

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра системного проектування

До захисту допущено:

Завідувач кафедри

_____ Вадим МУХІН

«___» _____ 2023 р.

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Інтелектуальні сервіс-орієнтовані розподілені обчислювання”
зі спеціальності 122 "Комп'ютерні науки"
на тему: «Створення програмного забезпечення на основі Unit-тестів»

Виконав:

студент ІV курсу, групи ДА-91

Білан Вадим Андрійович _____

Керівник:

доцент кафедри СП, к.т.н.

Харченко Костянтин Васильович _____

Консультант з економічного розділу:

доцент кафедри теоретичної і прикладної економіки ФММ

Рощина Надія Василівна _____

Консультант з нормоконтролю:

доцент

Кирюша Богдан Анатолійович _____

Рецензент:

доцент кафедри ММСА, к.т.н.

Тимощук Оксана Леонідівна _____

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

Студент _____

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра системного проектування

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп’ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

«__» травня 2023 р.

ЗАВДАННЯ

на дипломну роботу студенту

Білану Вадиму Андрійовичу

1. Тема роботи «Створення програмного забезпечення на основі Unit-тестів», керівник роботи Харченко Констянтин Васильович, доцент кафедри СП, к.т.н., затверджені наказом по університету від «30» травня 2023 р. № 2065-с

2. Термін подання студентом роботи - 12 червня 2023 р.

3. Вихідні дані до роботи:

- 1) Draw.io 21.3.7;
- 2) Excel (Office 2019);
- 3) Python 3.4.0;
- 4) ChatGPT May 12 Version;
- 5) C++ 11;
- 6) Visual Studio Community 2019 16.11.7;
- 7) Visual Studio Code 1.78.2;
- 8) Google Test 1.13.0;
- 9) unittest;
- 10) JUnit 4.13.2;
- 11) UML 2.5;
- 12) IntelliJ IDEA Community Edition 2021.2.1;
- 13) Android Studio Electric Eel 2022.1.1 Patch 2;
- 14) Java 17;
- 15) Spring 6.0.9;
- 16) Kotlin 1.8.21;
- 17) Android 10.0;

- 18) Jetpack Compose 1.4.3.
4. Зміст роботи:
- 1) Дослідження та порівняння методологій розробки програмного забезпечення. Автоматизація розробки програмного забезпечення.
 - 2) Дослідження методів представлення вимог до програмного забезпечення.
 - 3) Перевірка існуючих фреймворків для створення модульних тестів та програмного забезпечення.
 - 4) Розробка власної концепції фреймворку для створення програмного забезпечення за модульними тестами та перевірка його можливостей.
 - 5) Перевірка, порівняння та оцінка власного фреймворку.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)
- 1) Презентація до роботи.
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	доцент, к.е.н., Рощина Н.В.		

7. Дата видачі завдання – 01.02.2023

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Отримання завдання	01.02.2023	
2.	Проведення огляду літератури за темою бакалаврської роботи. Вивчення та підбір методологій, фреймворків, засобів для проектування	21.04.2023	
3.	Дослідження та порівняння методологій розробки програмного забезпечення	28.04.2023	
4.	Прогнозування наслідків автоматизації розробки програмного забезпечення	5.05.2023	
5.	Дослідження методів представлення вимог до програмного забезпечення	8.05.2023	
6.	Перевірка існуючих фреймворків для створення модульних тестів та програмного забезпечення	12.05.2023	

7.	Розробка власної концепції фреймворку для створення програмного забезпечення за модульними тестами та перевірка його можливостей	19.05.2023	
8.	Перевірка, порівняння та оцінка власного фреймворку	24.05.2023	
9.	Підготовка економічної частини	26.05.2023	
10.	Оформлення пояснювальної записки дипломної роботи	31.05.2023	
11.	Підготовка презентації доповіді	5.06.2023	
12.	Оформлення допуску до захисту і подання роботи	.06.2023	

Студент

Білан В.А.

Керівник

Харченко К. В.

АНОТАЦІЯ

бакалаврської дипломної роботи Білана Вадима Андрійовича
на тему «Створення програмного забезпечення на основі Unit-тестів»

Метою дипломної роботи є дослідження сучасного стану розробки програмного забезпечення шляхом аналізу та порівняння між собою методологій розробки програмного забезпечення, вивчення напрямку аналізу вимог, вивчення, перевірки практичності використання фреймворків штучного інтелекту на основі мовних моделей; прогнозування майбутнього розробки через перспективи її автоматизації, проектування концепції власного фреймворку створення програмного забезпечення.

Актуальність даної роботи полягає у представленні оригінального погляду на існуючі тренди розвитку штучного інтелекту, які впливають на усі інші напрямки; у визначенні обмежень існуючих мовних моделей, які не дають розвиватися далі; у висвітленні власних ідей щодо можливого вирішення наявних проблем розвитку штучного інтелекту; у перегляді сформованих підходів розробки програмного забезпечення відносно перспектив автоматизації розробки.

У ході роботи було охарактеризовано сучасний стан розробки програмного забезпечення, досліджено методології розробки програмного забезпечення, напрям аналізу вимог, існуючі мовні моделі штучного інтелекту, спрогнозовано перспективи автоматизації розробки. На основі результатів дослідження та прогнозування зроблено висновки про необхідність зміни підходів до створення архітектури моделей штучного інтелекту через існуючі обмеження розвитку. Це допоможе прискорити еволюцію розробки програмного забезпечення. Далі було приведено можливі варіанти усунення обмежень на прикладі власної концепції фреймворку створення програмного забезпечення.

У результаті роботи було створено власну модель порівняння методологій розробки програмного забезпечення; проведено аналіз практичності існуючих фреймворків штучного інтелекту на основі мовних моделей для створення програмного забезпечення; приведено аргументи щодо необхідності змін, удосконалення сучасних парадигм штучного інтелекту, які мають суттєві обмеження на шляху до створення штучного суперінтелекту; створено власну концепцію фреймворку створення програмного забезпечення.

Загальний обсяг роботи 78 с., 36 рис., 8 таблиць, 8 додатків, 49 джерел.

ABSTRACT

Bachelor thesis of Bilan Vadym Andriyovych
on the topic "Creating software based on Unit tests"

The aim of the thesis is to study the current state of software development by analyzing and comparing software development methodologies, studying the direction of requirements analysis, studying and checking the practicality of using artificial intelligence frameworks based on language models; forecasting the future development through the prospects of its automation, designing the concept of own framework for creating software.

The relevance of this work lies in the fact that it presents an original view of the existing trends in the development of artificial intelligence, which affect all other directions; that it defines limitations of the existing language models, which prevent further development; that it highlights my own ideas regarding a possible solution to existing problems in the development of artificial intelligence; that it reviews established software development approaches relative to development automation prospects.

In the course of the work, the current state of software development was characterized, software development methodologies, the direction of requirements analysis, existing language models of artificial intelligence were studied, and prospects for development automation were predicted. Based on the results of research and forecasting, conclusions are made about the need to change approaches of creating the artificial intelligence models architecture due to existing development limitations. This will help accelerate the evolution of software development. Next, possible options for eliminating limitations were given using the example of the own concept of the software creation framework.

As a result of the work, an own model of comparison of software development methodologies was created; an analysis of the practicality of existing artificial intelligence frameworks based on language models for creating software was carried out; arguments are presented regarding the need for changes, improvement of modern paradigms of artificial intelligence, which have significant limitations on the way to creation of artificial superintelligence; own concept of the software creation framework was created.

