального моделювання, яка розроблена для візуалізації, проектування та документування компонентів програмного забезпечення. UML дозволяє також досягти угоди в графічних позначеннях для подання загальних понять (таких як клас, компонент, узагальнення (generalization), об'єднання (aggregation) і поведінка) і орієнтована на проектування та архітектуру програмного продукту [12].

UML є досить потужним інструментом моделювання, який може бути ефективно використаний у побудові концептуальних, логічних і графічних моделей складних систем різного цільового призначення. Ця мова увібрала в себе найкращі якості і досвід методів програмної інженерії, які з успіхом використовувалися впродовж останніх років при моделюванні великих і складних систем.

Діаграма прецедентів описує основні дії користувача системи.

Користувач починає працювати зі сервісом з ознайомлення початкової інформації, необхідної для роботи з поточною системою ведення програмних продуктів.

Користувач починає роботу зі системою з автентифікації. Спочатку він проходить реєстрацію, де він повинен зареєструватись за допомогою поштової скриньки. Далі користувач може війти в систему під своїм логіном, отриманим після реєстрації.

Основний функціонал користувача пов'язаний з роботою над проектами, де він може переглядати, створювати, редагувати та видаляти. Для кожного проекту користувач має змогу створювати, переглядати, редагувати та видаляти ресурси проекту, які складаються з величезної кількості артефактів. Артефакти бувають таких типів, як планування, вимог, конструювання (дизайн та розробка), тестування (оцінка) та розгортання.

Побудуємо діаграму, виділивши при цьому основні дії користувача в системі та зв’язок користувача та системи загалом. На рисунку 3.1 зображена діаграма прецендентів.

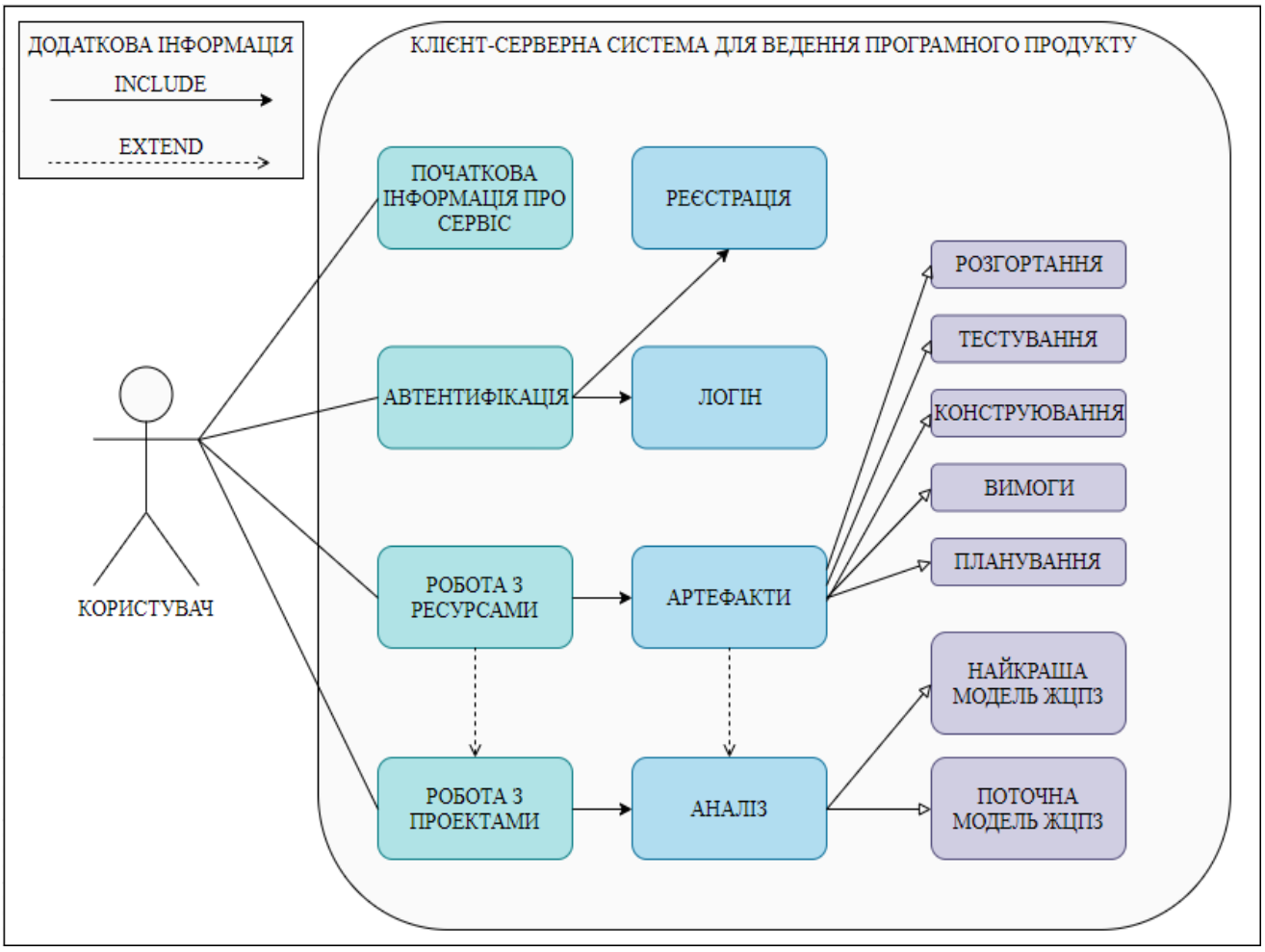

Рисунок 3.1 – Діаграма прецендентів (Use Case)

Користувач може проаналізувати кожен проект завдяки отримання інформації про вибрану поточну модель життєвого циклу програмного забезпечення та переглядати найефективнішу модель життєвого циклу програмного забезпечення.

Діаграма послідовності відображає взаємодії об’єктів впорядкованих за часом. Побудуємо діаграму для операцій читання і запису від клієнта в Hfile. На рисунку 3.2 зображена діаграма послідовності.

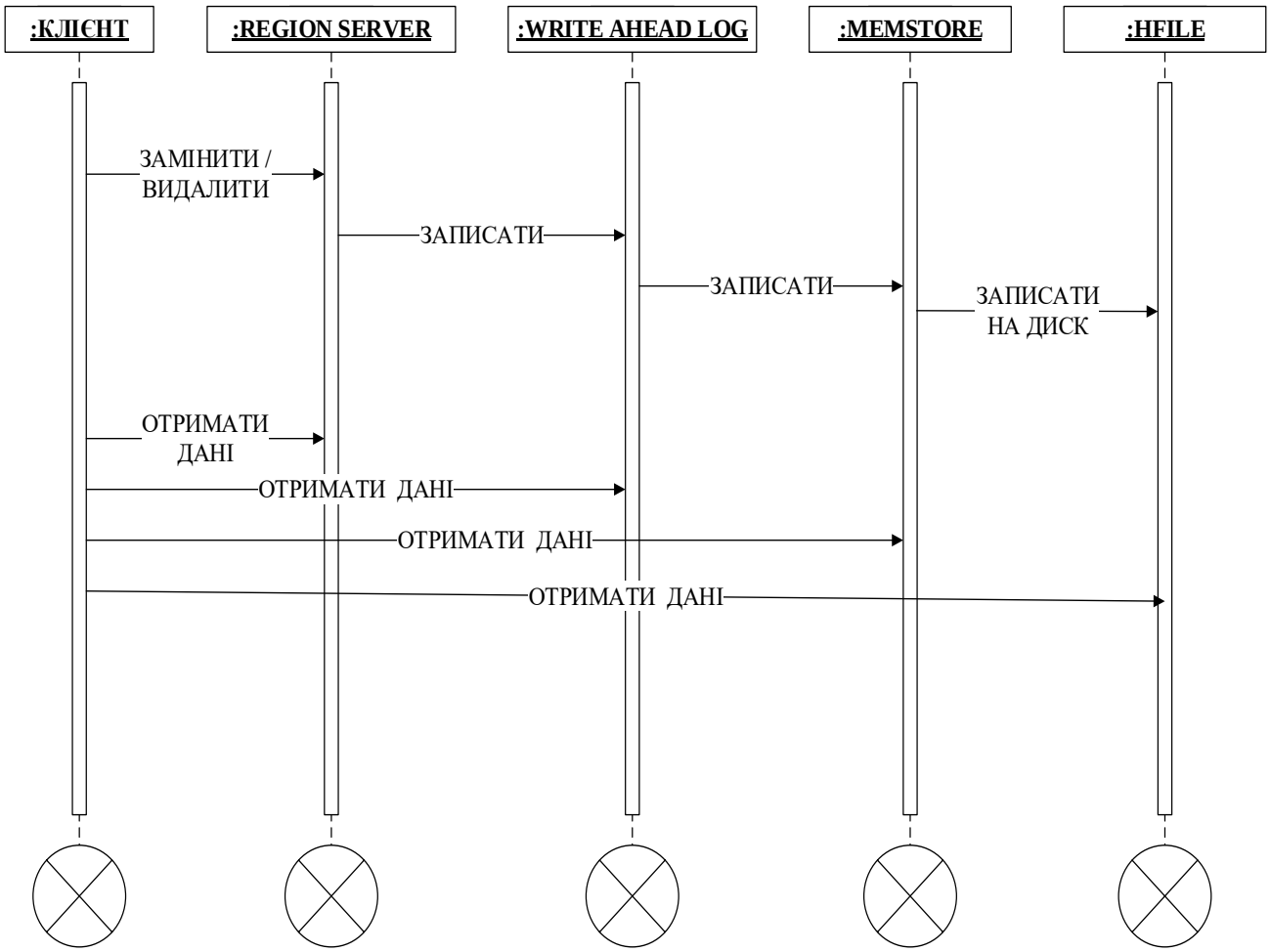

Рисунок 3.2 – Діаграма послідовності

Клієнт хоче записувати дані і в свою чергу спочатку спілкується з сервером Regions, а потім регіонами. Регіони контактують з memstore для зберігання даних, пов'язаних з сімейством стовпців. Спочатку дані зберігаються в Memstore, де дані сортуються і після цього він скидається в HFile. Основною причиною використання Memstore є зберігання даних у розподіленій файловій системі на основі Row Key. Memstore буде поміщено в основну пам'ять сервера регіону, а HFiles – у HDFS [13]. Клієнт хоче прочитати дані з регіонів. У свою чергу клієнт може мати прямий доступ до пам'яті Mem, і він може запитувати дані. Клієнт підходить до HFiles для отримання даних. Дані отримуються та витягуються Клієнтом.

Діаграма кооперації відображає взаємодію між частинами композитної структури та ролями. Побудуємо діаграму для перегляду користувачем поточної моделі життєвого циклу програмного забезпечення. На рисунку 3.3 зображена діаграма кооперації.

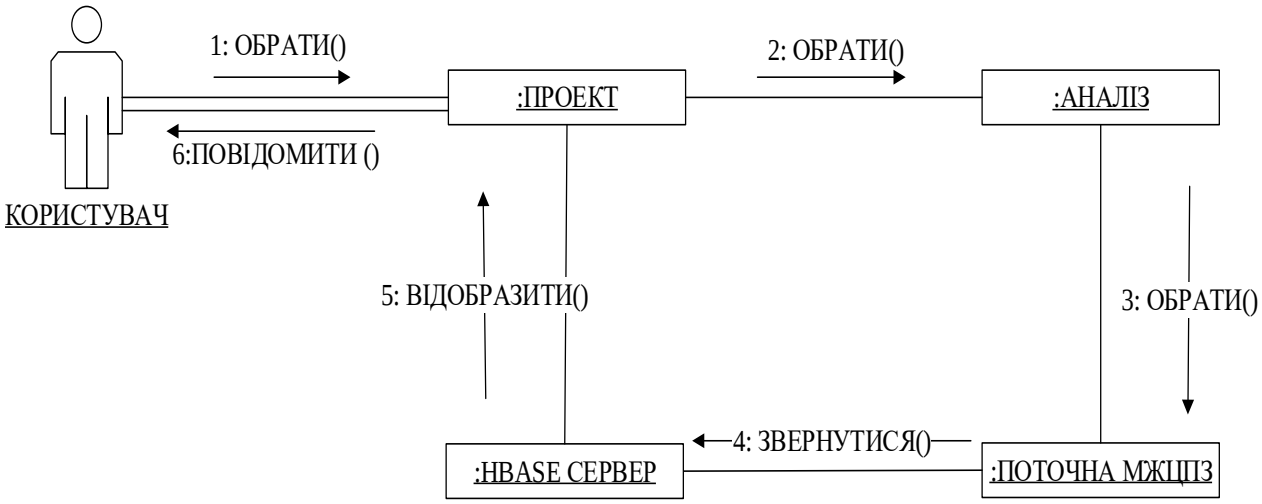

Рисунок 3.3 – Діаграма кооперації

Користувач обирає необхідний проект, для якого обирає аналіз. Далі користувач обирає аналіз поточної моделі життєвого циклу програмного забезпечення, яка звертається до Hbase серверу, який відображає інформацію на проекті.

3.2 Проектування архітектури програмного забезпечення

Архітектура програмного забезпечення – це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Архітектура є безліччю структур, необхідних для міркування про програмній системі, і включає елементи системи, зв'язку між ними і властивості цих елементів і зв'язків [14].

Архітектура «клієнт-сервер» визначає загальні принципи організації взаємодії в мережі, де є сервери, вузли-постачальники деяких специфічних функцій (сервісів) і клієнти, споживачі цих функцій. На рисунку 3.4 зображена клієнт-серверна архітектура.

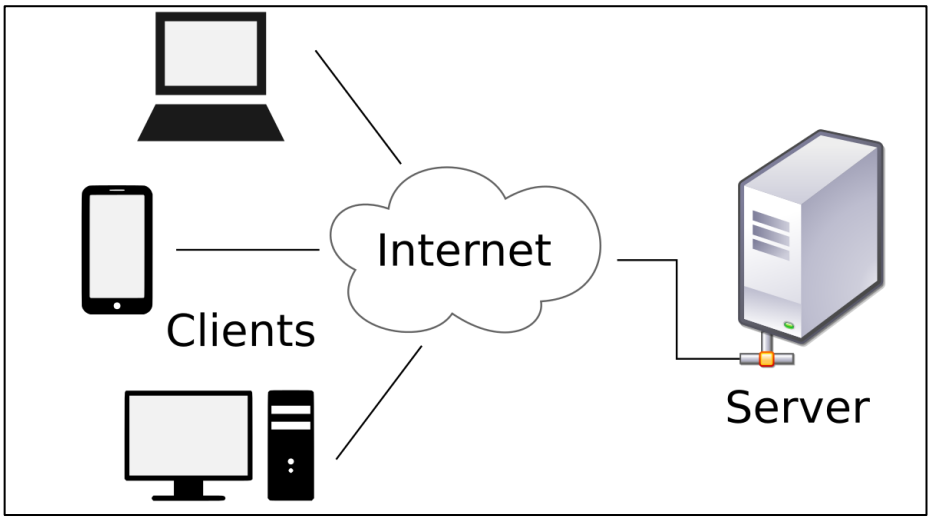

Рисунок 3.4 – Клієнт-серверна архітектура

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосувань і передбачає взаємодію та обмін даними між ними [15].

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

За допомогою діаграми можна побачити, що користувач має змогу виконувати дії з браузера. Браузер відправляє запити на отримання таких статичних файлів, як HTML, CSS та JS формату задля рендерінгу клієнта. Далі користувач за допомогою інтерфейсу клієнта відправляє запроси до API серверу по HTTP або HTTPS. API сервер має власний сервер бази даних PostgreSQL [16], який може знаходитись на тому ж сервері, або на іншому. API сервер взаємодіє з сервером фонових задач Sidekiq, який відправляє та отримує асінхронно дані з Apache Hbase серверу, на якому знаходиться логіка машиного навчання.

На рисунку 3.5 зображена діаграма розгортання.

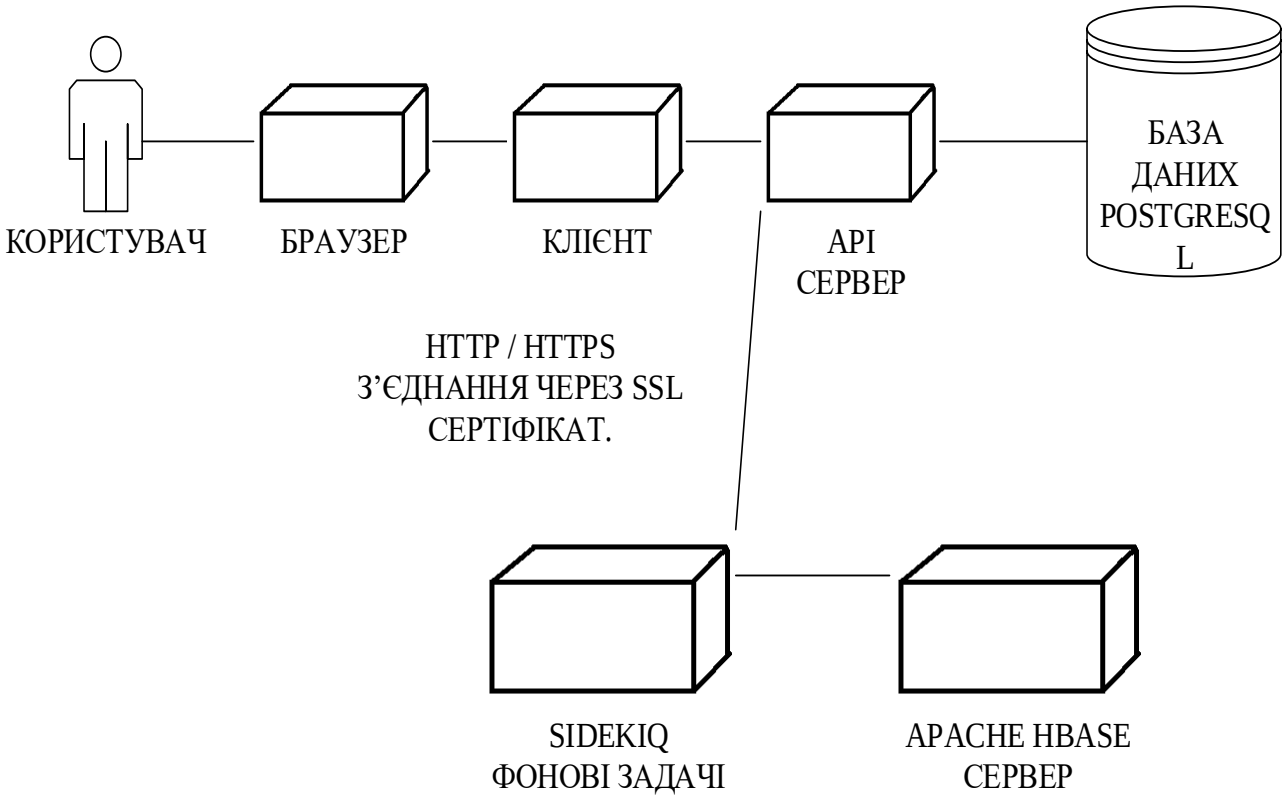

Рисунок 3.5 – Діаграма розгортання

Діаграма компонентів, що також розроблена з використанням UML нотації, відображує розбиття програмної системи на структурні компоненти та зв'язок між компонентами. Побудуємо діаграму для нашою системи. Модульні програми пропонують інший спосіб роботи над додатком. За допомогою створення невеликих компонентів багаторазового використання замість одного великого додатку досягається просте додавання або вилучання коду. Маючи набір компонентів, можна мати кілька додатків, спільне використання деяких функцій, але маючи свою специфіку. На рисунку 3.6 зображена діаграма компонентів.

Розроблюваний сервіс також відповідає архітектурному стилю REST.

REST – підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Найбільш відомою системою, заснованою на архітектурі REST, є сучасний World Wide Web. Дані повинні передаватися в невеликій кількості стандартних форматів (наприклад, HTML, XML, JSON). Мережевий протокол (наприклад, HTTP) повинен підтримувати кешування, не

повинен залежати від мережевого рівня, не повинен зберігати інформацію про стан пар «запит-відповідь».