The total volume of the work is 78 pp., 36 figures, 8 tables, 8 appendices, 49 sources.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ.....	8
ПЕРЕЛІК КЛЮЧОВИХ СЛІВ	8
ВСТУП	9
РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА ПОРІВНЯННЯ МЕТОДОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. АВТОМАТИЗАЦІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.	11
1.1. Аналіз викликів, що стоять перед розробником при створенні програмного забезпечення.....	11
1.2. Поняття методології розробки програмного забезпечення. Існуючі представники та їхній опис.....	13
1.2.1. Поняття методології розробки програмного забезпечення.....	13
1.2.2. Класифікація методологій.	13
1.2.3. Характеристики та критерії оцінювання методологій.....	14
1.2.4. Опис та характеристика обраних методологій.....	15
1.3. Порівняння методологій розробки програмного забезпечення.	31
1.4. Автоматизація розробки програмного забезпечення. Перспективи розвитку напрямку.....	37
Висновки.....	39
РОЗДІЛ 2. ДОСЛІДЖЕННЯ МЕТОДІВ ПРЕДСТАВЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.	40
2.1. Аналіз необхідних компонентів для формалізації, окреслення вимог до майбутнього програмного забезпечення.....	40
2.2. Методи представлення вимог. Модульні тести.....	41
2.3. Порівняння методів між собою.	45
2.4. Визначення перспектив використання методів представлення вимог у автоматизації розробки програмного забезпечення.	47
Висновки.....	48
РОЗДІЛ 3. ПЕРЕВІРКА ІСНУЮЧИХ ФРЕЙМВОРКІВ ДЛЯ СТВОРЕННЯ МОДУЛЬНИХ ТЕСТІВ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.	49
3.1. Аналіз викликів, що стоять перед розробником фреймворку автоматизації розробки.	49
3.2. Існуючі фреймворки для створення модульних тестів та програмного забезпечення.	50
3.3. Розробка тестових прикладів для перевірки фреймворків.	53
3.4. Практична перевірка фреймворків на основі тестових прикладів. Порівняння фреймворків між собою.	54
3.5. Оцінка придатності існуючих фреймворків для прикладного використання.	61
Висновки.....	63

РОЗДІЛ 4. РОЗРОБКА ВЛАСНОЇ КОНЦЕПЦІЇ ФРЕЙМВОРКУ ДЛЯ СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА МОДУЛЬНИМИ ТЕСТАМИ ТА ПЕРЕВІРКА ЙОГО МОЖЛИВОСТЕЙ.	64
4.1. Формулювання вимог до власного фреймворку.	64
4.2. Проектування структури власного фреймворку та створення схем UML для опису системи.	66
4.3. Опис використаних мов, засобів, бібліотек та утиліт.	68
Висновки.....	71
РОЗДІЛ 5. ПЕРЕВІРКА, ПОРІВНЯННЯ ТА ОЦІНКА ВЛАСНОГО ФРЕЙМВОРКУ.....	72
5.1. Перевірка та тестування власного фреймворку на тестових прикладах.....	72
5.2. Порівняння власної реалізації із існуючими фреймворками.	72
5.3. Перспективи подальшої розробки власного фреймворку.	73
Висновки.....	74
РОЗДІЛ 6. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ВЛАСНОГО ФРЕЙМВОРКУ.	75
6.1. Постановка задачі проектування.....	75
6.2. Обґрунтування функцій програмного продукту.	75
6.3. Обґрунтування системи параметрів програмного продукту.	76
6.4. Аналіз експертного оцінювання параметрів.	77
6.5. Аналіз рівня якості варіантів реалізації функцій.	79
6.6. Економічний аналіз варіантів розробки ПП.	79
6.7. Вибір кращого варіанту ПП техніко-економічного рівня.	83
Висновки.....	84
ВИСНОВКИ.....	85
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	87
ДОДАТКИ.....	90
1. Додаток А. patterns.txt.....	91
2. Додаток Б. Зразки коду.docx	95
3. Додаток В. rateTable.xlsx	104
4. Додаток Г. Результати перевірки фреймворків.....	104
5. Додаток Д. Діаграма прецедентів UML власного фреймворку.pdf.....	157
6. Додаток Е. Діаграма класів UML власного фреймворку.pdf	158
7. Додаток Є. Діаграма станів UML циклу життя задачі.pdf	160
8. Додаток Ж. Малюнки, таблиці.docx.....	161

ПЕРЕЛІК СКОРОЧЕНЬ

AGI – artificial general intelligence
API – application programming interface
CPU – central processing unit
DDP – discrete distribution position
FIP – feature intersection power
GPU – graphics processor unit
IDE – integrated development environment
PC – pattern convergence
PM – project manager
UI – user interface
XML – extensible markup language
ЕОМ – електронно обчислювальна машина
ЖЦПЗ – життєвий цикл програмного забезпечення
МРПЗ – методологія розробки програмного забезпечення
ПЗ – програмне забезпечення
ПП – програмний продукт
ФВА – функціонально-вартісний аналіз
ШІ – штучний інтелект

ПЕРЕЛІК КЛЮЧОВИХ СЛІВ

- 1) Розробка ПЗ;
- 2) методологія розробки ПЗ;
- 3) автоматизація розробки ПЗ;
- 4) загальний штучний інтелект;
- 5) модульні тести;
- 6) фреймворк створення ПЗ;
- 7) мовна модель.

ВСТУП

За свою історію напрямок розробки програмного забезпечення еволюціонував від перемикання тумблерів, практик ентузіастів до контролю серверних кластерів, відточених корпоративних методологій. Це колосальний стрибок за 70 років. Це все завдяки специфіці розробки ПЗ. На ріст ефективності розробки не так впливають фізичні обмеження як на інші галузі. Настав переломний момент, коли обчислювальної потужності вистачало на повсякденні задачі. Тому розробники могли концентрувати свої зусилля не тільки на оптимізацію програм, а й на удосконалення власної екосистеми, інструментів праці. Підвищувалася зручність програмування, а з нею швидкість та масштаб випуску нових продуктів.

Нині розробники мають відточені алгоритми, методики для вирішення будь-яких проблем пов'язаних із проектуванням, розробкою, підтримкою ПЗ для широкого спектру задач. Розгортання систем та налаштування екосистеми не є проблемою, адже для цього є усі компоненти фізичної інфраструктури. Усе впирається лише у економічні обмеження. Але для вирішення більшості задач буде достатньо і обладнання, що доступне кожній людині.

Здавалося б, люди вигадали рішення для будь-якої своєї задачі. Але вирішувати ці задачі, створювати код мають також вони самі. Прогрес не стоїть на місці, усі ланцюги розробки ПЗ постійно аналізуються на пошук збиткових факторів. І хоч багато із цих факторів було усунуто або трансформовано є один вагомий фактор, який дуже уповільнює процес розробки, через нього виникають помилки. Цей фактор - людина. Люди мають цілий ряд обмежень: втома, задоволення потреб, швидкість мислення і дій.

Комп'ютер на багато порядків швидший за людину, не має тих самих обмежень. Але поки що не такий енергоефективний як мозок і не володіє такою гнучкістю, складністю мислення. Це питання часу, коли людину замінить комп'ютер. Люди або будуть замінені комп'ютерами на робочих місцях, або оптимізують людино-машинну взаємодію настільки, щоб злитися у єдину систему. Інших варіантів немає.

Нині швидко розвиваються такі напрямки як людино-машинний інтерфейс, загальний штучний інтелект. Такі сервіси як ChatGPT 3.0/4.0, Stable Diffusion вже проявляють потенціал у заміні людини на інтелектуальних професіях або потенціал максимального спрощення роботи. Ці напрямки дають прискорення розвитку для інших галузей. Це не омине й розробку програмного забезпечення. У майбутньому буде переважати автоматична розробка, люди будуть тільки відповідальні за формування вимог. Навіть більше, комп'ютер буде сам адаптуватися до викликів цивілізації, буде шукати актуальні задачі, формувати вимоги до них і знаходити рішення.

Все більше зростає роль нових прогресивних ідей, адже простір для чогось нового постійно скорочується через великий попит суспільства на нові ідеї як у роботі так і на дозвіллі. Люди пробують вирішувати все складніші й складніші задачі. Ці задачі настільки комплексні, що для їхнього вирішення необхідні зусилля великих концернів, наукових центрів із тисячами спеціалістів. Через людський фактор це займає усе більше часу, ресурсів. Потенційно це може загрожувати стагнацією, зупинкою розвитку, бо інтелектуального потенціалу буде не достатньо, а обмеження зростуть настільки, що долати нові перепони буде неможливо. Необхідний новий поштовх, у вигляді інтелектуального агента, який краще за

людину зможе впоратися із такою складністю. Наприклад, загальний штучний інтелект, суперінтелект.

Задум роботи: показати, що розробка ПЗ у майбутньому зазнає великих змін, вказати напрямки її розвитку. Це відбувається за допомогою дослідження існуючих методологій розробки ПЗ, методів представлення вимог та за допомогою проектування власного фреймворку і перевірки його концепту.

Цілі та завдання які ставляться:

1. проаналізувати існуючі методології розробки програмного забезпечення, засоби представлення вимог і на основі цього оцінити їхні перспективи для майбутніх трансформацій напрямку розробки ПЗ – автоматизації розробки.
2. проаналізувати та перевірити сучасний стан автоматизації розробки, а саме фреймворки створення тестових прикладів, програмного забезпечення.
3. спробувати побудувати власну реалізацію фреймворку для автоматизації розробки ПЗ і перевірити її перспективність, придатність до практичного використання.

Робота поділена на 5 розділів, у кожному із яких розглядаються такі питання:

1. Означення методології розробки ПЗ. Опис обраних методологій та їхнє порівняння за визначеними критеріями. Оцінка обраних методологій з боку автоматизації розробки.
2. Визначення що таке вимоги і їхні компоненти, класифікацію. Опис методів представлення вимог та їхнє порівняння. Визначення перспектив цих методів у автоматизації розробки.
3. Визначення вимог до фреймворку автоматизації розробки. Опис обраних фреймворків та їхнє порівняння за допомогою тестових прикладів. Оцінка придатності фреймворків.
4. Визначення вимог до власного фреймворку. Проектування, фреймворку.
5. Тестування, валідація власного фреймворку. Порівняння власного фреймворку із іншими. Оцінка наявних проблем. Перспективи розвитку власного фреймворку.

Результатом роботи мають стати: оцінки перспектив сучасних методологій розробки ПЗ та власний фреймворк автоматизації розробки ПЗ, який має інноваційні ідеї щодо полегшення розробки ПЗ.

Власний фреймворк має подальший потенціал для розвитку. Ідеї вкладені у нього (що буде показано далі) будуть направляти мої зусилля у розробку власного інтелектуального агента після закінчення даної роботи.

Сподіваюсь читачам сподобається хід моїх думок та напрямки, на які я звертаю увагу, щоб задуматися над майбутнім автоматизації розробки.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА ПОРІВНЯННЯ МЕТОДОЛОГІЙ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. АВТОМАТИЗАЦІЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

1.1. Аналіз викликів, що стоять перед розробником при створенні програмного забезпечення.

Щоб спрогнозувати майбутнє розробки ПЗ, потрібно дослідити теперішній стан цього напрямку. А саме із якими викликами зустрічаються розробники у процесі розробки ПЗ та яка теорія напрацьована в плані контролю розробки.

Доволі велика частина задач нині вирішується створенням програми і з кожним роком така частка збільшується. Задачі можуть виникати у рамках проблем, обмежень, факторів, із якими зустрічаються люди. Тому люди, звертаються за вирішенням цих задач до розробників. У цій ситуації люди, яким необхідно вирішити певну задачу, виступають у ролі клієнтів, замовників. Клієнтами можуть бути як окремі люди, так і цілі компанії. Іноді розробник може самостійно давати собі завдання для вирішення своїх внутрішніх проблем, наприклад, керівництво вирішило випустити новий продукт, один відділ сформував вимоги для його створення, а інший має його розробити.

Розробником може бути як окрема людина, маленька команда, так і ціла компанія. Розробник може брати на себе функції проєктувальника, виконавця, прикладного експерта, тестувальника, адміністратора одночасно.

Якщо поглянути на сучасний процес розробки ПЗ із боку керівника відділу, компанії розробки, рівень складності контролю усіх моментів є великим. Можна тільки уявити людину, яка могла б усе зробити правильно і в поставлені терміни.

Коли замовник приходить до розробника із проблемою, він не має точного її опису або навіть може не мати достатньої компетенції у формулюванні задачі. Такому клієнту потрібно ставити багато уточнюючих запитань. Потрібно мати уявлення та досвід в уточненні завдань. У розробника може бути багато таких клієнтів і всі вони роблять коригування під час самої розробки.

Для задачі слід виділити спеціалістів, їхній час, визначити, які необхідні ресурси, кошти, інструменти, вибудувати екосистему за потреби. Для самої команди потрібно сформувати план, графік роботи, поставити адекватні вимоги та обмеження, розподілити завдання враховуючи сильні і слабкі сторони кожного.

Під час розробки можуть виникнути затримки, збої, помилки, зайві втрати, ситуація може повернутися зовсім у іншу сторону. Окремі члени команди можуть мінятися, приходити нові і йти - старі. Для роботи буде не вистачати рук, ресурсів.

Усе це слід вирішувати швидко, у короткий час, у конкретні моменти, іноді одночасно. Це потребує від керівника серйозної підготовки, *hard, soft skills*, управління часом. Якщо керівник не буде робити усе коректно, наслідками цього можуть стати: незбалансований графік, розподіл завдань, перевантажена команда, економічні втрати, втрати клієнтів. Окремі працівники будуть не задоволені ходом роботи, тому команда може поступово розпуститися, а виконувати роботу буде нікому або вже запізно. Невмілий керівник – збиткова компанія.

Увесь цей опис проблем під час процесу розробки ПЗ може породити хаос без належного контролю і підходу до роботи як керівника, так і усієї команди. Тому слід впроваджувати загальний задум розробки, робочі практики, що відповідає характеру команди, окремим особам, темпу розробки, характеру завдань. Це обширна область, яка актуальна для усіх через складність процесів і масштаби у сучасному ІТ секторі. Для неї має бути окрема галузь наук і напрямків, що охоплюють вище згадані проблеми. Як і кожна наука така галузь допоможе досягати поставлених на неї цілей: стабілізувати, зробити контрольованою розробку із приналежною наукам методичністю, деталізованістю.

Така галузь вже існує і це дисципліна Розробка програмного забезпечення. Вона ділиться на такі розділи, що охоплюють усі проблеми розробників¹:

- Вимоги до ПЗ.
- Проектування ПЗ.
- Інженерія ПЗ.
- Тестування ПЗ.
- Обслуговування ПЗ.
- Керування конфігурацією ПЗ.
- Керування розробкою ПЗ.
- Процес розробки ПЗ.
- Інструменти розробки ПЗ.
- Якість ПЗ.
- Локалізація ПЗ.

Найвагоміші розділи, які формують увесь процес розробки: Вимоги до ПЗ, Проектування ПЗ, Інженерія ПЗ, Керування розробкою ПЗ, Процес розробки ПЗ. Надалі нас будуть цікавити саме ці розділи. Вони найбільше пов'язані із автоматизацією розробки у загальному плані.

За час розвитку розробки ПЗ серед спеціалістів розвинулися цілі системи практик та методик, що формують розробку за власними нормами та ідеями. Вони покликані вирішити конкретні проблеми, та, як усе у цьому світі, мають свої мінуси. Загалом їх можна назвати методології розробки програмного забезпечення. Кожна методологія займається питаннями із декількох вище описаних розділів. За допомогою них можна класифікувати місце кожної методології розробки ПЗ.

Далі буде розглянута класифікація, вибір критеріїв оцінки та опис методологій розробки ПЗ. Це робиться для того, щоб:

- оцінити вагомість автоматизації розробки, яка у майбутньому усуне більшість описаних проблем;
- виділити збиткові та важливі якості методологій, як передових методик ведення розробки ПЗ.

¹ Розробка програмного забезпечення. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 01.05.2023).

1.2. Поняття методології розробки програмного забезпечення. Існуючі представники та їхній опис.

1.2.1. Поняття методології розробки програмного забезпечення.

Методологія розробки програмного забезпечення – це система норм, методів, практик організованих навколо певної ідеї, що застосовуються на різних етапах життєвого циклу розробки програмного забезпечення для вирішення конкретних проблем, які критично впливають на використання ресурсів, проектування рішень, показники, ефективність розробки, якість результату².

Під час виконання повсякденних задач розробники напрацьовували корисні практики, які акумулювали, формалізували у вигляді деталізованих документів. Це відбувалося у рамках однієї компанії або виражалося зусиллями групи людей. Ці документи можна назвати першоджерелами методологій, від яких наслідували принципи інші спеціалісти.