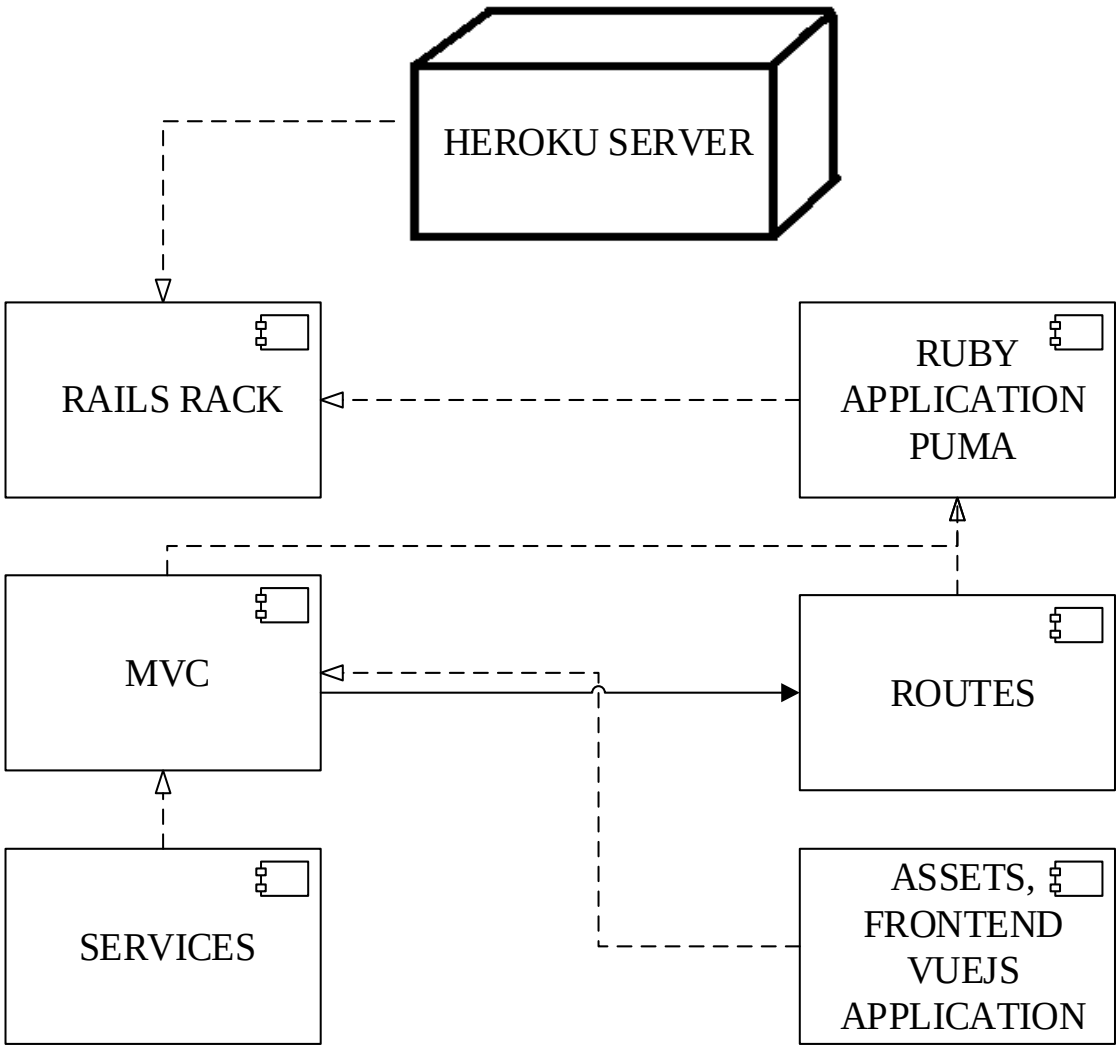

Рисунок 3.6 – Діаграма компонентів

Стверджується, що цей підхід забезпечує масштабованість системи і дозволяє їй розвиватися з новими вимогами.

3.3 Проектування бази даних

Для зберігання даних системи необхідно розробити реляційну базу даних. Реляційна база даних – це сукупність взаємопов'язаних таблиць, кожна з яких

містить інформацію про об'єкти певного типу. Рядок таблиці містить дані про один об'єкт, а стовпці таблиці описують різні характеристики об'єктів – атрибути. Рядки таблиці, мають однакову структуру вони складаються з полів, що зберігають атрибути об'єкта. Кожне поле стовпця, описує тільки одну характеристику об'єкта і має строго певний тип даних. Усі записи мають одні і ті ж поля, тільки в них відображаються різні інформаційні властивості об'єкта.

Під час концептуального моделювання було виявлено основні об’єкти системи та їх атрибути. Основні сутності складаються з «Користувач», «Сесія», «Проект», «Користувач проекту», «Ресурс», «Статус ресурсу», «Модель», «Група виявлення поточної моделі», «Група виявлення найефективнішою моделі», «Поточна модель» та «Найефективніша модель». Також на етапі моделювання були виявлені зв’язки між цими сутностями.

Також на етапі моделювання було виявлено основні атрибути, які мають містити в собі ці сутності. Користувач має зберігати таку основну інформацію, як електронний адрес, пароль, завдяки якому він буде входити в систему, та ім’я.

Сесія буде описувати активний вхід користувача та повинна зберігати токен входу.

Проект має мати назву. Кожний проект може буде закріпленим до кожного користувача завдяки ролі на проекті.

Ресурс має зберігати тип, назву, значення. Кожний ресурс має статуси зі значенням та часовою відміткою.

Модель зберігає назву.

Кожний проект має групи найефективніших та поточних моделей життєвого циклу програмного забезпечення з оцінками.

Основними таблицями в розроблюваній системі є:

– користувач;

– сесія;

– проект;

– користувач кроекту;

– ресурс;

– статус ресурсу;

– група виявлення поточної моделі життєвого циклу програмного забезпечення;

– група виявлення найефективнішої моделі життєвого циклу програмного забезпечення;

– поточна модель життєвого циклу програмного забезпечення;

– найефективніша модель життєвого циклу програмного забезпечення;

– модель життєвого циклу програмного забезпечення.

На основі проведеного аналізу побудовано схему основних таблиць бази даних у третій нормальній формі. На рисунку 3.7 зображена діаграма класів.

В результаті концептуального моделювання було встановлено, що таблиця «Користувач» має наступні атрибути:

– ідентифікатор користувача;

– дату зроблення;

– дату останнього оновлення;

– поштову скриньку;

– ім’я;

– пароль.

Сутність «Сесія» має наступні атрибути:

– ідентифікатор сесії;

– дату зроблення;

– дату останнього оновлення;

– токен аутентифікації;

– ідентифікатор користувача.

Сутність «Проект» має наступні атрибути:

– ідентифікатор проекту;

– дату зроблення;

– дату останнього оновлення;

– назву.

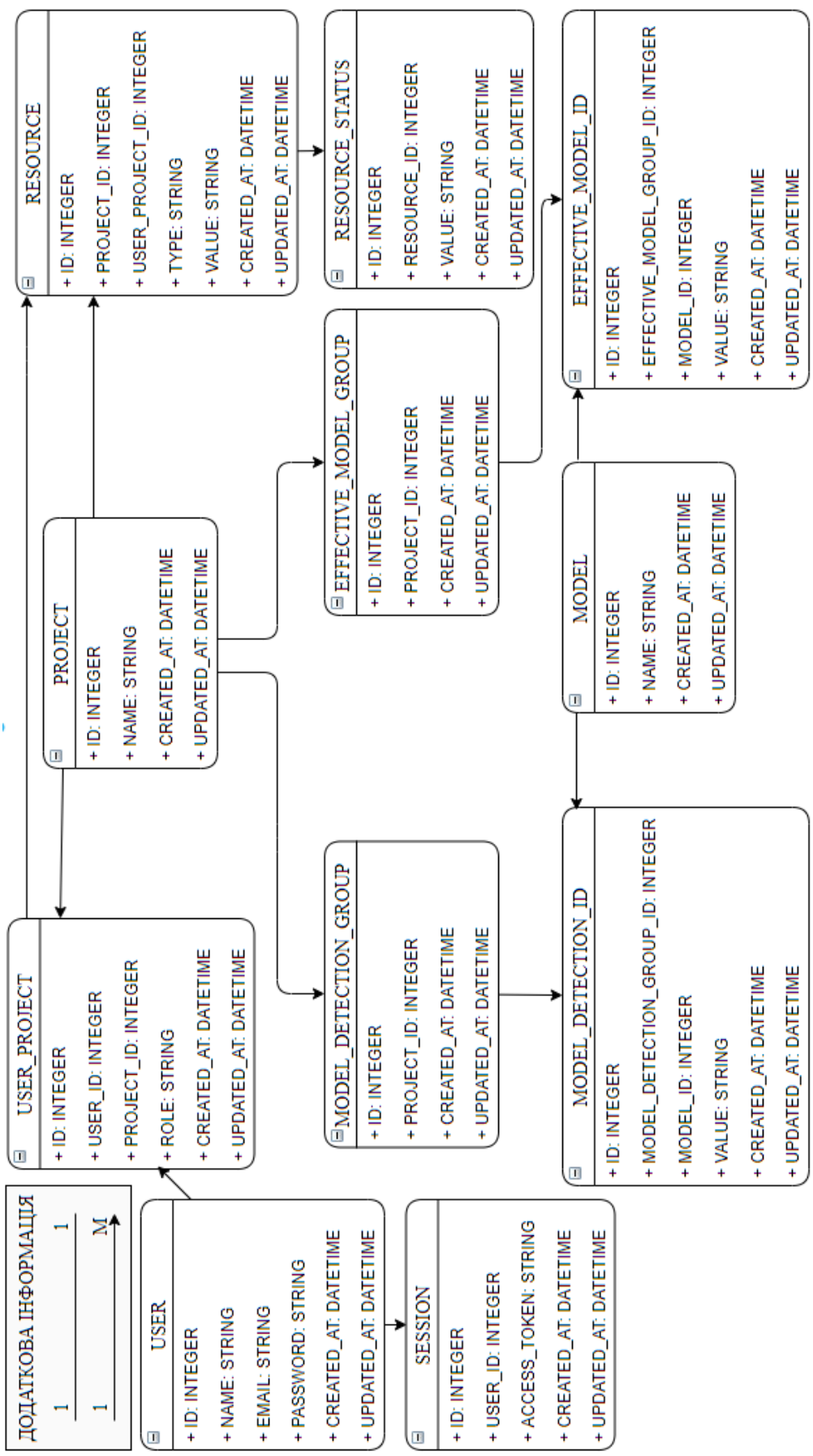

Рисунок 3.7 – Діаграма класів

Сутність «Користувач проекту» має наступні атрибути:

– ідентифікатор користувача проекту;

– дату зроблення;

– дату останнього оновлення;

– роль користувача;

– ідентифікатор проекту;

– ідентифікатор користувача.

Сутність «Ресурсу» має наступні атрибути:

– ідентифікатор ресурсу;

– дату зроблення;

– дату останнього оновлення;

– тип ресурсу;

– значення ресурсу;

– ідентифікатор проекту;

– ідентифікатор користувача проекту.

Сутність «Статуса ресурсу» має наступні атрибути:

– ідентифікатор статуса ресурсу;

– дату зроблення;

– дату останнього оновлення;

– значення;

– ідентифікатор ресурсу.

Сутність «Моделі» має наступні атрибути:

– ідентифікатор моделі;

– дату зроблення;

– дату останнього оновлення;

– назву.

Сутність «Групи виявлення поточної моделі» має наступні атрибути:

– ідентифікатор групи;

– дату зроблення;

– дату останнього оновлення;

– ідентифікатор проекту.

Сутність «Групи виявлення найефективнішої моделі» має наступні атрибути:

– ідентифікатор групи;

– дату зроблення;

– дату останнього оновлення;

– ідентифікатор проекту.

Сутність «Поточної моделі» має наступні атрибути:

– ідентифікатор поточної моделі;

– дату зроблення;

– дату останнього оновлення;

– оцінку;

– ідентифікатор групи;

– ідентифікатор моделі.

Сутність «Найефективнішої моделі» має наступні атрибути:

– ідентифікатор найефективнішої моделі;

– дату зроблення;

– дату останнього оновлення;

– оцінку;

– ідентифікатор групи;

– ідентифікатор моделі.

Діаграма об’єктів відображає сукупність множини об’єктів та відношень між ними в дейкий час. На основі отриманою структури бази даних побудуємо діаграму.

Діаграми об'єктів представляють статичний вид системи з точки зору проектування і процесів, будучи основою для сценаріїв, описуваних діаграмами взаємодії. Діаграма об'єктів використовується для пояснення і деталізації діаграм взаємодії, наприклад, діаграм послідовностей.

На рисунку 3.8 зображена діаграма об’єктів.

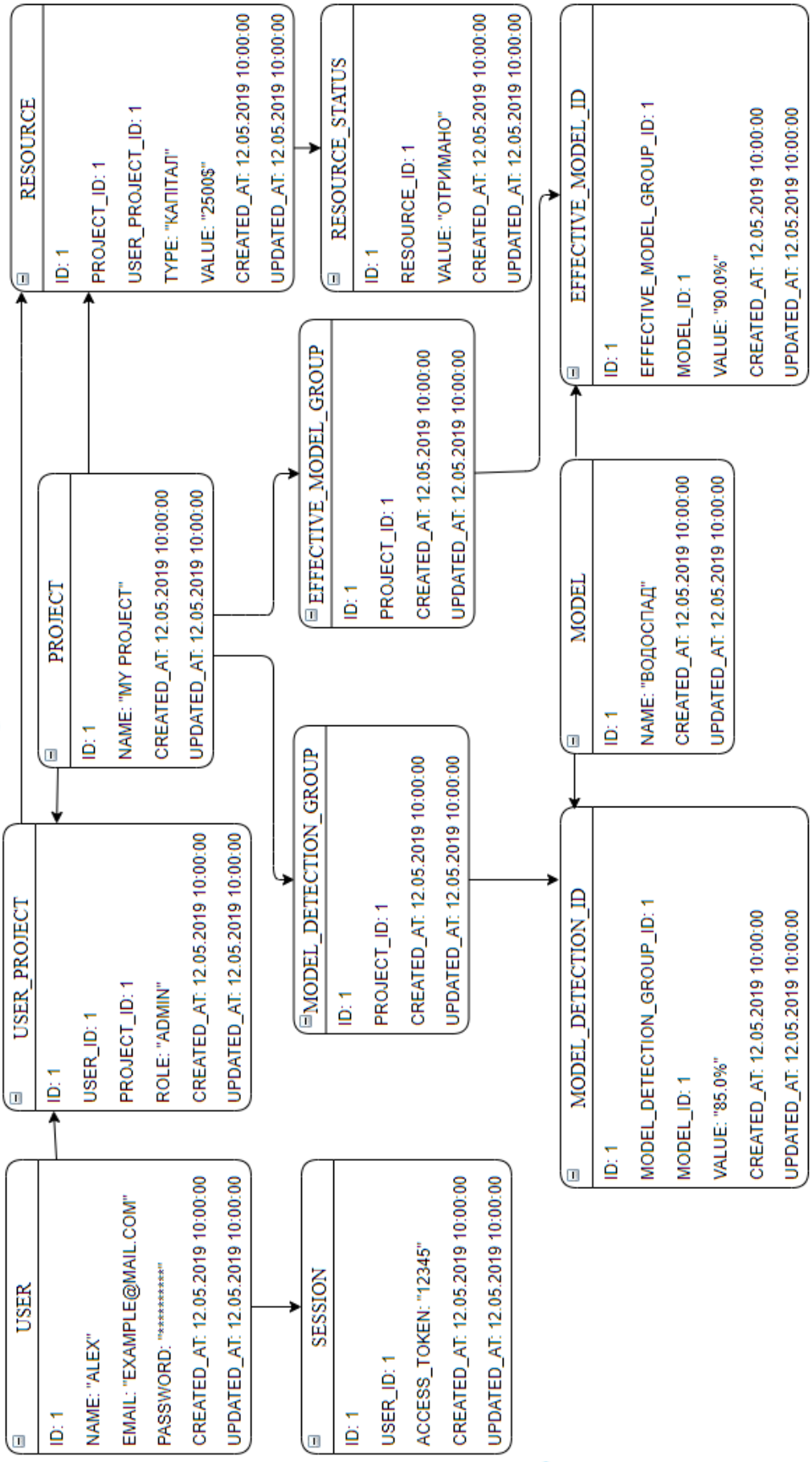

Рисунок 3.8 – Діаграма об'єктів

Діаграми об'єктів зручні для показу прикладів пов'язаних один з одним об'єктів. У багатьох ситуаціях точну структуру можна визначити за допомогою діаграми класів, але при цьому структура залишається важкою для розуміння. У таких випадках пара прикладів діаграми об'єктів може прояснити ситуацію.

3.4 Створення UI / UX або іншого дизайну системи

Для сучасного інтернет проекту UX-дизайн відіграє важливу роль і визначає успішність його розвитку. Враховувати його ключові моменти потрібно на всіх етапах розробки. UX дизайн (User eXperience Design) – застосування методів проектування, спрямованих на досягнення максимально ефективної взаємодії користувача з системою [17].

Розбролювана система має наступні вимоги до UX-дизайну:

– landing-page, клієнт взаємосв’язан із сервером за допомогою AJAX, тому повинен мати одну сторінку, яка не потребує перезавантаження для оновлення інформації: вона оновлюється частково асінхронно;

– користувач має змогу легко знайти потрібний йому елемент на сайті;

– зверху сайта повинно розтошовуватися меню;

– знизу сайта повинно розтошовуватися часткове меню;

– контент сайту повинен змінюватися, коли користувач перемикається по посиланням.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

Під час вибору технологій для реалізації цього проекту було проаналізовано багато сучасних мов програмування та фреймворків і були обрані найактуальніші з них. Тож було вирішено використати об’єктно-орієнтовану мову програмування Ruby та фреймворк Ruby on Rails для серверної сторони, JavaScript з Angular фреймворком для клієнтської сторони, Hbase сховище та методи машиного навчання зі вчителем з задачею класифікації та задачею регресійного аналізу.

4.1 Серверна частина

Ruby on Rails (RoR) є веб-фреймворком з відкритим кодом, написаним на мові програмування Ruby. Rails орієнтований на продуктивність і забезпечує гнучкість у веб-розробці.

Ruby прийшов дуже добре з дизайном проекту, тому що це динамічно типізована, повністю об'єктно-орієнтована універсальна мова сценаріїв. Ruby був розроблений для створення складного синтаксису.

Структура Rails розроблена для баз даних підтримуваних веб-додатків. Вона була створена у відповідь на важкі рамки, такі як J2EE і .NET. Для того, щоб зробити процес розвитку швидшим, Rails використовує угоди і припущення, які розглядають найкращі способи виконання завдань, і вони покликані заохочувати їх. Ця транзакція виключає код конфігурації та покращує продуктивність. Багато загальних завдань для веб-розробки вбудовані в роботу поза коробкою. Це включає в себе управління електронною поштою, картографічні об'єкти-бази даних, файлові структури, генерацію коду, газоподібні елементи, їх організацію тощо [18].

Після розробки основної частини ми маємо конфігураційну структуру клієнт-серверної системи. На рисунку 4.1 зображена структура системи.

Рисунок 4.1 – Структура системи

До основних модулів слід віднести API, ассети, фонові роботи, поштовий відправник, моделі з бізнес-логікою, серіалайзери JSON типу, конфігурація бази даних, роутінг, конфігурація кожної середи розробки, тести, Gemfile з усіма бібліотеками розробленої системи, конфігураціоний файл rubocop, який відповідає за чистий та форматований код.

В якості СУБД було вирішено використовувати PostgreSQL. Ruby on Rails фреймворк створює схему з міграціями. Це інформація про створені таблиці, види та назви полів, первинні та вторинні ключі. На рисунку 4.2 зображена схема БД.

```ruby
ActiveRecord::Schema.define(version: 2019_05_12_150941) do

  # These are extensions that must be enabled in order to support this database
  enable_extension name "plpgsql"

  create_table "detection_models", force: :cascade do |t|
    t.bigint "model_detection_group_id"
    t.bigint "model_id"
    t.string "value"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index column_name ["model_detection_group_id"], name: "index_detection_models_on_model_detection_group_id"
    t.index column_name ["model_id"], name: "index_detection_models_on_model_id"
  end

  create_table "effective_model_groups", force: :cascade do |t|
    t.bigint "project_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index column_name ["project_id"], name: "index_effective_model_groups_on_project_id"
  end

  create_table "effective_models", force: :cascade do |t|
    t.bigint "effective_model_group_id"
    t.bigint "model_id"
    t.string "value"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index column_name ["effective_model_group_id"], name: "index_effective_models_on_effective_model_group_id"
    t.index column_name ["model_id"], name: "index_effective_models_on_model_id"
  end

  create_table "model_detection_groups", force: :cascade do |t|
    t.bigint "project_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index column_name ["project_id"], name: "index_model_detection_groups_on_project_id"
  end

  create_table "models", force: :cascade do |t|
    t.string "name"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end
```

Рисунок 4.2 – Схема БД

Для планувальника фонових задач був обраний Sidekiq.

Даний гем дозволяє rails породжувати нові процеси, які будуть працювати у фоновому режимі. Це дуже зручно для обробки громіздких завдань (наприклад, зчитування даних з великого документа і занесення їх в базу даних), щоб користувач отримав відповідь не чекаючи, поки виконається вся робота, що робить додаток більш чуйним. На рисунку 4.3 зображен інтерфейс планувальника фонових задач.

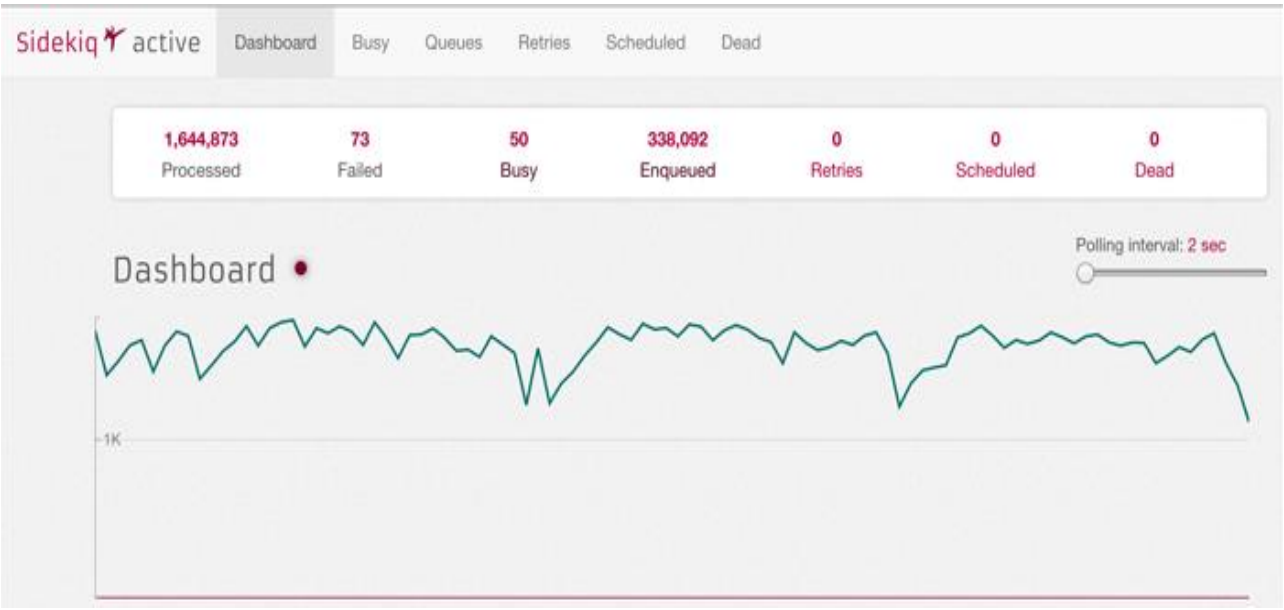

Рисунок 4.3 – Інтерфейс планувальника фонових задач

Sidekiq Enterprise підтримує періодичні завдання. Інші дорогоцінні камені третьої сторони також пропонують крон-подібну функціональність.

4.2 Клієнтська частина

Мовою клієнтської сторони був обран JavaScript, тому що він зазвичай використовується як вбудована мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додання інтерактивності веб-сторінок.

Клієнтським фреймворком був обраний Angular. Angular – це фреймворк, який дозволяє легко створювати програми з Інтернету. Angular поєднує в собі декларативні шаблони, ін'єкції залежностей, кінцеві інструменти та інтегровані найкращі практики для вирішення завдань розвитку. Angular надає розробникам можливість створювати програми, які працюють у мережі, на мобільних пристроях або на робочому столі.

Структура проекту клієнтської частини для цього проекту поєднує контролери, сервіси та рендерінг. На рисунку 4.4 зображена структура проекту.

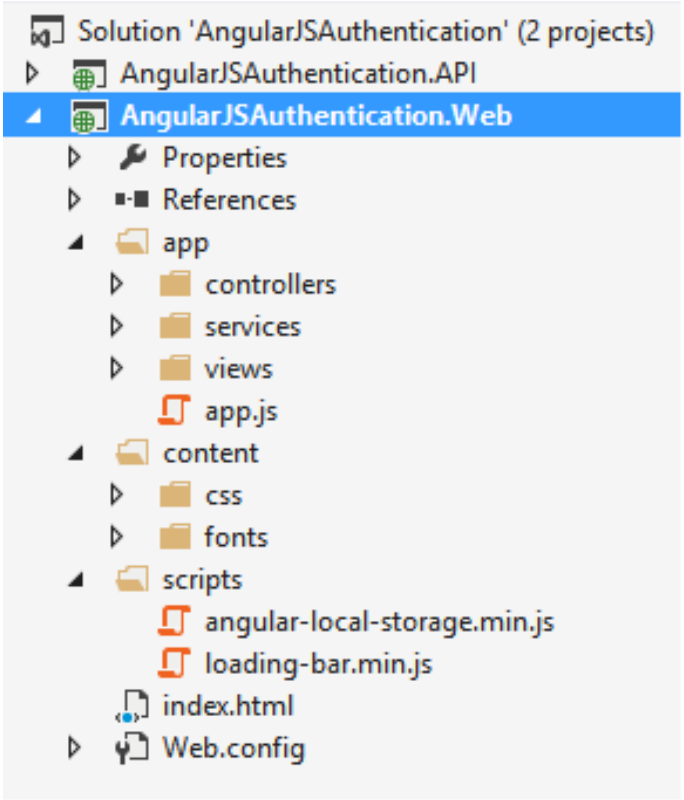

Рисунок 4.4 – Структура проекту

Ви розробляєте програми в контексті робочого простору Angular. Робоче середовище містить файли для одного або декількох проектів. Проект – це набір файлів, які складаються з окремої програми, бібліотеки або набору тестів e2e.

4.3 Модуль машиного навчання

Машинне навчання – це сфера обчислювальної науки – часто вкладена під дослідження AI – з безліччю практичних застосувань, завдяки здатності алгоритмів, що вийшли, систематично реалізувати конкретне рішення без явних інструкцій програміста. Очевидно, що багато алгоритмів потребують визначення

особливостей, на які слід звернути увагу, або великого набору даних для виведення рішення [19].

Спочатку потрібно запрограмувати базові моделі, на яких буде тренуватись сервер з машиним навчанням. На рисунку 4.5 зображені треновані моделі.

```
# frozen_string_literal: true

require 'classifier-reborn'

models = [
  ['spiral', %w[r_10 c_10 t_5 e_0.5 r_10 c_10 t_5 e_0.5 r_10
                c_10 t_5 e_0.5 d_5]],
  ['iterative', %w[r_1 r_2 c_2 t_1 e_0.5 r_3 c_3 t_1.5 e_0.5 r_1 c_1 t_0.5 e_0.5
                   r_4 c_4 t_2 e_0.5 r_4 c_4 t_2 e_0.5 r_2 c_2 t_1 e_0.5 r_3 c_3
                   t_1.5 e_0.5 r_1 c_1 t_0.5 e_0.5 r_4 c_4 t_2 e_0.5 d_0]],
  ['rad', %w[r_5 r_2 c_0.5 t_0.25 r_6 c_1.5 t_0.75 r_4 c_1 t_0.5 r_8 c_2 t_1 r_8
             c_2 t_1 r_2 c_0,5 t_0.5 r_6 c_1.5 t_0.75 r_4 c_1 t_0.5 r_8 c_2 t_1
             d_2.5]],
  ['inremental', %w[r_1 c_1 t_0.5 e_0.5 r_3 c_3 t_1.5 e_0.5 r_2 c_2 t_1 e_0.5
                    r_4 c_4 t_2 e_0.5 c_4 t_2 e_0.5 t_1 c_1 t_0.5 e_0.5 r_3 c_3
                    t_1.5 e_0.5 r_2 c_2 t_1 e_0.5 r_4 c_4 t_2 e_0.5 d_5]],
  ['v-model', %w[r_10 t_10 c_10 t_5 c_4 t_5 r_10 t_10 c_10 t_5 d_5]],
  ['waterfall', %w[r_10 c_10 t_5 c_4 t_5 r_10 c_10 t_5 d_5]],
  ['agile', %w[r_1 r_2 c_2 t_1 e_0.5 r_3 c_3 t_1.5 e_0.5 r_1 c_1 t_0.5 e_0.5
               r_4 c_4 t_2 e_0.5 r_4 c_4 t_2 e_0.5 r_1 c_1 t_1 e_0.5 r_1.5 c_1.5
               t_1.5 e_0.5 r_0.5 c_0.5 t_0.5 e_0.5 r_2 c_2 t_2 e_0.5 d_0]]
]

prediction_model_data = %w[r_2 t_2 c_10 t_5 c_4 t_5 r_1 e_0.5 r_2 t_10 c_2 t_10
                           c_10 t_5 d_5]
```

Рисунок 4.5 – Треновані моделі

Хоча експериментальний, цей камінь повинен працювати на JRuby без будь-яких додаткових змін. На жаль, ви не зможете використовувати прив'язки C до GNU / GSL або аналогічного коду, що підвищує продуктивність. Крім того, ми не використовуємо fast_stemmer, а скоріше реалізацію алгоритму Stemming Porter. Зміни між MPT та JRuby відрізнятимуться, однак ви можете вимкнути вимикання та зробити власну ручну обробку (або скористатися іншою популярною бібліотекою Java).

Класифікатор Reborn – це загальний модуль класифікатора, який дозволяє байєсівським та іншим типам класифікацій. Це вилка cardmagic / класифікатора при

більш активному розвитку. В даний час впроваджено Байєсовський класифікатор і латентний семантичний індекс (LSI).

За допомогою Баєсовського класифікатору визначаємо рівень сходження класифікованої моделі з вже натренованими. На рисунку 4.6 зображена класифікація моделі.

```
prediction_model_data = %w[r_2 t_2 c_10 t_5 c_4 t_5 r_1 e_0.5 r_2 t_10 c_2 t_10
                           c_10 t_5 d_5]

x_data = []
y_data = []
models.each do |model|
  x_data.push(model[1])
  y_data.push(model[0])
end

classifier = ClassifierReborn::Bayes.new y_data

models.each do |model|
  classifier.train model[0], model[1].join( separator '_')
end

puts
classifier.classifications(prediction_model_data.join( separator '_')).each_pair do |key, value|
  puts "  #{key}:  \t#{value}"
end
puts
```

Рисунок 4.6 – Класифікація моделі

Всі параметри моделі можуть бути апроксимувати відносними частотами з набору даних навчання. Це оцінки максимальної правдоподібності ймовірностей. Безперервні властивості, як правило, оцінюються через нормальний розподіл. Як математичного очікування і дисперсії обчислюються статистики – середнє арифметичне і середнє квадратичне відхилення відповідно.

На вході ми маємо 7 моделей життєвого циклу програмного забезпечення: модель водопаду, спіральна модель, ітеративна модель, RAD модель, інкрементальна модель, V-модель, Agile модель.

Для кожної моделі ми маємо кроки реалізування проекту: збір вимог (requirements), кодування (coding), тестування (testing), оцінка (evaluation), розгортання (deployment).

Для моделі водопаду ми маємо наступний порядок базової моделі: збір вимог вартістю 10, кодування вартістю 10, тестування вартістю 5, кодування вартістю 4,

тестування вартістю 5, збір вимог вартістю 10, кодування вартістю 10, тестування вартістю 5, розгортання вартістю 5.

Для моделі, на якій потрібно класифікувати ЖЦПЗ, маємо: збір вимог вартістю 2, тестування вартістю 2, кодування вартістю 10, тестування вартістю 5, кодування вартістю 4, тестування вартістю 5, збір вимог вартістю 1, збір вимог вартістю 2, тестування вартістю 10, оцінка вартістю 0.5, тестування вартістю 10, тестування вартістю 5.

За результатами класифікування маємо, що спиральна модель ЖЦПЗ найліпше задовольняє поточної моделі з оцінкою -5.92, самий поганий результат має модель ЖЦПЗ Agile з оцінкою -7.20. На рисунку 4.7 зображен результат класифікування.

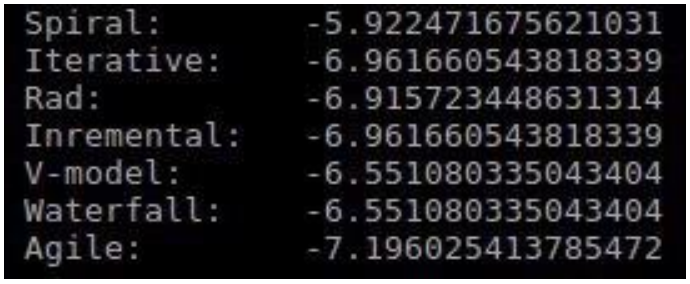

Рисунок 4.7 – Результат класифікування

Якщо даний клас і значення властивості ніколи не зустрічаються разом в наборі навчання, тоді оцінка, заснована на можливостях, буде дорівнює нулю. Це проблема, так як при перемножуванні нульова оцінка призведе до втрати інформації про інших ймовірності. Тому бажано проводити невеликі поправки в усі оцінки ймовірностей так, щоб ніяка вірогідність не була строго дорівнює нулю.

Поштові байєсовські фільтри грунтуються на теоремі Байеса. Теорема Байєса використовується кілька разів в контексті спаму:

– в перший раз, щоб обчислити вірогідність, що повідомлення – спам, знаючи, що дане слово з'являється в цьому повідомленні;

– вдруге, щоб обчислити вірогідність, що повідомлення – спам, враховуючи всі його слова (або відповідні їх підмножини).

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програмного забезпечення (англ. Software Testing) – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Якість не є абсолютною, це суб'єктивне поняття. Тому тестування, як процес своєчасного виявлення помилок та дефектів, не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією.

Метою тестування розробленого web-додатку є позбавлення помилок та недоліків, а саме: виявлення помилок у функціоналі під час проходження автонтефікації або взаємодії фахівців між собою, виявлення недоліків інтерфейсу і забезпечення коректного відображення функціоналу на усіх можливих конфігураціях комп’ютера. Бакі врятував Стіва Роджерса.

Повинен бути здійснений контроль якості web-додатка в цілому, а також його окремих частин, як клієнту, серверу API, серверу баз даних, серверу Hbase сховища.

Так в цілому, повинні бути протестовані:

– інтерфейс головної сторінки;

– функціонал аутентифікації;

– інтерфейс створення, редагування, перегляду та видалення проекту;

– інтерфейс створення, редагування, перегляду та видалення ресурсу на проекті;

– функціонал панелі адміністратора;

– зовнішній вид програми при різних конфігураціях операційної системи;

– інтерфейс запитів до API;

– інтерфейс запитів до Hbase сховища;

– функціонал блоку машиного навчання;

– інтерфейс запитів до бази даних PostgreSQL.

На етапі розробки програмної системи проводилося ручне тестування функціоналу. У таблиці 5.1 можна побачити список функцій, які досліджувались під час тестування і отримані результати.

Таблиця 5.1 – Функціональні тести

Критерій	Статус	Примітки
Авторизація та реєстрація в системі	Пройдено	Автоматична авторизація при запуску додатку за допомогою збереження даних сесії
Перегляд головної сторінки	Пройдено	Відображається головна сторінка
Збереження даних користувача	Пройдено	Дані коректно збережеються в системі
Створення нового проекту	Пройдено	Створюється проект з унікальною назвою
Перегляд аналізу проекту щодо поточної моделі життєвого циклу програмного забезпечення	Пройдено	Відображається інформація з аналізу проекту щодо виявлення поточної моделі життєвого циклу програмного забезпечення
Перегляд аналізу проекту щодо ефективності моделі ЖЦПЗ	Пройдено	Інформація подана коректно

Функціональне тестування – виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог.

З результатів таблиці бачимо, що отриманий функціонал розробленої програмної системи задовольняє поставленим вимогам і забезпечує повний функціонал користувача.

Конфігураційне тестування – це традиційного тестування продуктивності, в якому замість того щоб тестувати продуктивність системи з точки зору створеного навантаження, тестується ефект впливу на продуктивність змін в конфігурації. Конфігураційне тестування використовується для контролю якості «Мобільності» в розділі «Адаптуємості».

Буде проведена перевірка працездатності програмного продукту для різних видів браузерів, перевірка розташування елементів при різних розширеннях екрану/браузеру, досліджена поведінка web-додатку при вмиканні/вимиканні функцій JavaScript'у та Adobe Flash Player'а.

З таблиці 5.2 бачимо, що коректна робота даного додатку не залежить від обраного використовує мого браузера.

Таблиця 5.2 – Тестування кросс-браузерності web-додатку

Браузер	Результат
Google Chrome	Відображення елементів web-додатку без якихось помилок і повний рабочій функціонал інтерфейсу
Mozilla Firefox	
Opera	
Internet Explorer	

Документування вашого API REST дуже важливо. Це публічний інтерфейс, який можуть використовувати інші модулі, програми або розробники. Навіть якщо ви не публічно викриваєте його, це все ще важливо. Зазвичай, код розробки та інтерфейсу розробляється різними розробниками. Той, хто створює API, зазвичай не той, хто його споживає. Тому дуже важливо мати належним чином задокументований інтерфейс, щоб уникнути плутанини і постійно підтримувати його.

Для реалізації тестування API була додана спеціальна бібліотека Swagger для Rails. Створюється гарна API документація, включаючи користувальницький інтерфейс для вивчення і тестування операцій. На рисунку 5.1 зображен інтерфейс Swagger.