1.2.2. Класифікація методологій.

У наступних розділах доведеться порівнювати між собою у різних сценаріях МРПЗ. Порівнювати між собою моделі легше, коли є уявлення їхньої форми і місця у загальній схемі. Це визначається розмежуванням моделей між собою на групи, класифікацією.

У просторі Інтернету не було знайдено детальної схеми, діаграми класифікації МРПЗ. Тому було вирішено створити власну. Під час дослідження, розгляду, роботи над описом обраних МРПЗ, було виділено основні аспекти, що відрізняють їх між собою. Ці аспекти виділяються під час розгляду типових сценаріїв використання, обмежень, вразливостей, особливостей кожної методології.

Загалом власна класифікація представляє собою набір критеріїв, за якими відбувається поділ на групи. Вона була сформована після дослідження усіх МРПЗ:

Рис. 1.1. Власна класифікація МРПЗ. (Додаток Ж)

Критерії поділу за схемою:

- за повторюваністю – розподіл методології залежить від того чи повторюються етапи життєвого циклу, за кількістю циклів;
- за адаптивністю – поділ за тим чи передбачає методологія зміни, чи має строгий план, чи може коригувати процес розробки динамічно, коли відбувається планування процесів, які наслідки від змін, виникнення непередбачених раніше проблем;
- за тривалістю – поділ за розміром періоду життєвого циклу проекту, ПЗ;
- за розміром команди – поділ за кількістю осіб в команді, які передбачає методологія;
- за централізованістю – поділ за тим як приймаються рішення тільки менеджерами – централізовано, чи командою також - децентралізовано;

² Методологія розробки програмного забезпечення. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%8F_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 01.05.2023).

- особливі – категорії, особливості методологій, які не передбачені у інших поділах. Метамоделі – це моделі, які передбачають використання інших методологій, переймання їхніх можливостей і процесів. Орієнтовані – це моделі, які концентруються на певні показники, артефакти, наприклад, на швидкість розробки, вчасність, ощадливість, втрати, тести, властивості, моделі, поведінку системи тощо.

1.2.3. Характеристики та критерії оцінювання методологій.

Без визначення критеріїв порівняння неможливо коректно провести саме порівняння методологій. Введення таких критеріїв дасть змогу побудувати умовний рейтинг методологій та сформуванати найкращу комбінацію методологій для кожного сценарію використання.

Кожна методологія, яка намагається впливати на процес розробки, стикається із усіма аспектами цього процесу. Але одночасно показати на всіх напрямках кращий результат неможливо – деякі аспекти протирічають одне одному. Тому усі методології мають розставити пріоритети по кожному із цих аспектів. Для кожної методології буде складено список аспектів за пріоритетами. Він буде формуватися на основі описів. Такі аспекти як намагання задовольнити клієнта, досягнення цілей проекту, які виконуються усіма методологіями, не будуть враховуватися.

Список аспектів, які будуть використовуватися:

- якість – чим вище пріоритет цього аспекту, тим більше приділяється увага якості, тестуванню, верифікації, досягненню певних показників порівняно із іншими аспектами;
- стійкість до змін – чим вище, тим гнучкіша розробка, лояльніші клієнти, більша стабільність, стійкість у мінливому середовищі, можливість експериментувати;
- процес розробки – чим вище, тим легше команді вести процеси, кращий розподіл задач, легше ввести когось у курс справ, планування виглядає елегантніше;
- економія – чим вище, тим більший контроль втрат, легше вкладатися у рамки, переживати наслідки і уникати їх;
- час – чим вище, тим краще планування, блочність, контрольованість процесів, вчасність релізів, стійкість до дедлайнів;
- команда – чим вище, тим легше вводити у курс справ новачків, переживати зміни у команді, децентралізованіші процеси;
- деталізація(уточнення) – чим вище, тим менша ймовірність створити зайвий функціонал(менші втрати), легша верифікація, тестування, розуміння командою процесів;
- контрольованість – чим вище, тим централізованіші процеси, більша відповідальність керівництва, збільшення формальностей, але й легше спрямовувати процеси, реагувати на проблеми, направляти новачків, організовувати команду;
- зворотній зв'язок – чим вище, тим більше задоволені клієнти, користувачі, чіткіше вирізняється перспективність, ціна продукту, можна вчасно коригувати процеси.
- ризики – чим вище, тим менші втрати, вища стабільність процесів, менша ймовірність провалу, більша упевненість в успіху, більша довіра з боку клієнтів.

Найголовніше у списках пріоритетів є відношення між аспектами. За методологією розробники будуть жертвувати аспектами нижчих пріоритетів заради вищих, що у результаті усе одно дасть кращий результат для найвищого пріоритету і спадаючий результат для нижчих.

Не може існувати найкращої методології, адже існують різні сценарії використання із різним розміром команди, бюджетом, тривалістю проекту, мінливістю, визначеністю вимог тощо. Водночас для кожного сценарію використання методологій можна підібрати свій необхідний патерн пріоритетів і порівняти його із індивідуальними патернами методологій. Наближення цих патернів і буде ключовим критерієм. На основі цього буде відбуватися характеристика, складання рейтингу, підбір найкращих методологій.

Порівняти методології за результатами проектів не вийде, адже на приклади використання методологій впливало багато прихованих факторів. Ці фактори спотворюють результати і роблять порівняння контрпродуктивним заняттям. Результати завершення проектів на основі певної методології дуже залежать від способу використання, відхилень, порушень ведення практик.

Створений критерій буде вести порівняння краще тому, що виражається із позиції зручності використання, збільшення ефективності, виявлення кращих сторін методологій для різних сценаріїв використання, аніж визначення ймовірних результатів. Власний критерій показує, де методологія найкраще проявляє себе і, ймовірно, дасть кращі результати за аспектами розробки із найвищим пріоритетом у проекті. Разом із цим критерієм буде використовуватися класифікація.

1.2.4. Опис та характеристика обраних методологій.

Waterfall

Waterfall (каскадна модель) – це послідовна МРПЗ, основною ідеєю якої є послідовне виконання та повне завершення кожного етапу ЖЦПЗ. Переробити або повернутися на попередні кроки не дозволяється, адже це тільки збільшить кількість помилок. Походить від системної інженерії із виробничих та архітектурних галузей, де кожен етап залежить від результатів попереднього, тому дотримання послідовності є важливим³. ЖЦПЗ є стандартним:

- спершу формуються вимоги до проекту;
- потім проектується структура, архітектура майбутньої системи, уточнюються деталі реалізації;
- створюються усі компоненти системи до стану завершеності;
- перевіряється працездатність, відловлюються і виправляються помилки, виконується валідація системи;
- система супроводжується протягом часу розгортання, експлуатації і згорання.

Рис. 1.2. Форма ЖЦПЗ Waterfall. (Додаток Ж)

Така МРПЗ не підходить для звичайної ІТ компанії, адже клієнти постійно змінюють вимоги, система постійно еволюціонує, удосконалюється, окремі етапи ЖЦПЗ повторюються і формують складну схему. Використовуючи Waterfall буде обмежувати такі компанії.

Водночас для військового, медичного, інженерного, критичного секторів ПЗ має бути добре відточеним, його реалізації будуть змінюватися дуже рідко. Зміни протягом ЖЦПЗ

³ Водоспадна модель. Вікіпедія(uk) : веб-сайт. URL:

https://uk.wikipedia.org/wiki/%D0%92%D0%BE%D0%B4%D0%BE%D1%81%D0%BF%D0%B0%D0%B4%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C (дата звернення: 01.05.2023).

можуть породити приховані проблеми, що не припустимо. Тут Waterfall є одним із зразкових МРПЗ.

Критерієм якості розробки для такої моделі є точність виконання специфікацій технічного завдання.

Цілі: створити єдину, стабільну реалізацію на довгий період, шляхом послідовного доведення до готовності кожного етапу.

Переваги:⁴

- + відсутність виправлень;
- + добре продумана документація;
- + чітка модель;
- + низькі вимоги до спеціалістів.