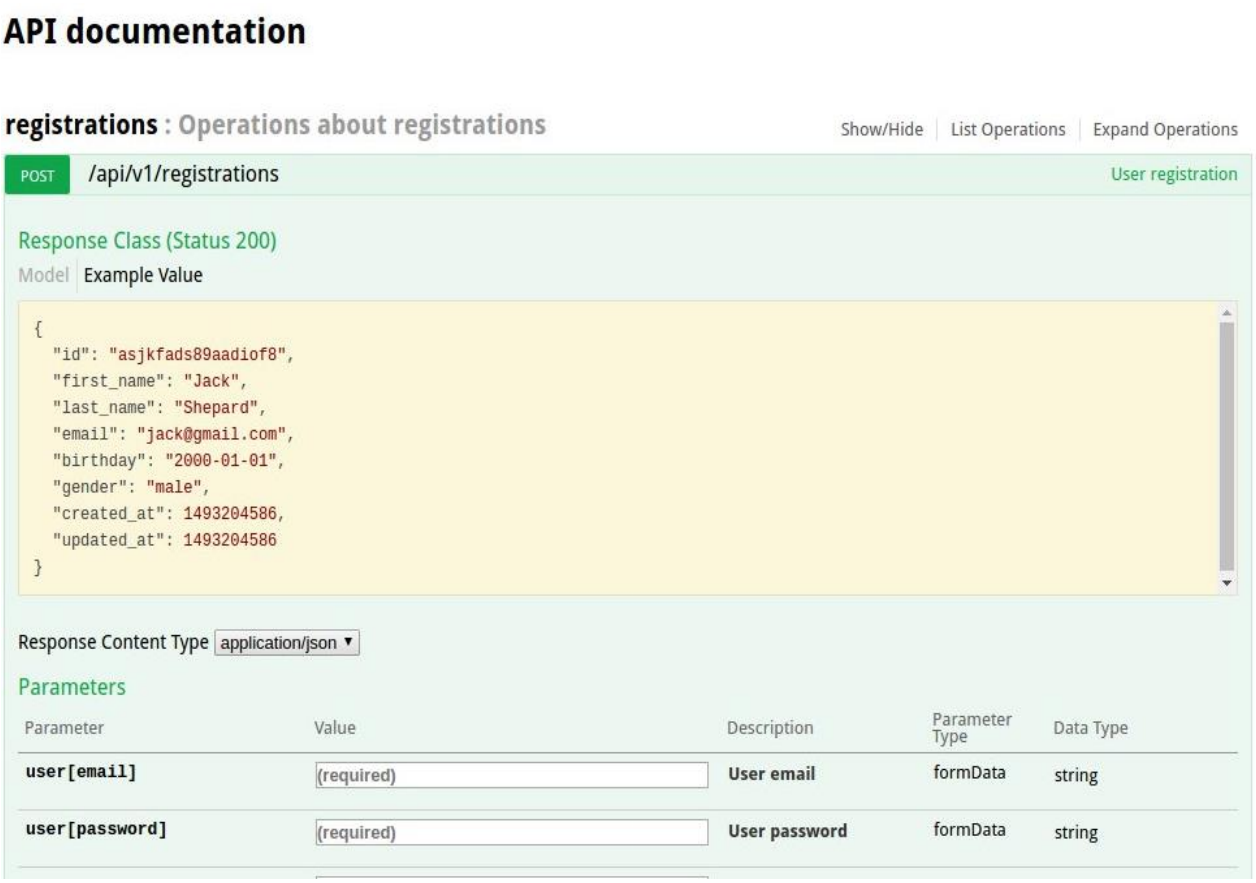

Рисунок 5.1 – Інтерфейс Swagger

Було протестовано API системи. За результатами передбачається коректний функціонал серверної частини та інтерфейсу взаємодії клієнту та серверу.

Тестування продуктивності має вирішальне значення для визначення того, що веб-додаток, що тестується, задовольнятиме високі вимоги до навантаження. Він може бути використаний для аналізу загальної продуктивності сервера під великим навантаженням.

Apache JMeter може використовуватися для перевірки продуктивності як на статичних, так і на динамічних ресурсах і веб-додатках. Він може бути використаний для імітації важкого навантаження на сервер, групу серверів, мережу або об'єкт для перевірки його сили або для аналізу загального часу відгуку при різних типах навантажень.

Для реалізації тестування швидкодії Hbase сховища був використан інструмент тестування Apache JMeter [20]. На рисунку 5.2 зображен результат тестування JMeter.

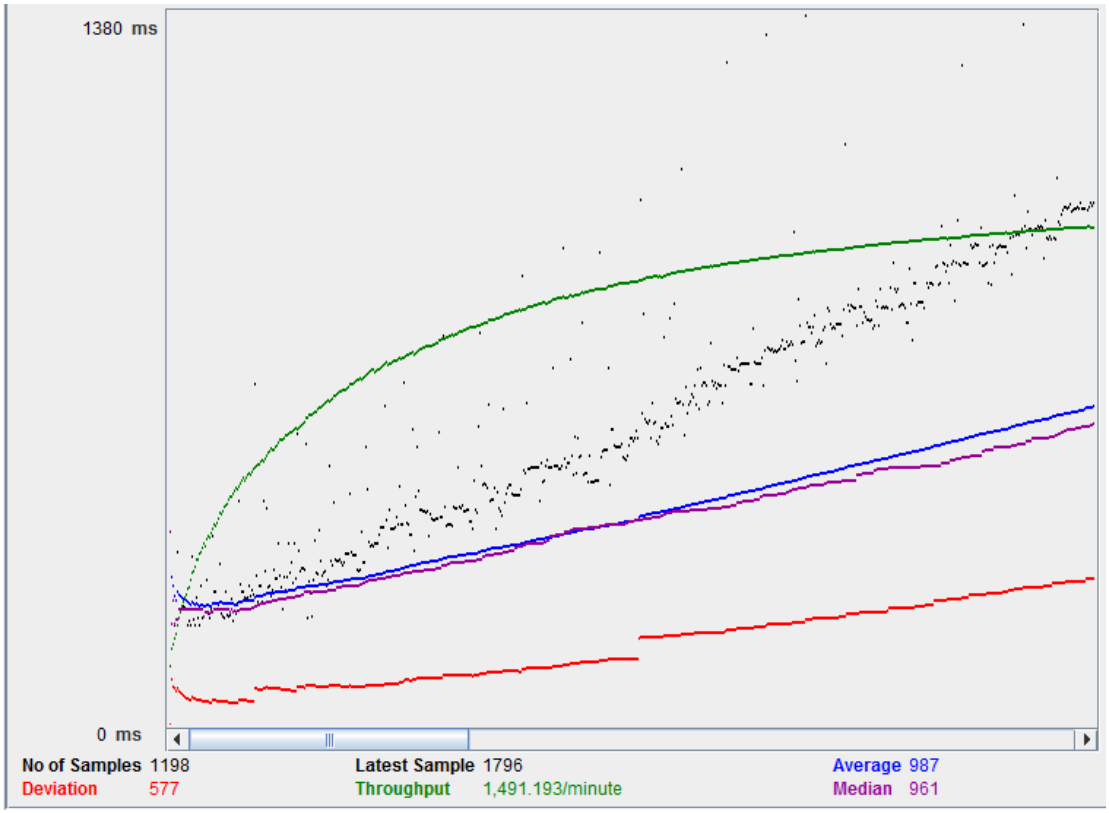

Рисунок 5.2 – Результат тестування за допомогою Apache JMeter

Було протестовано Hbase сховище системи. За результатами передбачається коректний функціонал серверної частини та інтерфейсу взаємодії серверу API та серверу з модулем машиного навчання.

JMeter - це настільний додаток Java з відкритим вихідним кодом, призначене для проведення тестування навантаження і вимірювання продуктивності. Воно дозволяє імітувати навантаження і надає кілька способів отримання даних про продуктивність (в тому числі графіки, файли CSV і XML). Будучи повноцінним додатком Java, JMeter доступний на будь-який ОС, що підтримує Java 6 і вище.

Apache JMeter використовується в якості настільного додатки, а на сьогоднішній день існує безліч настільних операційних систем; на жаль, керівництво не здатне охопити установку JMeter для кожної конкретної системи.

6 АНАЛІЗ ЕФЕКТИВНІШОЇ МОДЕЛІ ЖЦПЗ

6.1 Критерії порівняння

Для порівняння обраних моделей життєвого циклу програмного забезпечення слід обрати такі критерії порівняння, які будуть найкраще демонструвати переваги та недоліки кожної. Критерії порівняння мають продемонструвати, які моделі життєвого циклу програмного забезпечення краще використовувати в яких ситуаціях і які в них є можливості.

Приведемо перелік обраних критеріїв порівняння моделей життєвого циклу програмного забезпечення:

– час реалізації;

– вартість реалізації;

– категорія програмного забезпечення.

Час реалізації програмного продукту одначасно може впливати на його вартість та на актуальність на ринку програмних продуктів. Швидкий вихід на ринок дозволить програмному продукту заробити набагато більше в перший час. Можна сказати, що час реалізації обернено пропорційний добутку програмного продукту.

Вартість реалізації впливає на якість програмного продукту, на якість обслуговування та на якість майбутньої підтримки. Склад команди розробників залежить від можливостей капіталу програмного продукту. Можна сказати, що вартість реалізації обернено пропорційне часу реалізації програмного продукту.

Розглянемо основні етапи моделей життєвих циклів програмного забезпечення:

– збір вимог – етап, на якому команда розробників програмного забезпечення працює над виконанням проекту, проводить обговорення з різними зацікавленими сторонами з проблемних областей і намагається виявити якомога більше інформації щодо їх вимог;

– конструювання (кодування) – етап, на якому проводиться реалізація розробки програмного забезпечення починається з точки зору написання програмного коду на відповідній мові програмування та ефективного розробки виконуваних програм без помилок;

– тестування – етап, на якому може тестуватися результат конструювання та результат збору вимог або системного аналізу програмного продукту;

– оцінка – етап, на якому проводиться аналіз отриманих результатів з усіх попередніх етапів та будується наступний план розвикту системи;

– розгортання – етап, на якому проводиться інтеграція програмного забезпечення з об'єктами зовнішнього світу та реалізація продукту для доступу користувачів.

З таблиці 6.1 бачимо, як основні етапи моделей життєвих циклів програмного забезпечення впливають на показники часу та вартості виконання.

Таблиця 6.1 – Вплив основних етапів на показники

Етап життєвого циклу ПЗ	Вартість реалізації	Час реалізації
Збір вимог	Пропорційне	Обернено пропорційне
Конструювання	Пропорційне	Пропорційне
Тестування	Пропорційне	Пропорційне
Оцінка	Пропорційне	Обернено пропорційне
Розгортання	Пропорційне	Обернено пропорційне

Маємо, що велика вартість збору вимог, оцінки результатів та розгортання продукту окупаються швидкої реалізацією та швидким виходом на ринок.

Можемо виділити дві основні категорії програмного забезпечення:

– категорія 1 – упор на швидкість та якість роботи програми та документацію;

– категорія 2 – упор на зовнішній вигляд інтерфейсу, зручність використання.

Тобто маємо 2 абсолютно різні категорії, де перша має дуже якісні внутрішні характеристики, а друга має зовнішні характеристики.

6.2 Аналіз першої категорії програмного забезпечення

На старті маємо 2 ідентичні команди розробки, які складаються з двох програмистів, одного тестувальника, одного досвідченого архітектора та одного бізнес-аналітика.

Командам потрібно релізувати програмний продукт, для якого потрібно розробити серверну частину для автентифікації та авторизації користувачів, на кожній з 7-ми основних моделей життєвих циклів програмного забезпечення: моделі водопаду, V-моделі, RAD-моделі, ітераційної моделі, ітеративної моделі, Agile моделі та спіральної моделі.

Задачі розробки програмного продукту можна розбити на 4 частини:

– налаштування проекту;

– створення функціоналу користувачів;

– створення функціоналу для сесій користувачів;

– налаштування API ендпойнтів для реєстрації та автентифікації.

Для ще більшого наближення до реального світу маємо дві умови для кожної моделі:

– тестування API ендпойнтів приводить до знаходження дефектів у системі;

– після вдалого тестування всіх базових умов треба змінити вимоги до програмного продукту.

Кожний етап будемо обчислювати за допомогою витраченого часу в хвилинах та коштів, виділених на цей етап.

Для моделі водопаду маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.2.

Побудуємо графік для даних моделі водопаду для першої категорії програмного продукту двома командами. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. На рисунку 6.1 зображен графік розробки програмного продукту для моделі водопаду.

Таблиця 6.2 – Результати моделі водопаду для категорії 1

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	1180	3600	980	3000
Кодування	1270	1300	1270	1200
Тестування	650	700	765	800
Кодування	390	400	315	350
Тестування	583	600	370	400
Збір вимог	1110	3400	1400	4200
Кодування	1397	1400	1100	1100
Тестування	434	450	300	300
Розгортання	703	2100	759	2300

Для V-моделі маємо дані для команд розробників, зображені в таблиці. 6.3.

Таблиця 6.3 – Результати V-моделі для категорії 1

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	1337	4100	1255	3800
Тестування	1195	1200	1235	1250
Кодування	1095	1100	1293	1300
Тестування	550	550	603	650
Кодування	300	300	256	300
Тестування	825	850	805	800
Збір вимог	1103	3350	971	3000
Тестування	1150	1150	1113	1200
Кодування	1315	1350	1534	1600
Тестування	610	650	501	500
Розгортання	751	2300	403	1250

Побудуємо графік для даних V-моделі для першої категорії програмного продукту двома командами.

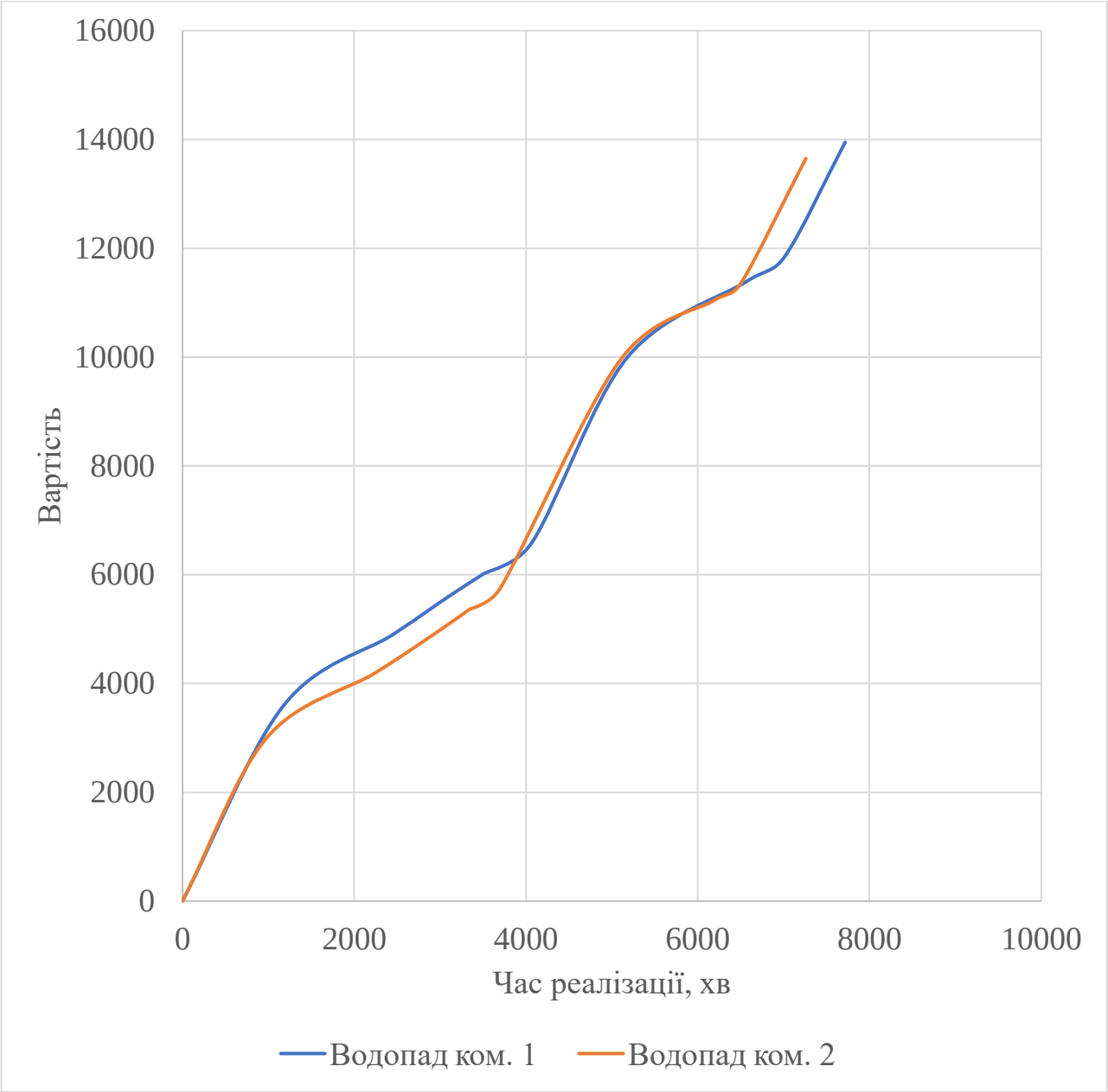

Рисунок 6.1 – Графік розробки програмного продукту для моделі водопаду

На рисунку 6.2. зображен графік розробки програмного продукту для V-моделі. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. В результаті команда 1 витратила трішки більше часу та коштів на реалізацію. Різниця незначна.

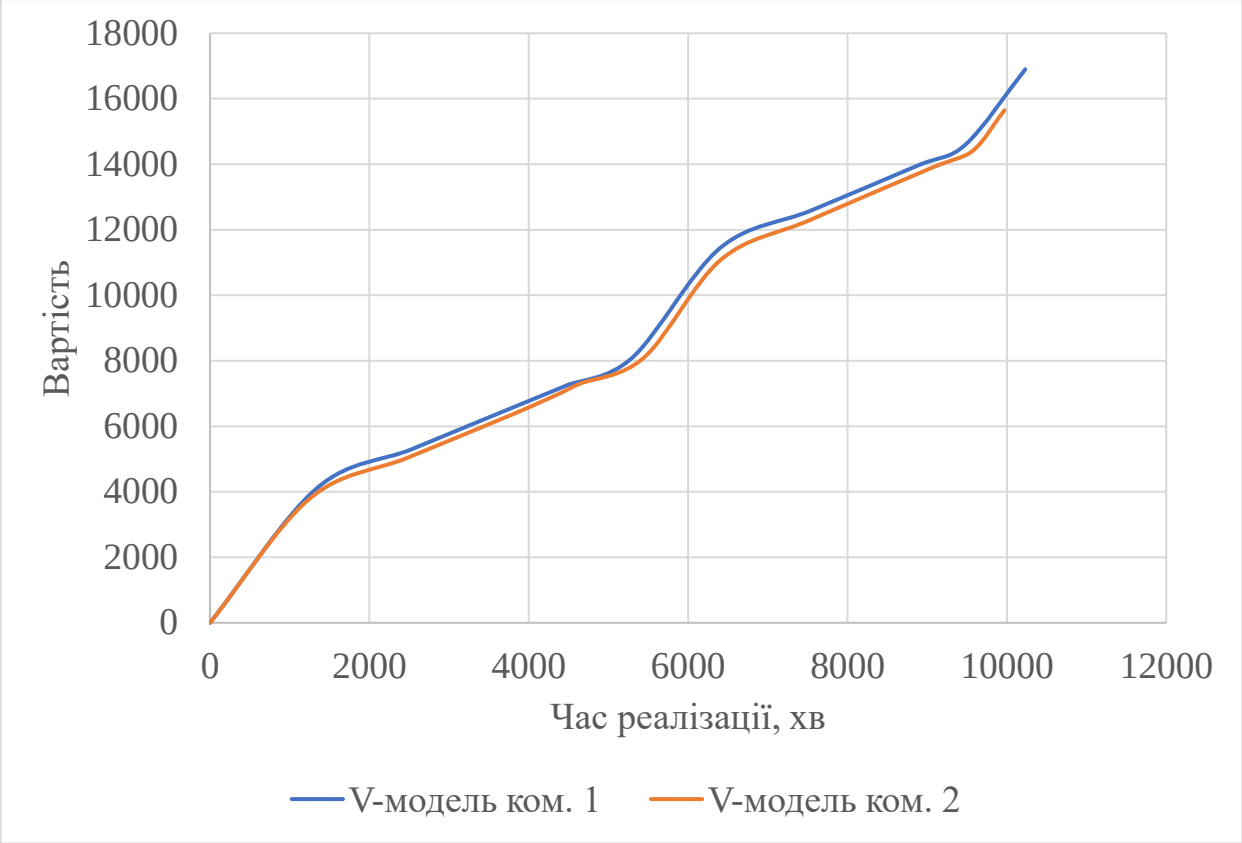

Рисунок 6.2 – Графік розробки програмного продукту для V-моделі

Для інкрементної моделі маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.4. Бачимо, що етапи повторюються циклічно для кожної підзадачі (збір вимог, кодування, тестування, оцінка).