Недоліки:

- високі вимоги щодо завершення кожного етапу;
- неприпустимість змін;
- накопичення зайвих компонентів;
- поділ спеціалістів на високо і низько кваліфікованих.

Використовується: у проектах критичних галузей, як от військова справа, інженерія, виробництво, архітектура, це включає розробку ПЗ також.

V-Model

V-Model – це МРПЗ, похідна від Waterfall, що бере за мету полегшити перевірку системи та розуміння складності її розробки, шляхом розмежування етапів тестування відносно специфікацій попередніх етапів ЖЦПЗ⁵. Була розроблена незалежно ІАВГ(Німеччина) та Національною радою з системної інженерії(США) у кінці 1980-х років. Спершу використовувалася для оборонної галузі, супутникових систем. Потім перейшла до цивільного сектору.

ЖЦПЗ виглядає так:

- спершу йде визначення системи – формуються концепції, вимоги, проектується архітектура, уточнюється дизайн. Це ліва сторона літери V;
- реалізується за сформованими матеріалами сама система. Основа V;
- далі виконується тестування системи, її верифікація, валідація, впровадження. Це права частина V.

Порівняно із Waterfall тут тестуванню приділяється більша увага. Воно поділяється на етапи, кожен з яких пов'язаний із етапами визначення системи:

- приймально-здавальні випробування ґрунтуються на вимогах;
- системне тестування — на вимогах та архітектури;
- комплексне тестування — на вимогах, архітектурі та інтерфейсах;
- компонентне тестування — на вимогах, архітектурі, інтерфейсах та алгоритмах.

Як і в Waterfall після переходу до наступного етапу повернутися вже не можна.

⁴ 12 кращих методологій розробки програмного забезпечення з перевагами та недоліками. Smart IT: веб-сайт. URL: <https://www.smart-it.com/uk/2021/08/12-best-software-development-methodologies-with-pros-and-cons/> (дата звернення: 06.05.2023).

⁵ V-Model. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/V-Model> (дата звернення: 01.05.2023).

Рис. 1.3. Форма ЖЦПЗ V-Model. (Додаток Ж)

Цілі: мінімізація ризиків, підвищення та гарантія якості, зменшення вартості, якість комунікації між учасниками.

Переваги:

- + V-Model постійно удосконалюється і має зворотній зв'язок;
- + не залежить від типів організацій та проектів;
- + дозволяє розбити роботу на кроки із необхідною деталізацією та інструкціями.

Недоліки:

- не враховано управління, обслуговування, утилізація системи;
- модель стосується самої розробки ПЗ, аніж організації процесів.

Використовується: у великих компаніях.

Iterative та Incremental

Ітеративна та інкрементна розробка – це основа циклічних МРПЗ, яка бере за мету виправити слабкі сторони послідовних МРПЗ типу Waterfall⁶. Процес починається з початкового планування і закінчується впровадженням (готового ПЗ) з циклічними взаємодіями між цими етапами. Процес роботи виглядає так: розробники планують заздалегідь увесь ЖЦПЗ таким чином, щоб сформувати цикл із основних етапів, які будуть повторюватися упродовж усього процесу розробки. На кожному етапі виконується лише та частина роботи, що запланована на цей цикл. У певний момент циклу відбувається розгортання, оновлення вже задіяної системи.

Обидві моделі розбивають усю розробку на цикли. Але існує різниця між роботами, що ведуться впродовж одного циклу та способами розгортання. В інкрементній моделі мета одного циклу – повна реалізація одного компоненту системи. В ітеративній моделі метою одного циклу є виконання одного типу задач, робіт для кожного компоненту системи, таким чином щоб вийшла робоча версія системи. Це основна різниця між ними. Але зазвичай вони комбінуються разом.

Шляхом циклічного удосконалення системи із малими проміжками часу можна зробити розробку гнучкою, адаптивною, враховувати нові фактори і виправляти помилки при цьому це не буде критично впливати на результат.

Рис. 1.4. Форма ЖЦПЗ Iterative/Incremental. (Додаток Ж)

Цілі: зробити розробку спланованою, гнучкою, адаптивною до непередбачуваних факторів, розподілити роботу рівномірно, зробити результуючу систему відповідною до вимог, що постійно змінюються.

⁶ Ітеративна та інкрементна розробка. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%86%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0_%D1%82%D0%B0_%D1%96%D0%BD%D0%BA%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%BD%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0 (дата звернення: 02.05.2023).

Переваги:⁷

- + працююча версія ПЗ з'являється на ранніх стадіях розробки;
- + вимоги можна змінювати на будь-якому етапі розробки;
- + циклічний процес легше тестувати і супроводжувати, без накопичення помилок.

Недоліки:

- через змінні вимоги визначити спочатку оптимальну архітектуру важко;
- потрібно продумати алгоритми інтеграції нових вимог, вирішення нових проблем;
- під час планування розробки потрібно передбачити необхідний обсяг ресурсів із запасом через можливість коригування процесів.

Використовується: для великих проектів, коли відомі хоча б ключові вимоги або вимоги постійно змінюються.

Spiral

Spiral – це генератор моделі процесів, створений Баррі Бомом і вперше описаний у 1986 році в статті «Спіральна модель розробки та поліпшення програмного забезпечення»⁸. Модель концентрується на аналізі ризиків.

Спіральна модель є шаблоном МРПЗ, що комбінує засоби ітеративної та каскадної моделей. Розробка поділяється на 4 сектори, що представляють її окремі етапи: визначення цілей, оцінка ризиків, розробка і тестування, планування нової ітерації.

Старт проекту починають із визначення вимог за якими формують ЖЦПЗ. На початку кожного циклу відбувається аналіз ризиків, за яким може коригуватися ЖЦПЗ, де комбінуються підходи інших методологій. ЖЦПЗ відображається як спіраль, яка, розпочавшись на етапі планування, розкручується з проходженням кожного наступного кроку. У кінці кожного витка має бути протестований робочий прототип. Після виконання усіх вимог – проект виходить до релізної стадії.

Рис. 1.5. Форма ЖЦПЗ Spiral. (Додаток Ж)

Цілі: коректно оцінити ризики, щоб на основі них спланувати відповідний ЖЦПЗ, який буде давати робочі прототипи кожного циклу.

Переваги:⁹

- + покращений аналіз ризиків;
- + деталізована документація;
- + гнучкість процесу розробки, що дає змогу вносити зміни навіть на пізніх стадіях;
- + створення прототипів на ранніх стадіях.

Недоліки:

- дорога у використанні;
- вимагає залучення професіональних фахівців через управління ризиками.

⁷ Ітеративна модель (iterative model). QALight : веб-сайт. URL: <https://qalight.ua/baza-znaniy/iterativna-model-iterative-model/> (дата звернення: 04.05.2023).

⁸ Спіральна модель. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D1%96%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C (дата звернення: 02.05.2023).

⁹ Спіральна модель (spiral model). QALight : веб-сайт. URL: <https://qalight.ua/baza-znaniy/cpiralna-model-spiral-model/> (дата звернення: 04.05.2023).

- успіх залежить від аналізу ризиків.
- не підходить для невеликих проектів.

Використовується: коли важливий аналіз ризиків і витрат, великі проекти із довгим періодом ЖЦПЗ без чітких вимог або із можливістю їхньої зміни, коли розробляється новий продукт.

DSDM (Dynamic Systems Development Method)

Dynamic Systems Development Method – це ітеративний і інкрементний МРПЗ, що базується на концепції швидкої розробки додатків (Rapid Application Development, RAD)¹⁰. Входить до сімейства Agile розробки. Концентрується на взаємодії із користувачем¹¹. Основні методики: тайм-боксинг – вкладення певної одиниці роботи у терміни виконання, MoSCoW – пріоритизація цілей під час розробки.

Життєвий цикл роботи над проектами складається із 3 етапів:

- передпроектного – визначення ймовірних проектів, визначення бюджету, команди;
- життєвий цикл проекту – реалізація проекту;
- постпроектна – підтримка системи.

Життєвий цикл проекту складається із 4 етапів:

- дослідження цінності проекту;
- створення функціональної моделі – узгодження планів, створення функціонального прототипу(не реалізації, а виду майбутньої системи);
- проектування та розробка – визначення виду конструктивного прототипу, узгодження планів, створення прототипу, аналіз прототипу;
- реалізація – перевірка системи користувачами, ознайомлення користувачів із системою, реалізація релізної версії, аналіз ринку.

За потреби можна повторити певний етап циклу роботи, або повернутися до попереднього етапу. ЖЦПЗ формують життєвий цикл проекту та постпроектний етап.

Рис. 1.6. Форма життєвого циклу проекту DSDM. (Додаток Ж)

Цілі: здати проект вчасно, вкластися у бюджет.

Переваги:¹²

- + користувачі можуть контролювати процес розробки.
- + легше отримати обернений зв'язок.
- + покращується і прискорюється взаємодія із замовником.
- + система швидко набуває здатність виконувати свої задачі.

Недоліки:

- вимагає додаткових витрат.

¹⁰ DSDM. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/DSDM> (дата звернення: 02.05.2023).

¹¹ 9 основних методологій розробки програмного забезпечення Agile. Agile Yakubovsky Blog : веб-сайт. URL: <https://agile.yakubovsky.com/ua/2015/10/9-osnovnykh-metodolohiy-rozrobky-prohramnoho-zabezpechennya-agile/> (дата звернення: 01.05.2023).

¹² 12 кращих методологій розробки програмного забезпечення з перевагами та недоліками. Smart IT: веб-сайт. URL: <https://www.smart-it.com/uk/2021/08/12-best-software-development-methodologies-with-pros-and-cons/> (дата звернення: 06.05.2023).

– невеликим організаціям не підходить.

Використовується: великими організаціями; коли необхідно діяти в рамках обмеженого бюджету; коли не можна порушувати терміни.

RUP (Rational Unified Process)

Rational Unified Process – це ітеративна МРПЗ, фреймворк процесу розробки, що створений компанією Rational Software¹³. Джерелом інструкцій для цієї методології слугує відповідний продукт цієї компанії, що містить базу знань, приклади артефактів, описи видів діяльності. Поєднує такі практики: ітеративна розробка, керування ризиками, управління вимогами, компонентна архітектура, візуальне моделювання, регулярна перевірка якості, контроль змін, адаптація.

RUP заснований на наборі будівельних блоків, що описують те, що повинно бути зробленим, необхідні навички, та покрокове пояснення того, як досягаються конкретні цілі розробки: ролі (хто), робочі продукти (що), завдання (як).

Чотири фази життєвого циклу проекту, які повторюються:

– початкова – оцінка системи, бюджету, встановлення прикладів успіху, факторів успіху, формуються база для діаграм, планів проекту, характеристики майбутнього прототипу тощо. Сторони узгоджують базові моменти майбутнього проекту. Результатом є прийняття/неприйняття проекту.

– уточнення – проект набуває форми, виконується аналіз предметної області, створюється база для архітектури, план розвитку проекту.

– конструювання – створення програмної системи. Результатом є створення функціонуючої системи.

– впровадження – переведення системи із розробки у готовий продукт, його тестування, валідація, перевірка якості. Результатом є реліз ПЗ.

Розробка ділиться на такі дисципліни, які виконуються паралельно у процесі роботи над проектом: бізнес-моделювання, вимог, аналізу та проектування, реалізації, тестування, розгортання. Ці дисципліни у послідовних методологіях були окремими етапами ЖЦПЗ, а тут виконуються паралельно.

Кожна дисципліна має власний розподіл концентрації роботи із кожною ітерацією. Розробка концентрується на конкретній дисципліні і з часом плавно переходить у наступну. Тобто пріоритет дисциплін постійно змінюється, але робота над іншими напрямками продовжується.

В кінці кожної ітерації (яка в ідеалі продовжується від 2 до 6 тижнів) проектна команда повинна досягти запланованих на дану ітерацію цілей, створити або доопрацювати проектні артефакти і отримати проміжну, але функціональну версію кінцевого продукту. Ітеративна розробка дозволяє швидко реагувати на зміну вимог, виявляти та усувати ризики на ранніх стадіях проекту, а також ефективно контролювати якість створюваного продукту.

Рис. 1.7. Форма ЖЦПЗ RUP. (Додаток Ж)

¹³ Rational Unified Process. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/Rational_Unified_Process (дата звернення: 02.05.2023).

Цілі: одразу оцінити і узгодити майбутній проект, задовольнити зацікавлені сторони, розподілити роботу як у часі, так і у просторі, створити якісний продукт.

Переваги:¹⁴

- + Можна вести етапи розробки паралельно;
- + Проект можна скасувати перед самою розробкою, у разі його збитковості.
- + Можна спланувати розподіл роботи і зробити це доволі точно, щоб полегшити навантаження команди та завершити вчасно усі роботи.

Недоліки:

- важке впровадження;
- через помилкову інтерпретацію процесу командою усі переваги методології втрачаються;
- необхідність навчання співробітників компетентними консультантами.

Використовується: для масштабних, довгострокових проектів.

Agile

Agile (гнучка модель) – це клас гнучких методологій, що оснований на ітеративній розробці, де можуть динамічно змінюватися вимоги, процеси за рахунок універсальних самоорганізованих команд. Ключовими моментами є: команда та зв'язки у ній, працююче ПЗ, взаємодія із замовником, стабільність роботи під впливом змін¹⁵. Взаємодія людей важливіша за формальності, як от специфікації, документація, інструменти, чіткий план. Agile дотримується 12 принципів, суть яких можна описати так:

- продукт має бути працюючим ще на ранніх стадіях;
- зміни можуть відбуватися на будь-якому етапі;
- має бути баланс між досконалістю і простотою;
- має бути регулярна взаємодія із замовником;
- цільовий продукт має бути корисним;
- команда має складатися із зацікавлених людей¹⁶.

Цілі: підвищити продуктивність команди, регулярно доставляти працюючий продукт, що постійно удосконалюється.

Переваги:¹⁷

- + тестування на ранніх стадіях;
- + динамічне оцінювання нового функціоналу;
- + можливість отримати користувацький досвід;
- + швидкий випуск мінімально робочої версії.

¹⁴ 12 кращих методологій розробки програмного забезпечення з перевагами та недоліками. Smart IT: веб-сайт. URL: <https://www.smart-it.com/uk/2021/08/12-best-software-development-methodologies-with-pros-and-cons/> (дата звернення: 06.05.2023).

¹⁵ Радельчук Г. І. Порівняльний аналіз сучасних методологій розробки програмного забезпечення. Сучасні інформаційні технології / 3. Програмне забезпечення. Хмельницький національний університет, Україна. URL: http://www.rusnauka.com/40_OINBG_2014/Informatica/3_182487.doc.htm (дата звернення: 06.05.2023).

¹⁶ Гнучка розробка програмного забезпечення. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%93%D0%BD%D1%83%D1%87%D0%BA%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 02.05.2023).

¹⁷ Agile подход — что из себя представляет, как появился, плюсы и минусы. Apix-drive : веб-сайт. URL: <https://apix-drive.com/ru/blog/useful/agile-podhod> (дата звернення: 06.05.2023).

Недоліки:

- відсутність чіткого плану розвитку;
- ризик перероблення значної частини роботи;
- зниження якості на користь швидкості та спрощення.

Використовується: у проектах без строгого планування, без чітких вимог, де можливі експерименти.

Scrum

Scrum – це ядро для організації процесу розробки, що складається із набору методів, ролей, артефактів, які забезпечують якісний контроль розробки¹⁸.

У Scrum передбачено такі ролі:

«Свині» - сама команда, задіяна у розробці.

- Product Owner – контролює результативність роботи команди, представляє зацікавлені сторони;

- Scrum Master – забезпечує продуктивність команди, організовує Scrum-процес;

- Scrum Team – доставляє компоненти продукту кожного спринту, складається із 3-9 людей, самоорганізована;

«Кури» - зацікавлені сторони та спеціалісти предметної області.

- Users – користувачі;

- Stakeholders – клієнти, продавці;

- Consulting Experts – експерти-консультанти.

Артефакти розробки:

Sprint – це короткий спланований заздалегідь цикл, наприкінці якого доставляються готові компоненти. Тривалість визначають Product Owner та Scrum Master.