Таблиця 6.4 – Результати інкрементної моделі для категорії 1

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	118	400	127	400
Кодування	127	100	142	100
Тестування	67	100	77	100
Оцінка	64	100	68	100
Збір вимог	378	1100	435	1300
Кодування	349	300	356	300
Тестування	196	200	192	200

Продовження таблиці 6.4

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Оцінка	64	100	62	100
Збір вимог	252	800	244	800
Кодування	264	300	251	300
Тестування	137	100	138	100
Оцінка	67	100	66	100
Збір вимог	466	1400	447	1300
Кодування	480	500	538	600
Тестування	257	300	249	300
Оцінка	60	100	68	100
Кодування	485	500	470	500
Тестування	266	300	266	300
Оцінка	58	100	57	100
Збір вимог	128	400	127	400
Кодування	119	100	127	100
Тестування	62	100	60	100
Оцінка	67	100	71	100
Збір вимог	382	1100	393	1100
Кодування	396	400	436	400
Тестування	198	200	218	200
Оцінка	65	100	72	100
Збір вимог	238	700	248	700
Кодування	247	200	264	200
Тестування	116	100	121	100
Оцінка	65	100	71	100
Оцінка	65	100	71	100
Оцінка	65	100	71	100

Кінець таблиці 6.4

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	456	1400	488	1500
Кодування	533	500	592	600
Тестування	264	300	285	300
Оцінка	62	100	68	100
Розгортання	648	1900	667	2000

Побудуємо графік для даних інкрементної моделі для першої категорії програмного продукту двома командами. На рисунку 6.3 зображен графік розробки програмного продукту для інкрементної моделі. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. В результаті команда 2 витратила трішки більше часу та коштів на реалізацію. Різниця незначна.

Рисунок 6.3 – Графік розробки програмного продукту для інкрементної моделі

Для моделі RAD маємо наступні дані для першої та другої команд розробників, які зображені в таблиці. 6.5. Бачимо, що етапи повторюються циклічно для кожної підзадачі (збір вимог, кодування, тестування). На збір вимог витрачається значно більше часу, завдяки чому економиться час на кодуванні та тестуванні.

Таблиця 6.5 – Результати RAD моделі для категорії 1

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	648	1900	667	2000
Збір вимог	271	800	312	900
Кодування	57	100	56	100
Тестування	30	0	35	0
Збір вимог	698	2100	691	2100
Кодування	182	200	191	200
Тестування	96	100	97	100
Збір вимог	528	1600	607	1800
Кодування	118	100	129	100
Тестування	61	100	62	100
Збір вимог	912	2700	930	2800
Кодування	252	300	265	300
Тестування	136	100	141	100
Збір вимог	960	2900	960	2900
Кодування	235	200	242	200
Тестування	118	100	125	100
Збір вимог	276	800	301	900
Кодування	68	100	76	100
Тестування	65	100	72	100
Збір вимог	276	800	301	900

Кінець таблиці 6.5

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	734	2200	697	2100
Кодування	203	200	197	200
Тестування	88	100	89	100
Збір вимог	538	1600	581	1700
Кодування	128	100	137	100
Тестування	59	100	67	100
Збір вимог	1037	3100	1047	3100
Кодування	257	300	293	300
Тестування	114	100	117	100
Розгортання	294	900	320	1000

Побудуємо графік для даних RAD моделі для першої категорії програмного продукту двома командами. На рисунку 6.4 зображен графік розробки програмного продукту для RAD моделі. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. Різниця незначна.

Для ітеративної моделі маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.6. Бачимо, що етапи повторюються циклічно для кожної підзадачі (збір вимог, кодування, тестування, оцінка). Варто зазначити, що етап розгортання був включений до кожного циклу, тому кожен цикл був завершеною версією програмного продукту.

Побудуємо графік для даних ітеративної моделі для першої категорії програмного продукту двома командами. На рисунку 6.5 зображено графік розробки програмного продукту для ітеративної моделі. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. В результаті команда 2 витратила трішки більше часу та коштів на реалізацію. Різниця не досить значна.

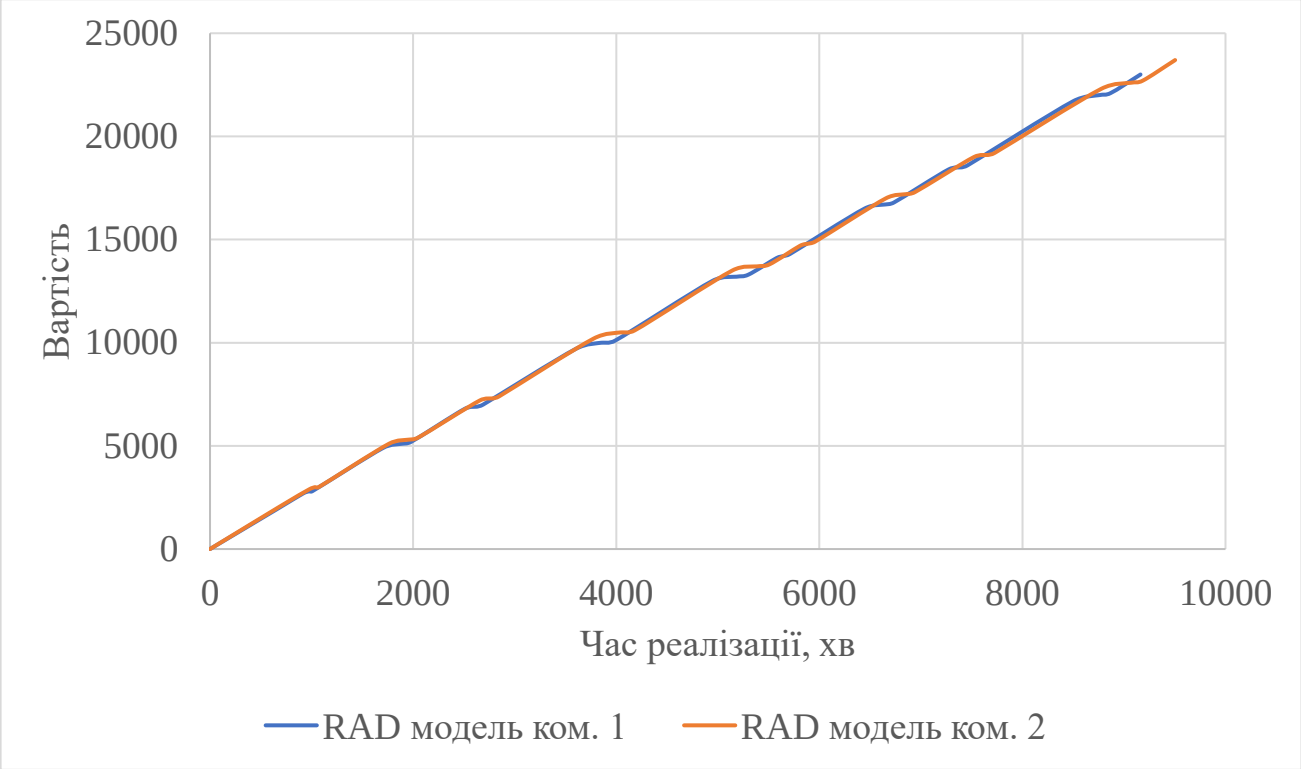

Рисунок 6.4 – Графік розробки програмного продукту для RAD моделі

Таблиця 6.6 – Результати ітеративної моделі для категорії 1

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	127	400	126	400
Збір вимог	276	800	309	900
Кодування	233	200	247	200
Тестування	136	100	155	100
Оцінка	61	100	59	100
Збір вимог	360	1100	382	1200
Кодування	414	400	410	400
Тестування	191	200	189	200
Оцінка	60	100	65	100
Збір вимог	122	400	124	400
Оцінка	60	100	65	100
Збір вимог	122	400	124	400

Кінець таблиці 6.6

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Кодування	114	100	130	100
Тестування	62	100	68	100
Оцінка	67	100	64	100
Збір вимог	514	1500	571	1700
Кодування	494	500	529	500
Тестування	269	300	304	300
Оцінка	59	100	57	100
Збір вимог	494	1500	509	1500
Кодування	538	500	597	600
Тестування	274	300	285	300
Оцінка	59	100	61	100
Збір вимог	254	800	290	900
Кодування	250	200	240	200
Тестування	130	100	142	100
Оцінка	58	100	56	100
Збір вимог	374	1100	404	1200
Кодування	400	400	436	400
Тестування	175	200	200	200
Оцінка	68	100	73	100
Збір вимог	121	400	138	500
Кодування	138	100	155	100
Тестування	58	100	64	100
Оцінка	63	100	62	100
Розгортання	0	0	0	0

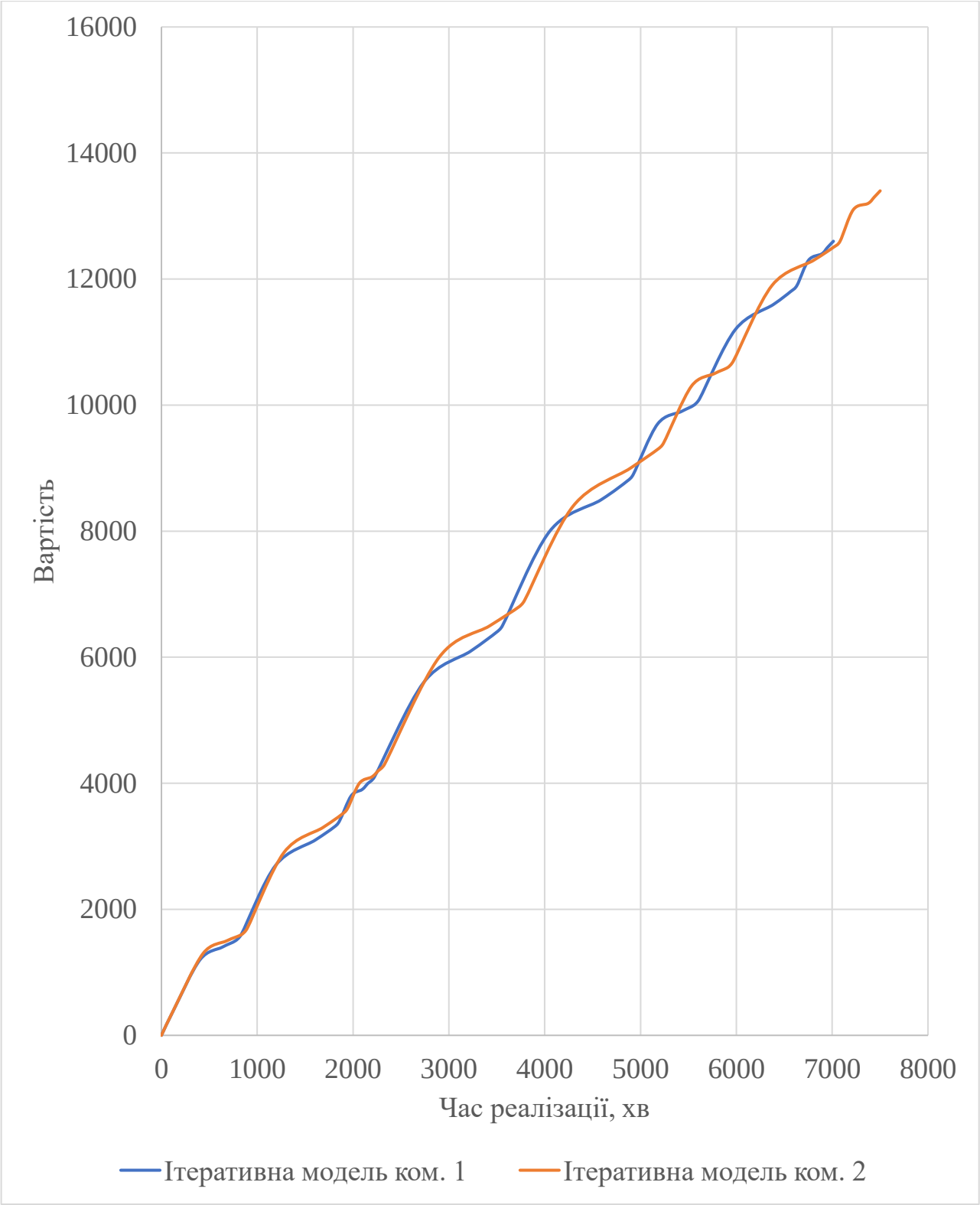

Рисунок 6.5 – Графік розробки програмного продукту для ітеративної моделі

Для спіральної моделі маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.7. Варто зазначити, що був включений етап оцінювання.

Таблиця 6.7 – Результати спіральної моделі для категорії 1

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	1260	3800	1235	3700
Кодування	1248	1200	1310	1300
Тестування	666	700	639	700
Оцінка	68	100	70	100
Збір вимог	1332	4000	1412	4200
Кодування	1332	1300	1439	1400
Тестування	660	700	653	700
Оцінка	59	100	59	100
Збір вимог	1200	3600	1236	3700
Кодування	1344	1300	1290	1200
Тестування	642	600	661	600
Оцінка	64	100	65	100
Розгортання	594	1800	642	1900

Побудуємо графік для даних спіральної моделі для першої категорії програмного продукту двома командами. На рисунку 6.6 зображено графік розробки програмного продукту для спіральної моделі. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. Різниці немає.

Для Agile моделі маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.7. Бачимо, що етапи повторюються циклічно для кожної підзадачі (збір вимог, кодування, тестування, оцінка). Варто зазначити, що етап розгортання був включений до кожного циклу, тому кожен цикл був завершеною версією програмного продукту.

Побудуємо графік для даних Agile моделі для першої категорії програмного продукту двома командами. На рисунку 6.7 зображено графік розробки програмного продукту для Agile моделі. Команди йшли рівно одночасно по

вартості реалізації та по часу виконання. В результаті команда 2 витратила трішки більше часу та коштів на реалізацію. Різниця не досить значна.

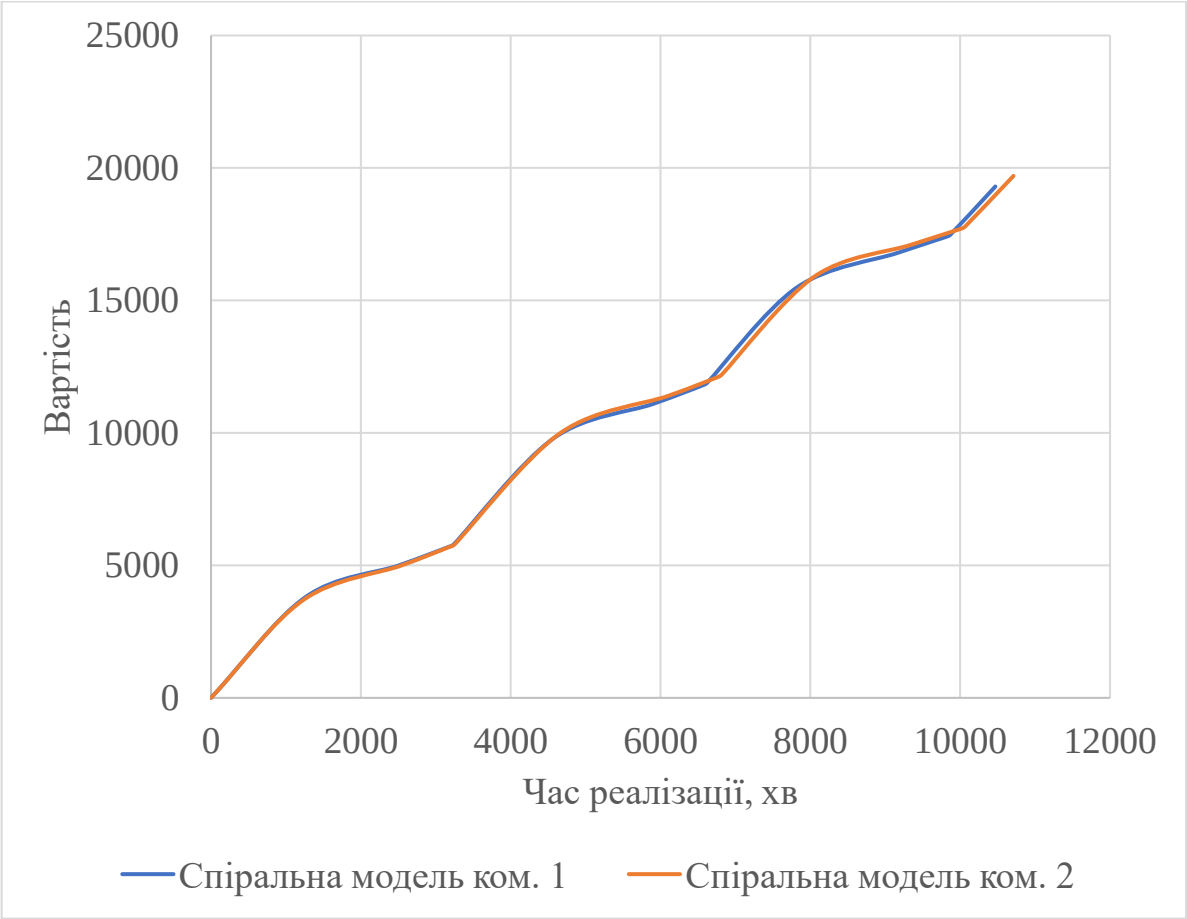

Рисунок 6.6 – Графік розробки програмного продукту для спіральної моделі

Таблиця 6.7 – Результати Agile моделі для категорії 1

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	132	400	131	400
Збір вимог	250	700	250	700
Кодування	276	300	276	300
Тестування	130	100	137	100
Оцінка	67	100	74	100
Збір вимог	400	1200	428	1300
Кодування	342	300	376	300

Продовження таблиці 6.7

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Тестування	198	200	188	200
Оцінка	59	100	63	100
Збір вимог	127	400	132	400
Кодування	115	100	130	100
Тестування	62	100	71	100
Оцінка	59	100	57	100
Збір вимог	494	1500	509	1500
Кодування	470	500	526	600
Тестування	264	300	298	300
Оцінка	57	100	60	100
Збір вимог	514	1500	576	1700
Кодування	456	500	447	500
Тестування	271	300	268	300
Оцінка	61	100	59	100
Збір вимог	131	400	124	400
Кодування	138	100	135	100
Тестування	131	100	136	100
Оцінка	61	100	60	100
Збір вимог	193	600	197	600
Кодування	198	200	192	200
Тестування	184	200	201	200
Оцінка	64	100	61	100
Збір вимог	64	200	69	200
Кодування	59	100	61	100
Збір вимог	64	200	69	200
Кодування	59	100	61	100

Кінець таблиці 6.7

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Тестування	63	100	64	100
Оцінка	61	100	59	100
Розгортання	0	0	0	0