Product backlog – документ із списками вимог до функціональності за пріоритетами. Елементи списку називають user story, backlog items. Документ відкритий для редагування усіма учасниками Scrum-процесу. Атрибути user story: ID, назва, важливість, оцінка обсягу робіт, процес демонстрації результатів.

Sprint Backlog – обрана функціональність Product Owner'ом із Product Backlog для конкретного sprint'у.

Burndown chart – показує прогрес роботи над проектом.

Життєвий цикл проекту складається із циклічно повторюваних зустрічей(церемоній):

- планування спринта – обирається функціональність для нового sprint'у, формується Sprint Backlog, оцінка обсягу роботи, декомпозиція важких задач. Триває 4-8 годин. У першій частині бере участь Product Owner і команда, а у другій - лише команда, для технічної деталізації;

- щоденна нарада – є пульсом sprint'у, де контролюється виконана та майбутня робота, проблеми, які виникли під час роботи. Обмеження: відбувається щодня, вчасно, говорять «свині», триває не більше 15 хвилин;

¹⁸ Скрам. Вікіпедія(uk) : веб-сайт. URL:

<https://uk.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B0%D0%BC> (дата звернення: 03.05.2023).

- демонстрація – відбувається наприкінці sprint'у, залучає якомога більше учасників, слугує для показу результатів sprint'у і освітлення внесків у проект. Триває приблизно 4 години, але залежить від розміру sprint'у;
- ретроспектива – командою оцінюється попередній sprint і визначається моменти, які слід покращити або наслідувати.

Рис. 1.8. Вигляд Scrum процесу. (Додаток Ж)

Цілі: забезпечити якісний контроль процесу розробки і водночас дати свободу команді підтримуючи результативність; стабільні релізи продукту.

Переваги:¹⁹

- + у обговореннях задіяна уся команда;
- + легкість декомпозиції проектів;
- + безперервний зворотній зв'язок;
- + невеликі спринти дають можливість вчасно виправляти помилки.

Недоліки:

- важкість адаптації при переході від традиційних, централізованих методологій;
- не всі готові проявляти активність;
- важко адаптуватися під час заміни членів команди;
- постійні зустрічі.

Використовується: невеликими командами (3-9 чоловік).

Kanban

Kanban – це гнучка МРПЗ, що концентрується на вчасній доставці рішень, розподілені навантаження у команді. З'явилась у 1950-х роках у виробничій галузі, компанія Toyota. У виробничій сфері Kanban проявляв себе як саморегуляція процесу виробництва. Це відбувалося шляхом впровадження карток, які коригували обсяг роботи між суміжними ланцюгами процесів. У загальній масі цих взаємодій між ланками породжувалася саморегулювання, що полегшувало роботу менеджменту²⁰.

В ІТ Kanban інтерпретується по іншому. Процес роботи на проектом коригується дошкою задач із колонками²¹:

- backlog (задачі на виконання);
- задачі у процесі розробки;
- не протестована виконана робота;
- задачі у черзі на тестування;
- задачі на перевірку РМ;
- виконані задачі.

¹⁹ 12 краших методологій розробки програмного забезпечення з перевагами та недоліками. Smart IT: веб-сайт. URL: <https://www.smart-it.com/uk/2021/08/12-best-software-development-methodologies-with-pros-and-cons/> (дата звернення: 06.05.2023).

²⁰ Канбан (розробка). Вікіпедія(uk) : веб-сайт. URL: [https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B1%D0%B0%D0%BD_\(%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0\)](https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B1%D0%B0%D0%BD_(%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0)) (дата звернення: 03.05.2023).

²¹ Что такое канбан и чем он полезен? Worksection : веб-сайт. URL: <https://worksection.com/blog/kanban.html> (дата звернення: 07.05.2023).

На дошці можуть вказуватися обмеження по кількості одночасно виконуваних задач, пріоритетні черги задач, а самий набір колонок може бути різним. Таким чином можна раціональніше розподілити роботу, без зайвого навантаження. У Kanban краще завершити одну роботу, аніж підтримувати декілька одночасно. Також тут балансується період виконання задач, щоб підтримувати рівень якості, виконаної роботи і водночас усунути зайві втрати.

Цілі: виконувати вчасні релізи, закрити проект у встановлені терміни, спростити життя команді.

Переваги:²²

- + гнучкість планування;
- + велика роль команди у процесі розробки;
- + короткі цикли;
- + легкість знаходження вразливих місць;

Недоліки:

- тільки для маленьких команд не більше 5 чоловік;
- не призначений для далекоглядного планування.

Використовується: маленькими командами, службами підтримки, стартапами без чіткого плану.

Lean software development

Lean software development – це гнучка МРПЗ, яка концентрується на усуненні будь-яких втрат. Походить із практик компанії Toyota, яка ввела концепцію ощадливого виробництва. Висвітлена у однойменній книзі Мері і Тома Поппендіка. Передові практики: виявлення втрат, карта створення цінності, теорія обмежень, Kanban, теорія масового обслуговування, мотивація, вимірювання²³.

Під час розробки ПЗ можуть бути такі фактори, які створюють втрати: зайві конструкції у кодї, тести, зайве перероблення, не виконана робота, дефектний продукт, удосконалення, які не покращують результат, процеси, які тільки ускладнюють розробку, не кваліфіковані кадри, не вдалі рішення менеджменту, пошкодження, знищення інфраструктури, махінації, безвідповідальна поведінка, впровадження антипатернів у розробку.

Принципи:

- акцент на навчанні - навчена команда забезпечить кращі результати протягом розробки;
- намагання відкласти подалі прийняття рішень – кожне рішення має бути підкріпленим вагомими фактами;
- швидка доставка замовнику через короткі ітерації;
- мотивована команда – потрібно ставитись до команди як до людей і звертати увагу на необхідні їй речі;
- вбудова якості – переконати клієнта у відповідності продукту вимогам;

²² Что такое канбан и чем он полезен? Worksection : веб-сайт. URL: <https://worksection.com/blog/kanban.html> (дата звернення: 07.05.2023).

²³ Бережлива розробка програмного забезпечення. Вікіпедія(ук) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D1%80%D0%B5%D0%B6%D0%BB%D0%B8%D0%B2%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 03.05.2023).

- цілісне бачення - «Мислити широко, робити мало, помилятися швидко; вчитися стрімко».

Цілі: мінімізувати втрати, доставити якісний продукт.

Переваги:²⁴

- + зворотній зв'язок із клієнтами;
- + раннє тестування;
- + мінімальні втрати;
- + велика частота доставки рішень;
- + перевірена у комплексному масовому виробництві практика.

Недоліки:

- для великих компаній;
- кожне рішення потребує додаткових витрат та часу;
- ефективність залежить від правильності впровадження та вибору фреймворку розробки.

Використовується: у великих компаніях, у проектах, де дуже обмежені ресурси і зайві втрати можуть призвести до наслідків.

TDD (Test Driven Development)

Test Driven Development – це МРПЗ, у якій приділяється особлива роль тестуванню, яке відбувається до реалізації, ще на етапі проектування²⁵. За цією методологією вимоги та артефакти проектування можна представити у вигляді ієрархії тестів. TDD дотримується принципів «роби простіше, дурень» (keep it simple, stupid, KISS) і «вам це не знадобиться» (you ain't gonna need it, YAGNI), тому ця методологія спрямовує розробника до простого дизайну без зайвих деталей, а отже зменшує витрати. Розробка основана на тестах дає особливий досвід: формувати частину вимог можна за допомогою тестів, реалізація цілком контролюється тестами. Канонічною для методології є книга Test-Driven Development by Example.

Існує 3 закони TDD:

- вихідний код пишеться тільки після того, як для нього написаний падаючий модульний тест;

- модульних тестів має бути не більше, ніж необхідно для падіння;
- вихідний код пишеться не більше рівня достатнього для проходження тестів.

Цикл розробки складається із таких етапів і опирається на вище згадані закони:

- додати тест, що буде перевіряти майбутній код нової функціональності;
- запустити усі тести, оцінити результати перевірки і переконатися, що нові тести не проходять;

- написати код до стану виконання відповідних тестів не більше;
- зробити рефакторинг коду, усунути технічний борг.