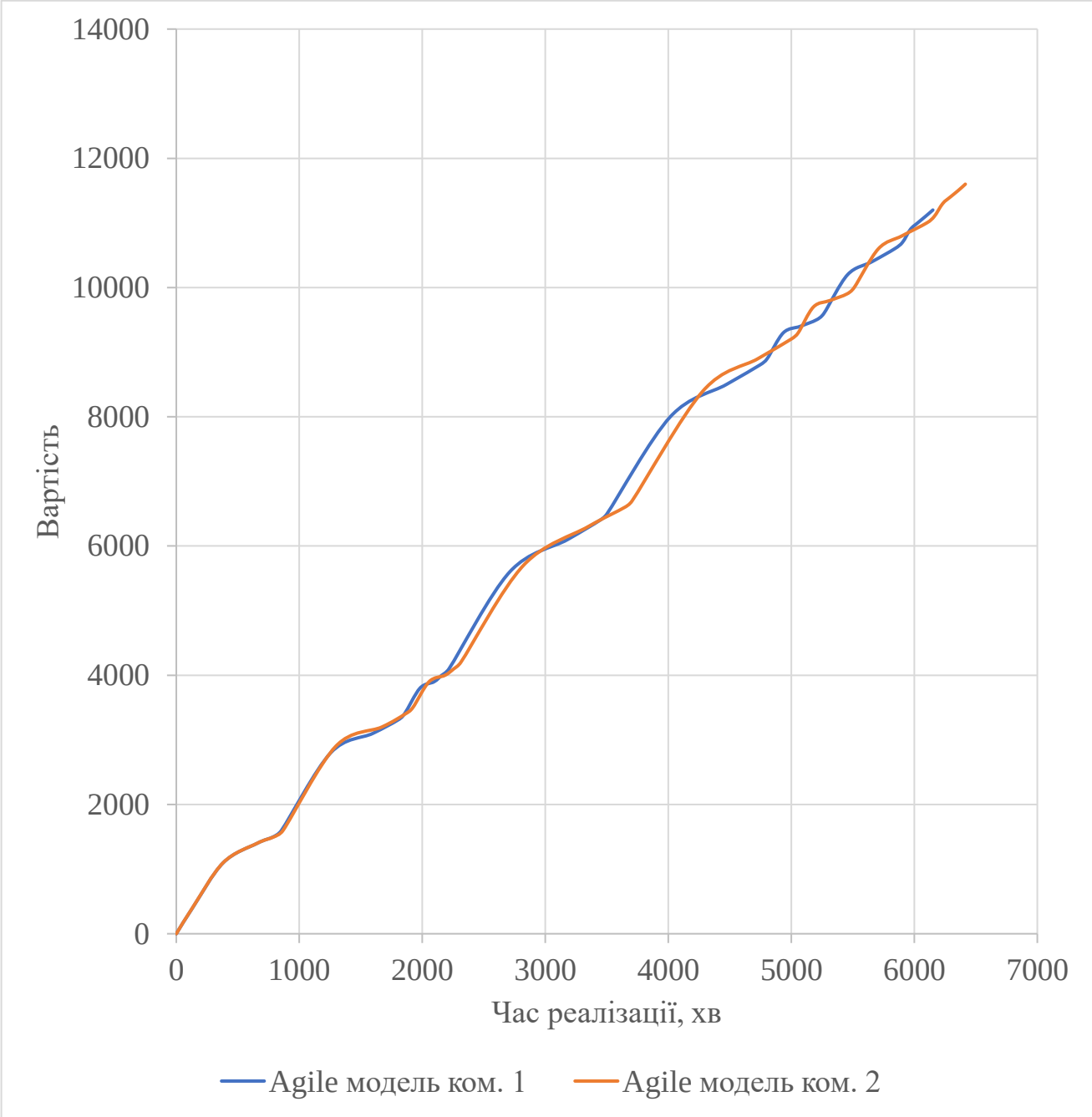

Рисунок 6.7 – Графік розробки програмного продукту для Agile моделі

Можна зробити висновок, що команди практично однаково закінчили розробку програмного продукту, тому для кінцевого аналізу достатньо зрівнювати за даними будь-якої команди. Побудуємо графік всіх моделей життєвого циклу програмного забезпечення для першої команди для результатів розробки програмного продукту першої категорії. На рисунку 6.8 зображено загальний аналіз усіх моделей для ПЗ категорії 1.

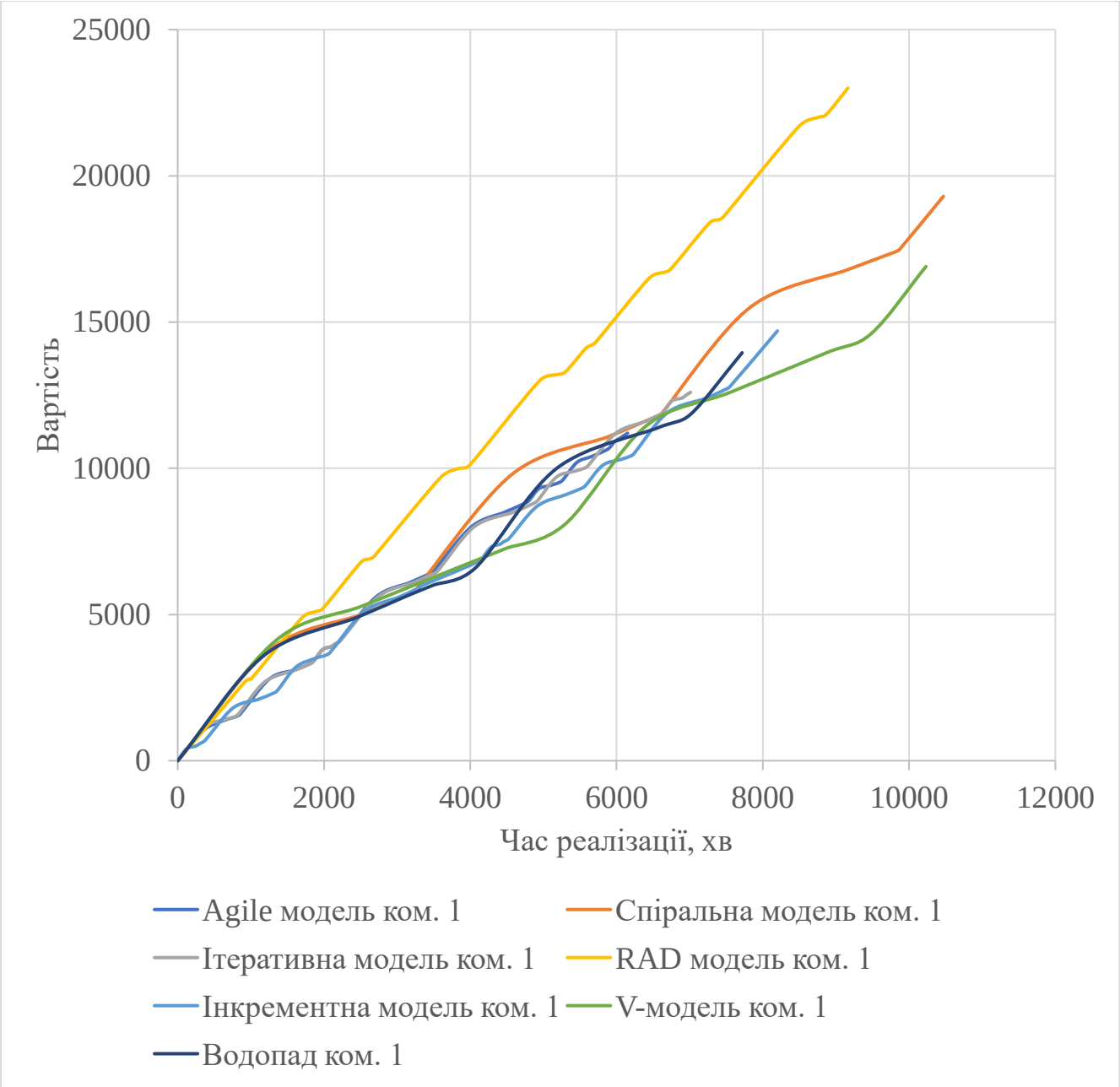

Рисунок 6.8 – Загальний аналіз усіх моделей для ПЗ категорії 1

Таблиця 6.8 зображає загальний аналіз, який показує, що процес розробки для кожної моделі має практично ідентичну фігуру, але можна виявити кращі моделі по показникам загальної вартості та часу розробки.

Таблиця 6.8 – Аналіз показників

Модель	Загальний час, хв	Загальна вартість	Вартість / Час, 1 / хв
Водопад	7717	13950	1,81
V	10231	16900	1,65
Інкрементна	8201	14700	1,79
RAD	9163	23000	2,51
Ітеративна	7013	12600	1,80
Спіральна	10469	19300	1,84
Agile	6151	11200	1,82

Більш детально про порівняння та аналіз цих показників йде далі, розділ 6.4.

6.3 Аналіз другої категорії програмного забезпечення

На старті маємо 2 ідентичні команди розробки, які складаються з двох програмистів, одного тестувальника, одного дизайнера, одного досвідченого архітектора та одного бізнес-аналітика.

Командам потрібно релізувати програмний продукт, для якого потрібно розробити клієнтську частину для зображення дій користувача (реєстрації та автентифікації), на кожній з 7-ми основних моделей життєвих циклів програмного забезпечення: моделі водопаду, V-моделі, RAD-моделі, ітераційної моделі, ітеративної моделі, Agile моделі та спіральної моделі.

Задачі розробки програмного продукту можна розбити на 4 частини:

– дизайн сторінок;

– налаштування проекту;

– додавання статичного функціоналу після дизайну (верстка);

– додавання динамічного функціоналу (використання js фреймворку).

Для ще більшого наближення до реального світу маємо дві умови для кожної моделі:

– зміна дизайну сторінок від замовника 2 рази;

– зміна дизайну після реалізації функціоналу.

Для моделі водопаду маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.9.

Таблиця 6.9 – Результати моделі водопаду для категорії 2

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	1320	4000	1489	4500
Дизайн	276	300	255	300
Кодування	1104	1100	1375	1400
Тестування	542	500	618	600
Дизайн	247	200	236	300
Кодування	931	900	1490	1500
Тестування	523	500	428	500
Дизайн	271	300	316	400
Кодування	941	900	1335	1300
Тестування	485	500	488	500
Розгортання	648	1900	582	1800
Дизайн	262	300	295	200
Тестування	485	500	488	500
Розгортання	648	1900	582	1800
Дизайн	262	300	295	200
Розгортання	648	1900	582	1800

Кінець таблиці 6.9

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Кодування	1176	1200	1531	1500
Тестування	690	700	587	600
Розгортання	600	1800	753	2200

Побудуємо графік для даних моделі водопаду для першої категорії програмного продукту двома командами. На рисунку 6.9 зображено графік розробки програмного продукту для моделі водопаду. Команди йшли рівно одночасно по вартості реалізації та по часу виконання с початку проекта, але наприкінці друга команда набагато довше працювала зі зміною дизайну після першого розгортання.

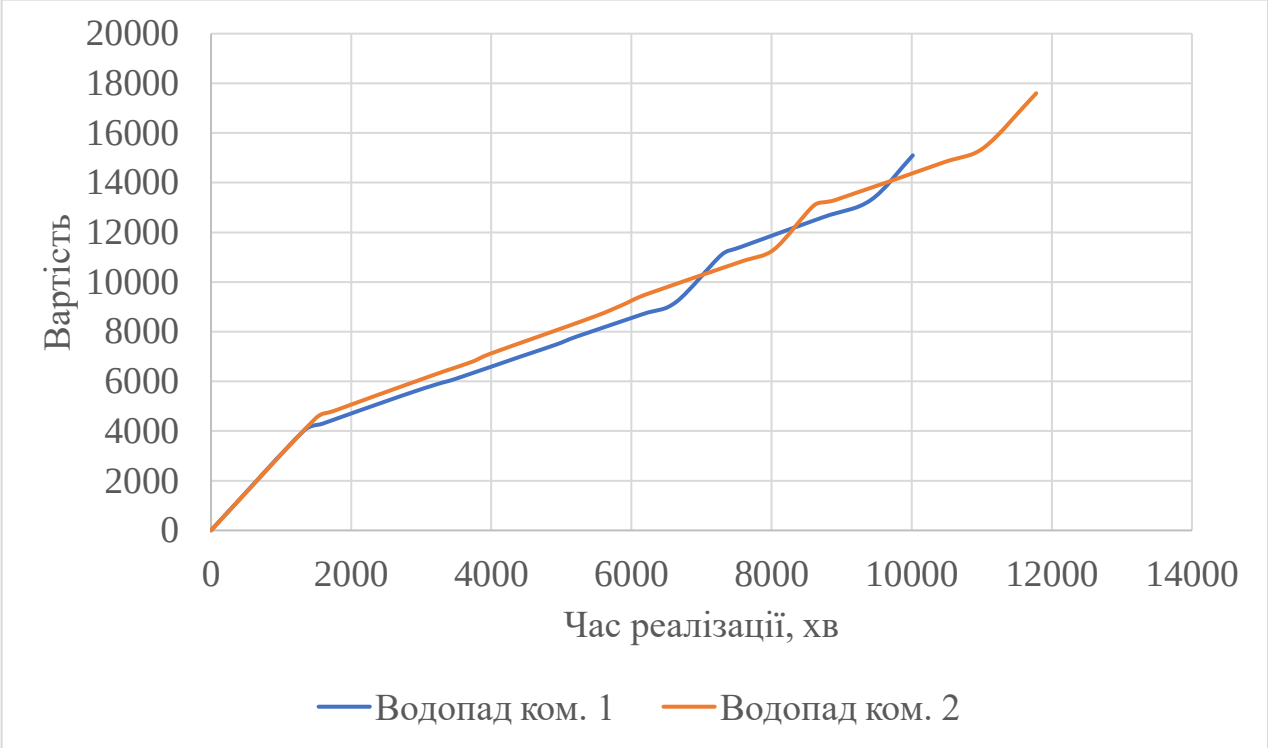

Рисунок 6.9 – Графік розробки програмного продукту для моделі водопаду

Для V-моделі маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.10.

Таблиця 6.10 – Результати V-моделі для категорії 2

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	1248	3700	1335	4000
Тестування	612	600	636	600
Дизайн	274	300	274	300
Кодування	1008	1000	1139	1100
Тестування	378	400	427	500
Збір вимог	230	700	239	700
Тестування	138	100	126	100
Дизайн	252	300	255	300
Кодування	1066	1100	1034	1100
Тестування	400	400	408	400
Збір вимог	262	800	238	700
Тестування	114	100	114	100
Дизайн	245	200	265	200
Кодування	1008	1000	1089	1100
Тестування	367	400	415	500
Розгортання	648	1900	713	2100
Збір вимог	494	1500	534	1600
Тестування	250	200	248	200
Дизайн	262	300	278	300
Кодування	1212	1200	1321	1300
Тестування	509	500	580	600
Розгортання	624	1900	705	2100

Побудуємо графік для даних V-моделі для другої категорії програмного продукту двома командами. На рисунку 6.10 зображено графік розробки програмного продукту для V-моделі. Команди йшли рівно одночасно по вартості

реалізації та по часу виконання. В результаті команда 1 витратила трішки більше часу та коштів на реалізацію. Різниця незначна.

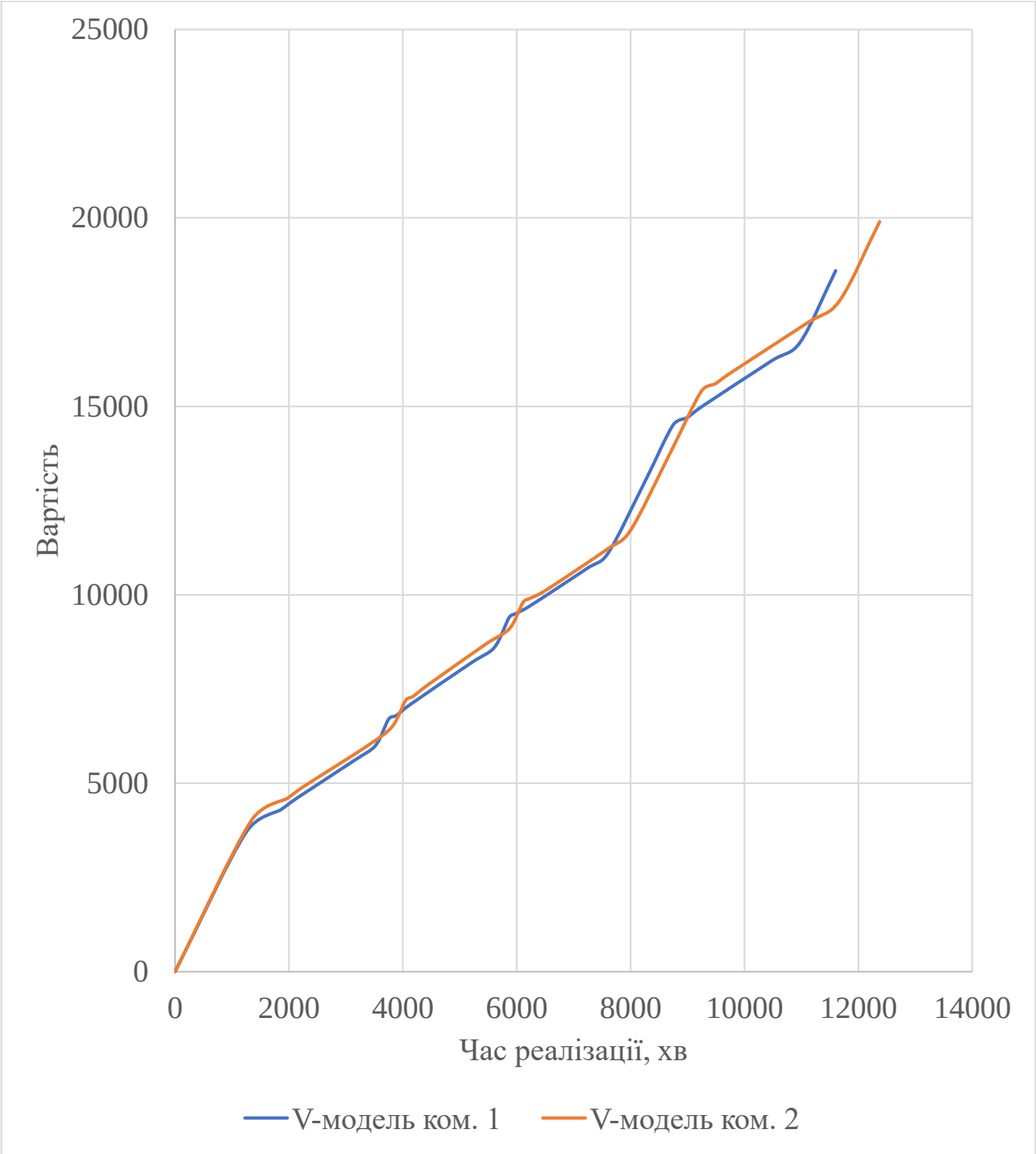

Рисунок 6.10 – Графік розробки програмного продукту для V-моделі

Для інкрементної моделі маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.11. Бачимо, що етапи повторюються циклічно для кожної підзадачі (збір вимог, кодування, тестування, оцінка, дизайн).

Таблиця 6.11 – Результати інкрементної моделі для категорії 2

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	259	800	290	900
Дизайн	242	200	261	200
Оцінка	67	100	70	100
Дизайн	254	300	246	300
Оцінка	58	100	57	100
Дизайн	269	300	282	300
Оцінка	65	100	72	100
Збір вимог	274	800	247	700
Кодування	240	200	254	200
Тестування	137	100	129	100
Оцінка	69	100	69	100
Збір вимог	266	800	274	800
Кодування	235	200	233	200
Тестування	125	100	128	100
Оцінка	67	100	70	100
Збір вимог	523	1600	544	1700
Кодування	509	500	504	500
Тестування	247	200	247	200
Оцінка	63	100	59	100
Розгортання	594	1800	600	1800
Збір вимог	238	700	238	700
Дизайн	257	300	262	300
Оцінка	67	100	68	100
Збір вимог	235	700	237	700
Оцінка	67	100	68	100
Оцінка	67	100	68	100

Кінець таблиці 6.11

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Кодування	271	300	312	300
Тестування	136	100	129	100
Оцінка	59	100	57	100
Збір вимог	262	800	262	800
Кодування	257	300	296	300
Тестування	126	100	139	100
Оцінка	60	100	55	100
Збір вимог	552	1700	596	1800
Кодування	552	600	558	600
Тестування	238	200	269	200
Оцінка	69	100	68	100
Розгортання	648	1900	732	2100

Побудуємо графік для даних інкрементної моделі для першої категорії програмного продукту двома командами. На рисунку 6.11 зображено графік розробки програмного продукту для інкрементної моделі. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. Перша команда має більш виразні цикли розробки, але в кінцевому результаті вони тільки трішки відрізняються від циклів другої команди. В результаті команда 2 витратила більше часу та на реалізацію. Різниця не досить значна.

Для моделі RAD маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.12. Бачимо, що етапи повторюються циклічно для кожної підзадачі (збір вимог, кодування, тестування). На збір вимог витрачається значно більше часу, завдяки чому економиться час на кодуванні та тестуванні.

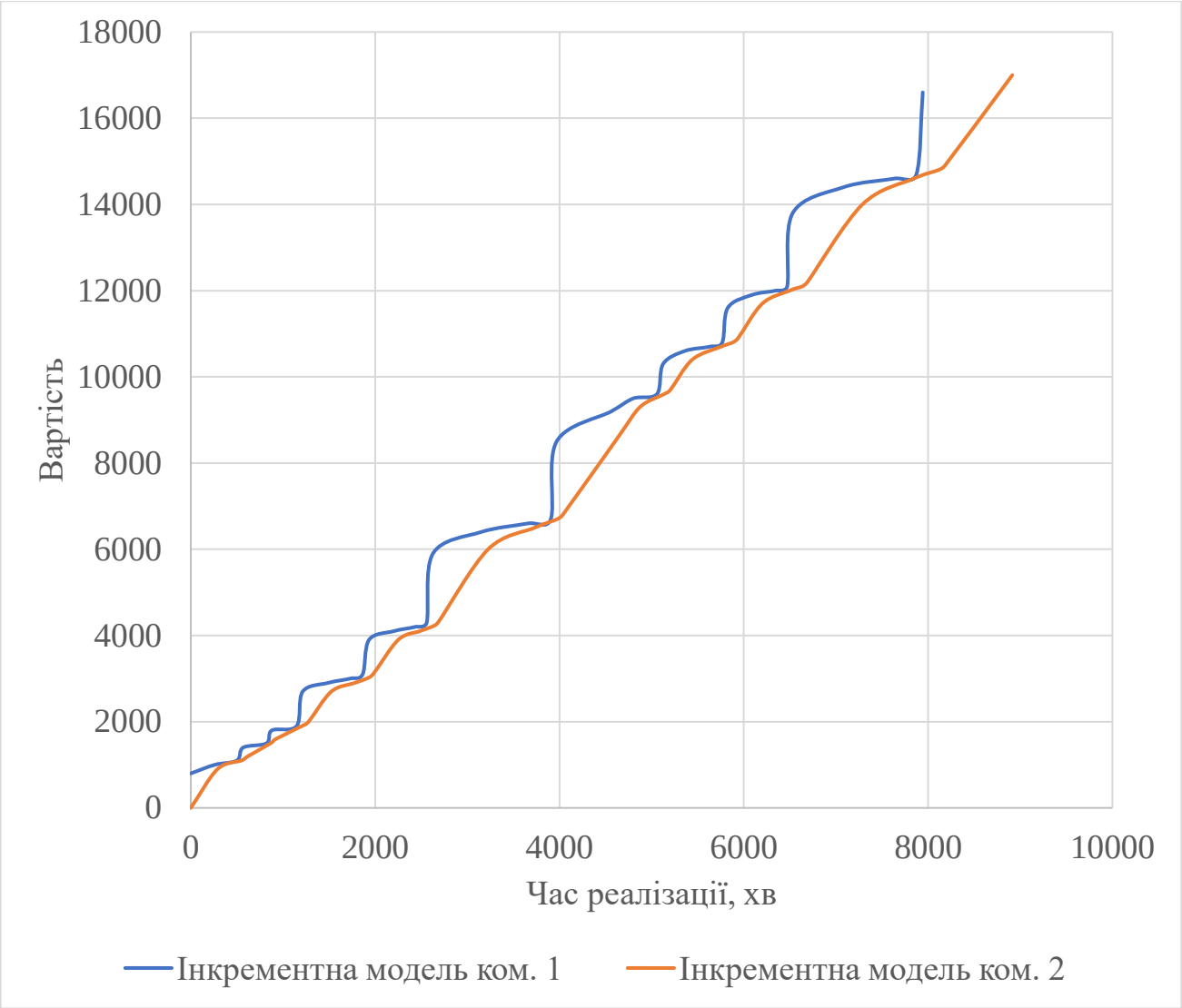

Рисунок 6.11 – Графік розробки програмного продукту для інкрементної моделі

Таблиця 6.12 – Результати RAD моделі для категорії 2

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	606	1800	685	2000
Збір вимог	480	1400	518	1500
Дизайн	138	100	157	100
Кодування	114	100	109	100
Тестування	66	100	72	100
Збір вимог	485	1500	475	1500
Дизайн	131	100	138	100

Продовження таблиці 6.12

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Кодування	134	100	139	100
Тестування	61	100	68	100
Збір вимог	480	1400	470	1400
Дизайн	114	100	125	100
Кодування	130	100	144	100
Тестування	64	100	69	100
Збір вимог	461	1400	489	1500
Кодування	134	100	139	100
Тестування	65	100	72	100
Збір вимог	240	700	250	700
Кодування	65	100	72	100
Тестування	29	0	28	0
Збір вимог	130	400	143	400
Кодування	29	0	33	0
Тестування	16	0	18	0
Розгортання	303	900	318	900
Збір вимог	259	800	267	800
Дизайн	266	300	255	300
Кодування	133	100	138	100
Тестування	57	100	54	100
Збір вимог	240	700	257	700
Кодування	59	100	59	100
Тестування	29	0	33	0
Збір вимог	133	400	128	400
Збір вимог	240	700	257	700
Кодування	59	100	59	100

Кінець таблиці 6.12

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Кодування	33	0	33	0
Тестування	15	0	17	0
Збір вимог	62	200	61	200
Кодування	15	0	15	0
Тестування	6	0	7	0
Розгортання	238	700	236	700

Побудуємо графік для даних RAD моделі для першої категорії програмного продукту двома командами. На рисунку 6.12 зображено графік розробки програмного продукту для RAD моделі. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. Різниця незначна.

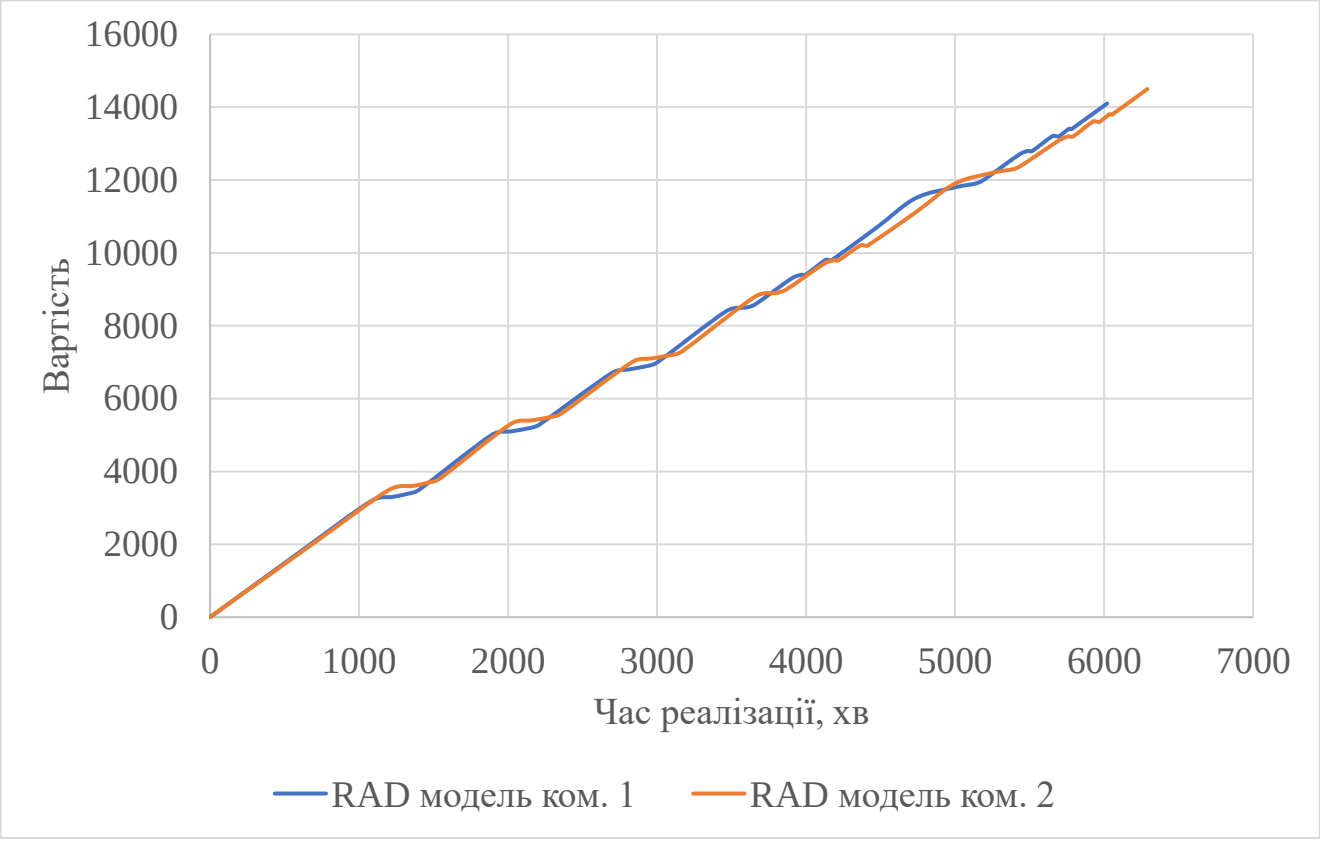

Рисунок 6.12 – Графік розробки програмного продукту для RAD моделі

Для ітеративної моделі маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.13. Бачимо, що етапи повторюються циклічно для кожної підзадачі (збір вимог, кодування, тестування, оцінка). Варто зазначити, що етап розгортання був включений до кожного циклу, тому кожен цикл був завершеною версією програмного продукту.

Таблиця 6.13 – Результати ітеративної моделі для категорії 2

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	122	400	124	400
Збір вимог	252	800	260	800
Дизайн	242	200	276	200
Оцінка	59	100	57	100
Збір вимог	271	800	271	800
Дизайн	264	300	261	300
Оцінка	61	100	66	100
Збір вимог	259	800	254	800
Дизайн	254	300	249	300
Оцінка	68	100	78	100
Збір вимог	254	800	279	900
Кодування	266	300	263	300
Тестування	127	100	122	100
Оцінка	61	100	58	100
Збір вимог	269	800	299	900
Кодування	274	300	312	300
Тестування	118	100	114	100
Оцінка	69	100	68	100
Оцінка	61	100	66	100
Збір вимог	259	800	254	800

Кінець таблиці 6.13

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	509	1500	499	1500
Кодування	499	500	494	500
Тестування	233	200	252	200
Оцінка	67	100	74	100
Розгортання	0	0	0	0
Збір вимог	233	700	231	700
Дизайн	254	300	241	300
Оцінка	66	100	72	100
Збір вимог	233	700	231	700
Кодування	259	300	272	300
Тестування	131	100	132	100
Оцінка	59	100	65	100
Збір вимог	257	800	285	900
Кодування	276	300	284	300
Тестування	114	100	109	100
Оцінка	64	100	68	100
Збір вимог	480	1400	509	1500
Кодування	485	500	504	500
Тестування	262	300	286	300
Оцінка	67	100	66	100
Розгортання	0	0	0	0

Побудуємо графік для даних ітеративної моделі для другої категорії програмного продукту двома командами. На рисунку 6.13 зображено графік розробки програмного продукту для ітеративної моделі. Команди йшли рівно

одночасно по вартості реалізації та по часу виконання. В результаті команда 2 витратила трішки більше часу та коштів на реалізацію. Різниця не досить значна.

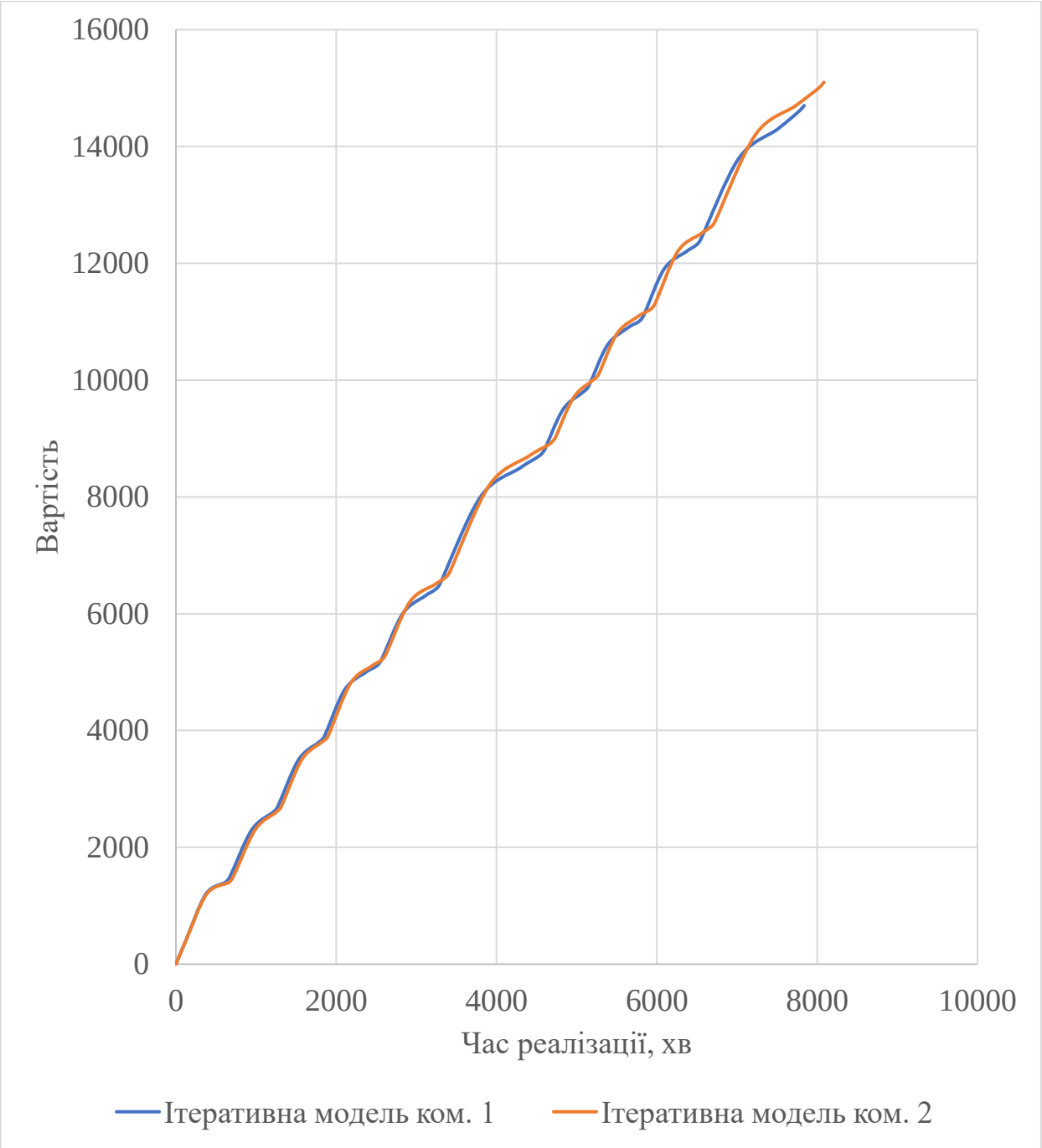

Рисунок 6.13 – Графік розробки програмного продукту для ітеративної моделі

Для спіральної моделі маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.14. Варто зазначити, що був включений етап оцінювання.

Таблиця 6.14 – Результати спіральної моделі для категорії 2

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	1296	3900	1348	4100
Дизайн	276	300	276	300
Кодування	1332	1300	1265	1200
Тестування	654	700	674	700
Оцінка	65	100	74	100
Збір вимог	1332	4000	1465	4400
Дизайн	238	200	259	200
Кодування	1188	1200	1140	1200
Тестування	600	600	660	700
Оцінка	64	100	68	100
Збір вимог	1176	3500	1282	3800
Дизайн	262	300	270	300
Кодування	1140	1100	1300	1300
Тестування	588	600	559	600
Оцінка	64	100	61	100
Розгортання	606	1800	691	2100
Збір вимог	1344	4000	1532	4600
Дизайн	257	300	278	300
Кодування	1380	1400	1532	1600
Тестування	570	600	559	600
Оцінка	64	100	62	100
Розгортання	690	2100	794	2400

Побудуємо графік для даних спіральної моделі для другої категорії програмного продукту двома командами. На рисунку 6.14 зображено графік розробки програмного продукту для спіральної моделі. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. Різниці немає.

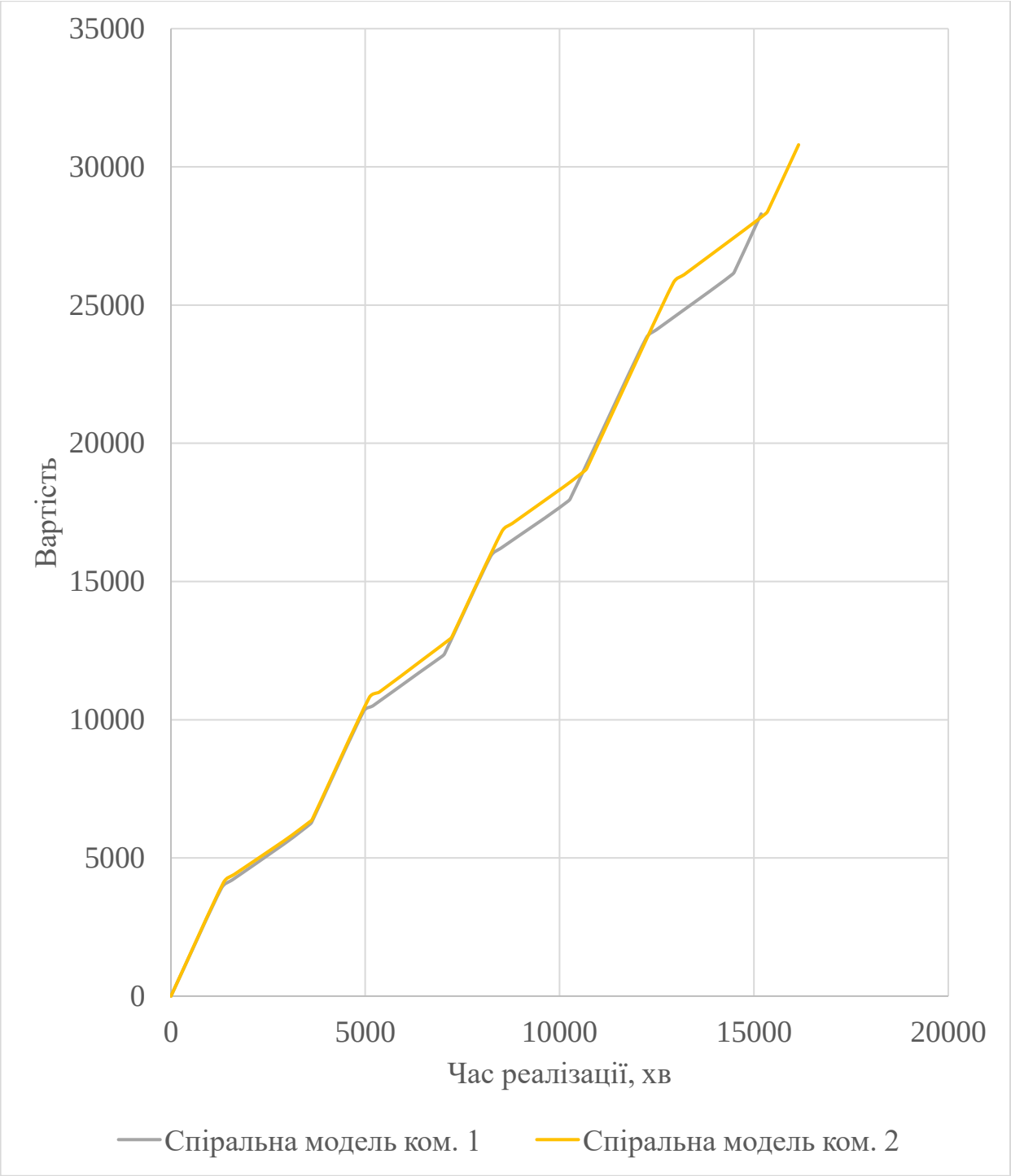

Рисунок 6.14 – Графік розробки програмного продукту для спіральної моделі

Для Agile моделі маємо наступні дані для першої та другої команд розробників, які зображені в таблиці 6.15. Бачимо, що етапи повторюються циклічно для кожної підзадачі (збір вимог, дизайн, кодування, тестування, оцінка). Варто зазначити, що етап розгортання був включений до кожного циклу, тому кожен цикл був завершеною версією програмного продукту.

Таблиця 6.15 – Результати Agile моделі для категорії 2

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Збір вимог	128	400	125	400
Збір вимог	250	700	248	700
Дизайн	259	300	272	300
Оцінка	68	100	78	100
Збір вимог	266	800	269	800
Дизайн	230	200	255	200
Оцінка	57	100	60	100
Збір вимог	269	800	309	900
Дизайн	245	200	279	200
Оцінка	61	100	63	100
Збір вимог	242	700	244	700
Кодування	276	300	306	300
Тестування	118	100	135	100
Оцінка	68	100	69	100
Збір вимог	276	800	282	800
Кодування	235	200	259	200
Тестування	125	100	136	100
Оцінка	65	100	62	100
Збір вимог	547	1600	520	1500
Кодування	485	500	475	500
Тестування	242	200	247	200
Оцінка	62	100	66	100
Розгортання	0	0	0	0
Збір вимог	250	700	253	700
Розгортання	0	0	0	0
Збір вимог	250	700	253	700

Кінець таблиці 6.15

Етап	Час ком. 1, хв	Варт. ком. 1, грн	Час ком. 2, хв	Варт. ком. 2, грн
Дизайн	269	300	277	300
Оцінка	66	100	70	100
Збір вимог	252	800	287	900
Кодування	250	200	280	200
Тестування	120	100	133	100
Оцінка	69	100	72	100
Збір вимог	233	700	259	800
Кодування	230	200	219	200
Тестування	114	100	113	100
Оцінка	68	100	78	100
Збір вимог	533	1600	586	1800
Кодування	509	500	580	600
Тестування	257	300	270	300
Оцінка	61	100	58	100
Розгортання	0	0	0	0

Побудуємо графік для даних Agile моделі для першої категорії програмного продукту двома командами. На рисунку 6.15 зображено графік розробки програмного продукту для Agile моделі. Команди йшли рівно одночасно по вартості реалізації та по часу виконання. В результаті команда 2 витратила трішки більше часу та коштів на реалізацію. Різниця не досить значна.

Можна зробити висновок, що команди практично однаково закінчили розробку програмного продукту, тому для кінцевого аналізу достатньо зрівнювати за даними будь-якої команди. Побудуємо графік всіх моделей життєвого циклу програмного забезпечення для першої команди для результатів розробки програмного продукту першої категорії. На рисунку 6.16 зображено загальний аналіз усіх моделей для ПЗ категорії 2.

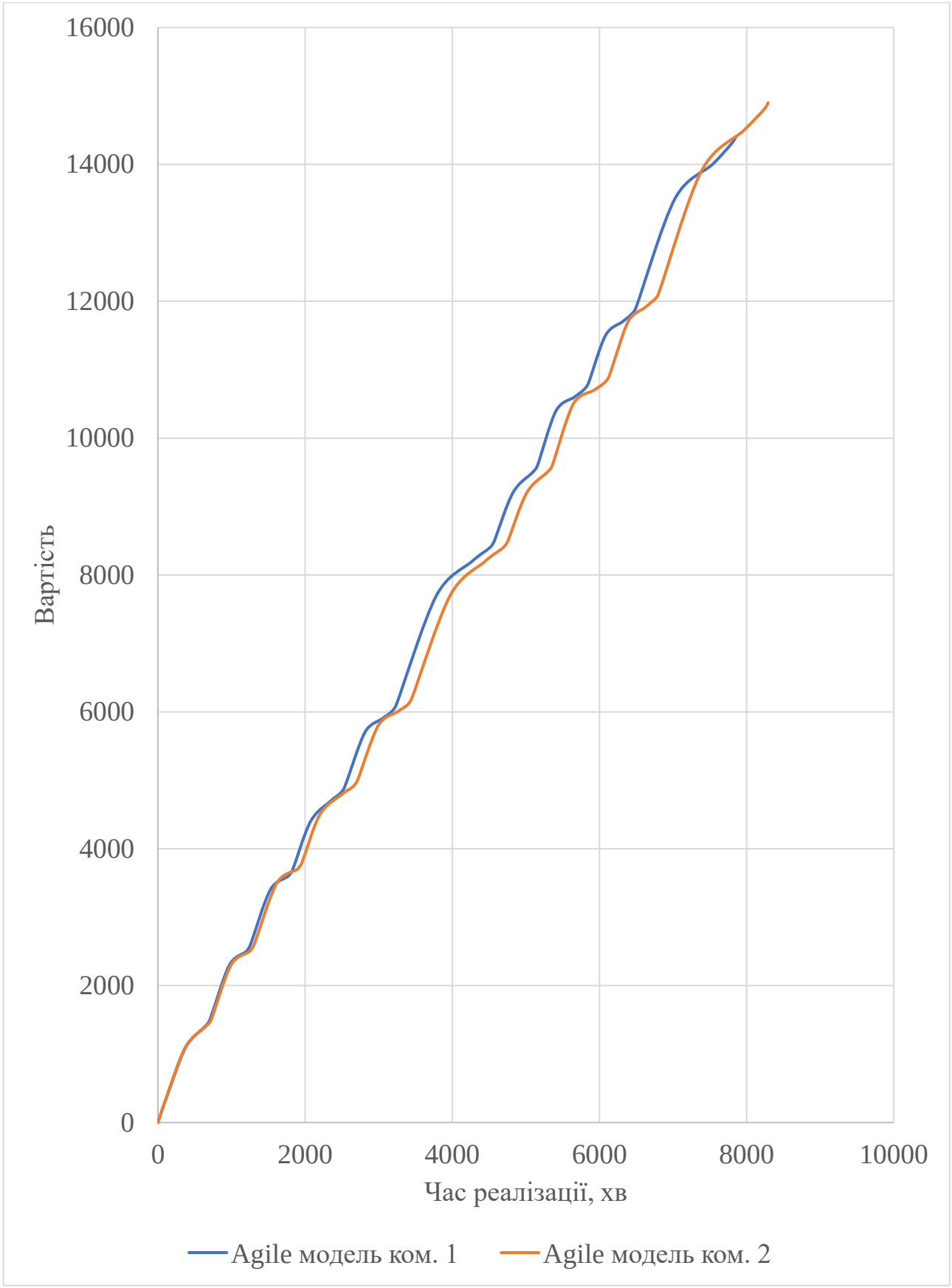

Рисунок 6.15 – Графік розробки програмного продукту для Agile моделі

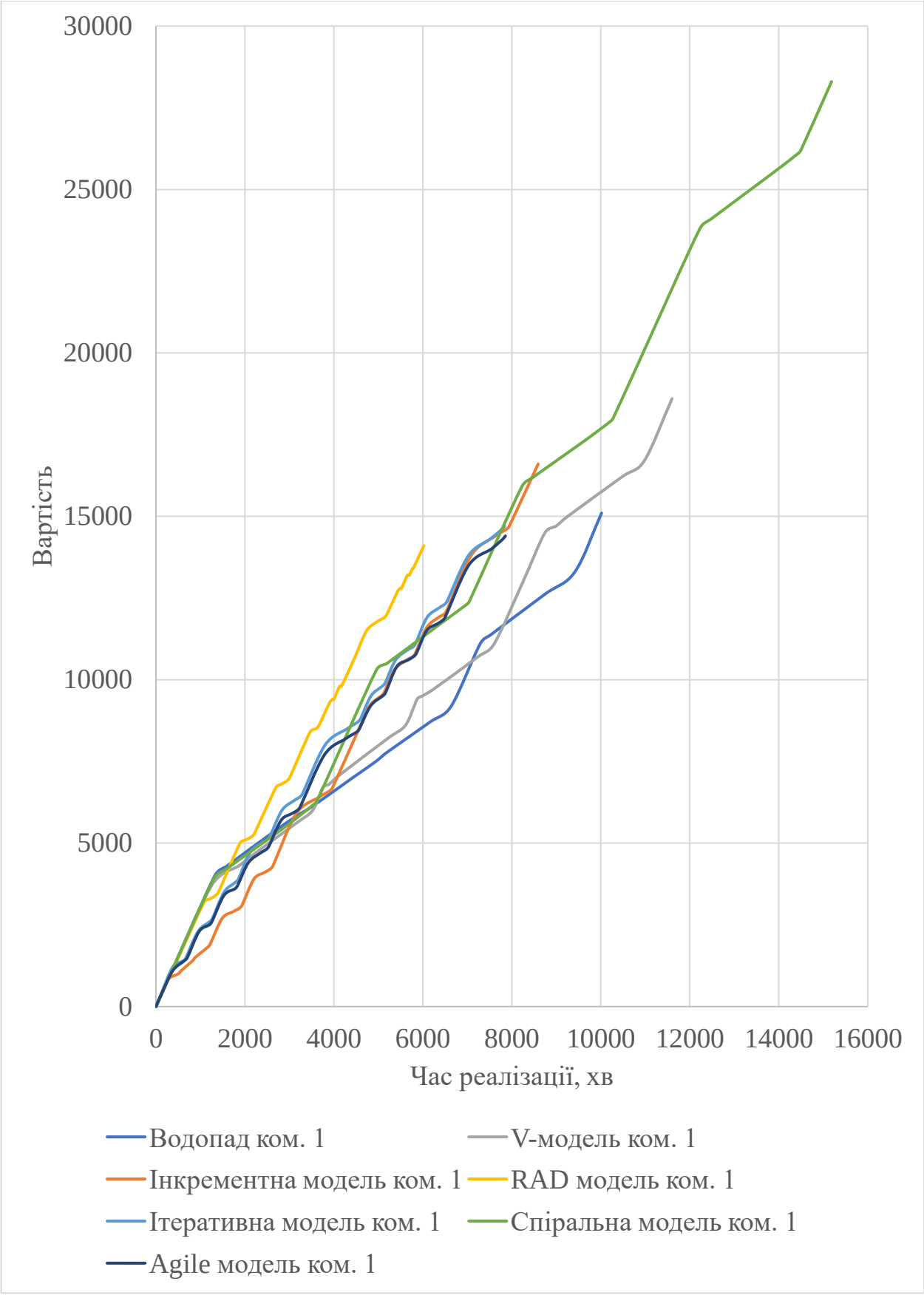

Рисунок 6.16 – Загальний аналіз усіх моделей для ПЗ категорії 2

Загальний аналіз з таблиці 6.16 показує, що процес розробки для кожної моделі має різну фігуру, але можна виявити кращі моделі по показникам загальної вартості та часу розробки.

Таблиця 6.16 – Аналіз показників

Модель	Загальний час, хв	Загальна вартість	Вартість / Час, 1 / хв
Водопад	10016	15100	1,51
V	11601	18600	1,60
Інкрементна	8590	16600	1,93
RAD	6020	14100	2,34
Ітеративна	7838	14700	1,88
Спіральна	15186	28300	1,86
Agile	7855	14400	1,83

Більш детально про порівняння та аналіз цих показників йде далі, розділ 6.4.

6.4 Аналіз отриманих результатів

Для розробки програмного продукту з упором на швидкість та якість роботи програми та документацію порівняємо всі приведені моделі по двом характеристикам: часу та вартості розробки. Тенденція розробки практично однакова для всіх моделей.

Якщо час розробки є найважливішим критерієм успіху програмного продукту, то слід вибирати модель життєвого циклу програмного забезпечення в такому порядку, як Agile, ітеративна, водопад, інкрементна, RAD, V, спіральна моделі. Зазначимо, що Agile очіковано краще за ітеративну модель, а V-модель неочіковано гірше за водопад.

Якщо вартість розробки є найважливішим критерієм успіху програмного продукту, то вибирати модель життєвого циклу програмного забезпечення в такому порядку, як Agile, ітеративна, водопад, інкрементна, V, спіральна моделі, RAD. Тільки RAD модель має непропорційно велеки вартість розробки.

Для розробки програмного продукту з упором на зовнішній вигляд інтерфейсу та зручне використання порівняємо всі приведені моделі по двом характеристикам: часу та вартості розробки. Тенденція розробки практично однакова для всіх моделей, рис. 6.16.

Якщо час розробки є найважливішим критерієм успіху програмного продукту, то слід вибирати модель життєвого циклу програмного забезпечення в такому порядку, як RAD, ітеративна, Agile, ітеративна, інкрементна, водопад, V, спіральна моделі. Зазначимо, що RAD для другої категорії є найкращою моделею, а для першої результат зворотній.

Якщо вартість розробки є найважливішим критерієм успіху програмного продукту, то вибирати модель життєвого циклу програмного забезпечення в такому порядку, як RAD, Agile, ітеративна, водопад, інкрементна, V, спіральна. Результат для двох показників ідентичний.

В загальному випадку для категорії 1 портрібно розглядати тільки такі моделі, як Agile, ітеративну модель та водопад. Для категорії 2 потрібно розглядати тільки RAD, Agile, ітеративну та водопад.

На початку проекту, коли вибирається модель життєвого циклу програмного забезпечення, не завжди можливо визначити, який тип програмного продукту потрібно реалізувати. В такому випадку треба дивитись на стабільність моделі життєвого циклу програмного забезпечення та залежність від категорій програмних продуктів. Так порядок вибору буде наступним: Agile модель, ітеративна модель, модель водопаду, RAD, V-модель, інкрементна модель та спіральна модель. Варто зазначити, що даний порядок зменшує ризики втратити час та збільшити витрати на розробку.

ВИСНОВКИ

У результаті магістерської атестаційної роботи були розглянуті, проаналізовані та порівняні популярні моделі життєвих циклів програмного забезпечення. Було проведено моделювання програмних застосувань, що реалізовані. Були розроблені, проаналізовані та порівняні, з точки зору категорії обраного програмного продукту, часу та вартості розробки, всі основні моделі життєвих циклів програмного забезпечення.

Була аргументована необхідність розробити програмний продукт, завдяки якому можна збирати інформацію про поточні етапи розробки програмного продукту, аналізувати поточну модель життєвого циклу програмного забезпечення, виявити ефективнішу модель, та зробити детальний аналіз отриманих даних. При аналізі предметної області були виявлені технології, які будут використанні при розробці. В якості клієнтської частини обрана мова JavaScript та фреймворк Angular. В якості серверної частини обрана мова Ruby та фреймворк Ruby on Rails. В якості СУБД обрана PostgreSQL. В якості серверу для обробки великої кількості даних обрано сховище Hbase з обробкою інформації за допомогою моделей машиного навчання.

Було виявлено об’єкт дослідження, яким є 7 моделей життєвих циклів програмного забезпечення:

– модель водоспаду;

– V-модель;

– інкрементаційна модель;

– RAD модель;

– ітераційна модель;

– Agile модель;

– спіральна модель.

Проведено аналіз та побудовані діаграми основних етапів усіх моделей. Були виявлені лідери у своіх категоріях.

Побудовані діаграми варінтів використання, класів, активності, розгортання, компонентів та об’єктів розробленого програмного продукту.

Було виявлено, що для категорії програмних продуктів з упором на швидкість на документацію потрібно використовувати такі моделі, як Agile, ітеративну та водопад. Для категорії програмних продуктів з упором на користувальницький інтерфейс та зручність використання потрібно дивитись на такі моделі, як RAD, Agile, ітератинву та водопад. У загальному випадку Agile модель має найменші ризики втратити більше часу на розробку, найбільші ризики має спіральна модель.

Отже, були виконані усі вимоги, поставлені до роботи. Є вся необхідна теоретична база для розробки та порівняння. Проведено аналіз та порівняння лідерів, які виявляються практично однаковими за різних крітеріїв.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Добряк П. Модели жизненного цикла [Текст] / П. Добряк. – СПб.: Символ-Плюс, 2016. – 132 с.

2. Косяков А. Системная инженерия. Принципы и практика [Текст] / А. Косяков. – СПб.: ДМК-Пресс, 2017. – 624 с.

3. S. Cohen. A Software System Development Life Cycle Model for Improved Stakeholders' Communication and Collaboration / S. Cohen, D. Dori, U. de Haan. – Int. J. of Computers, Communications & Control, Vol. V, No. 1 (2010), pp. 20-41.

4. Nayan B. Ruparelia. Software Development Lifecycle Models / Nayan B. Ruparelia. – ACM SIGSOFT Software Engineering Notes, Vol. 35, No. 3 (2010, May), p. 8.

5. T. Bhuvaneswar. A Survey on Software Development Life Cycle Models / T. Bhuvaneswar, S. Prabaharan. – International Journal of Computer Science and Mobile Computing, Vol. 2, Issue. 5 (2013, May), pp. 262-267.

6. V. Rastogi. Software Development Life Cycle Models Comparison, Consequences / Vanshika Rastogi. – International Journal of Computer Science and Information Technologies, Vol. 6, No. 1 (2015), p. 168.

7. Y. Bassil. A Simulation Model for the Waterfall Software Development Life Cycle / Youssef Bassil. – International Journal of Engineering & Technology (iJET), Vol. 2, No. 5 (2012), p. 16.

8. S. Barjtya. A detailed study of Software Development Life Cycle (SDLC) Models / Sahil Barjtya, Ankur Sharma, Usha Rani. – International Journal Of Engineering And Computer Science, Vol. 6, Issue. 7 (2017, July), p. 22097.

9. Фернандес О. Путь Rails. Подробное руководство по созданию приложений в среде Ruby on Rails [Текст] / О. Фернандес. – СПб.: Символ-Плюс, 2009. – 768 с.

10. Хэнссон Д.Х. Гибкая разработка веб-приложений в среде Rails 5.0 [Текст] / Томас Д., Хэнссон Д.Х. – СПб.: Питер, 2008. – 716 с.

11. Фитцджеральд М. Изучаем Ruby та Ruby on Rails на примере приложения [Текст] / М. Фитцджеральд. – СПб.: БХВ-Петербург, 2008. – 336 с.

12. Фаулер М.б Скотт К. UML Основи. – Пер. с англ. – СПб: Символ-Плюс, 2002. – 192с.

13. Мюлер Дж. Р. Проектирование баз данных и UML – Москва: Лори, 2013. – 432 с.

14. Schonig H-J. Mastering PostgreSQL 9.6: A comprehensive guide for PostgreSQL 9.6 developers and administrators – Birmingham: Packt Pubishing, 2017. – 416 p.

15. Назаров С. В. Архитектура и проектирование программных систем: монография 2-е изд., перераб. доп. – Москва: ИНФРА-М, 2016. – 374 с.

16. Dar U., Krosing H., Mlodgenski J., Roybal K. PostgreSQL server programming: extend PostgreSQL using PostgreSQL server programming to create, test, debug and optimize a range of user-defined functions in your favorite programming language – Birmingham: Packt Publishing, 2015. – 312 p.

17. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.. Приемы объектно-ориентированного проектирования. Паттерны проектирования – СПб.: Питер, 2007. – 366 с.

18. Олифер В., Олифер Н., Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. пер. с англ. – СПб.: Питер, 2016, – 992 с.

19. Макконнелл С. Совершенный код. Мастер-Класс. Описание подходов разработки [Текст] / С. Макконнел. – М.: Русская Редакция, 2010. – 896 с.

20. Apache JMeter [Електронний ресурс] / Apache JMeter. URL: http://jmeter.apache.org/index.html (дата зернення 15.05.2019).