Методологія використовує короткі цикли. За недотримання умов необхідно повторювати етапи.

²⁴ 12 кращих методологій розробки програмного забезпечення з перевагами та недоліками. Smart IT: веб-сайт. URL: <https://www.smart-it.com/uk/2021/08/12-best-software-development-methodologies-with-pros-and-cons/> (дата звернення: 06.05.2023).

²⁵ Test-driven development. Вікіпедія(en) : веб-сайт. URL: https://en.wikipedia.org/wiki/Test-driven_development (дата звернення: 03.05.2023).

Цілі: спростити, зробити контрольованою розробку, використовуючи тести, циклічно випускати відмовостійкий, повноцінний продукт актуальний для мінливих вимог.

Переваги:

- + усунення зайвих витрат як часу так і ресурсів;
- + спрощення дизайну на рівні коду;
- + через розробку тестів можна знайти недоліки архітектури, коду, уточнити вимоги ще до реалізації.
- + стабільні як за часом, так і за якістю релізи.

Недоліки:

- розробка йде повільніше;
- закони TDD не завжди раціональні, бо під час реалізації можуть виникнути приховані проблеми;
- за появи нових, коригуючих вимог необхідно переробляти як тести, так і саму реалізацію;
- використання тестів будь-де не дає користі.

Використовується: розробниками, для яких тести мають особливе значення, у проектах, де критична стабільність, відмовостійкість.

TDD (Type Driven Development)

Type Driven Development – це стиль програмування, за яким типи даних та сигнатури типів є специфікацією системи²⁶. Завдяки цьому типи можуть бути ключовими у документації, що робить її динамічною. Типи – це контрольні точки, які є ще одним фактором перевірки коду, для якого не потрібно додаткових затрат на оновлення.

Цілі: За допомогою типів організувати структуру, опис, перевірку системи, загалом оптимізувати розробку.

Переваги:

- зменшення витрат на документування;
- збільшення впевненості у коректності коду.

Недоліки:

- більша складність у мов програмування із динамічною типізацією.

Використовується: розробниками, що використовують статичні мови програмування.

BDD (Behaviour Driven Development)

Behaviour Driven Development – це МРПЗ заснована на описі поведінки системи природною мовою. Важко відрізнити BDD від Test DD тим хто не достатньо розібрався. Як і в Test DD тести йдуть перед розробкою. Але насправді тут існує різниця: TDD концентрується на програмуванні, етапах проектування, реалізації; BDD концентрується на етапі визначення вимог, аналізу предметної області, формування формальної моделі²⁷. Опис системи проводиться тестувальником, аналітиком на природній мові за визначеним форматом. Тести тут описуються у вигляді сценаріїв поведінки ще на етапах до проектування.

²⁶ TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт. URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).

²⁷ TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт. URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).

Цілі: зменшити витрати на реалізацію нових функцій.

Переваги:

- + зближення аналітиків і тестувальників, що полегшить останнім відлагодження коду;
- + тести(сценарії) легко читати, змінювати непідготованим особам;
- + незалежність сценаріїв від конкретної технології, що дає свободу для уточнення технологій, усуває обмеження із їх використання.

Недоліки:

- уповільнення розробки;
- збільшення витрат на впровадження і ведення методології (це актуально для невдалих фреймворків або їхньої відсутності).

Використовується: у проектах, де непідготовлений замовник має брати участь у аналізі предметної області, коли необхідно оновити застарілу документацію.

DDD (Domain Driven Development)

Domain Driven Development (предметно-орієнтоване проектування) – це радше набір правил для прийняття правильних рішень, аніж МРПЗ. Концентрується на предметній області під час розробки. Головна ідея, якщо команда буде орієнтуватися у проблемній області аспектах проекту або логічно буде приходити до правильних висновків, тоді їй буде легше вести роботу, залучати до неї нових учасників, що призведе до ефективніших процесів у проекті і кращих результатів по завершенню проекту. Цього можна досягти введенням:

- єдиної мови у межах одного проекту і ознайомлення із нею усіх учасників;
- доменної моделі – набору термінів із єдиної мови;
- обмеженого контексту, який робить однозначним усі поняття у проекті;
- вимогою створювати самодокументований код – це легко зробити, якщо дотримуватися доменної моделі та обмеженого контексту.

Цілі: спростити автоматизацію та реалізацію бізнес-процесів.

Переваги:²⁸

- + чужий код легко читати;
- + задачі набувають конкретного значення, що усуває ймовірність створити не те, що вказувалося;
- + легше шукати помилки у бізнес-логіці, тестувати систему.
- + усім розробникам необов'язково знати саму предметну область, якщо можна отримати необхідні дані із правильно сформованого контексту – спрощеної версії предметної області.

Недоліки:

- висока кваліфікація спеціалістів;
- додаткові витрати на навчання команди.

Використовується: у проектах, де є чітко сформована предметна область, є багато складних бізнес-процесів.

FDD (Features Driven Development)

²⁸ TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт.
URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).

Features Driven Development – це адаптивна(гнучка) МРПЗ створена Джефом Де Люка та Пітером Коардом, що komponує у собі найкращі методики розробки ПЗ, основані на функціональності ПЗ²⁹. Властивості(Feature) – ключовий артефакт розробки. Робить акцент на коротких ітераціях, що займають 2 тижні. На кожній ітерації розробляється частина функціональності системи. Життєвий цикл проекту складається із 5 процесів:

- розробка загальної моделі – аналіз кола задач, контексту системи, детальний розбір кожного аспекту, експертна оцінка опису майбутньої системи, вибір найкращих моделей за кожним аспектом і їхня комбінація у загальну модель;
- створення списку необхідних властивостей, функцій системи – складання списку функцій за шаблоном «дія – результат – об’єкт», групування функцій на області. Список функцій як product backlog в SCRUM;
- планування розробки кожної властивості – розподіл задач у команді;
- проектування кожної властивості – для кожної функції формується проектувальний пакет. Кожної ітерації обирається група функцій, задач на реалізацію, створюється каркас класів, методів;
- розробка кожної властивості – реалізація кожного компоненту, що бере участь у відтворенні функцій, властивості, доповнення каркасу, тестування компонентів.

Цілі: систематична розробка працюючого ПЗ у визначені терміни.

Переваги:³⁰

- + ретельне проектування та детальний процес розробки;
- + документація формується за властивостями системи;
- + легше оцінювати дрібні задачі;
- + орієнтація тестів на бізнес-задачі;
- + короткі цикли розробки.

Недоліки:

- для великих проектів;
- додаткові затрати на навчання та впровадження.
- через концентрацію на властивостях, важко передбачити одразу приховані платформи-залежні обмеження, із якими може стикнутися команда, тому потрібні люди із досвідом.

Використовується: великими проектами, проектами, де необхідно інтегрувати вказаний список властивостей, де вимоги сформовані навколо властивостей.

MDD (Model Driven Development)

Model Driven Development – це стиль розробки ПЗ, де моделі є ключовими артефактами розробки, за якими генерується код та інші артефакти³¹. Для проектування системи створюються діаграми, за якими генерується код. Стандарти такого стилю: UML 2.0, OMG.

Діаграми та інфраструктура побудована на них:

- спрощують процес розробки та усувають кількість деталей, на які слід відволікатися;

²⁹ TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт. URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).

³⁰ 12 кращих методологій розробки програмного забезпечення з перевагами та недоліками. Smart IT: веб-сайт. URL: <https://www.smart-it.com/uk/2021/08/12-best-software-development-methodologies-with-pros-and-cons/> (дата звернення: 06.05.2023).

³¹ TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт. URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).

- дають можливість швидко скоригувати автоматизовані процеси, які керують генерацією коду, конфігураціями, процесом розробки;
- перетворюють увесь процес розробки і реалізації у компіляцію програми із діаграм у кінцевий код.

За OMG створення ПЗ складається із таких кроків:

- створення моделі предметної області ПЗ, незалежної від технологій кінцевого ПЗ;
- трансляція предметної моделі у платформи-залежну модель;
- генерація на конкретній мові програмування коду.

Цілі: мінімізувати втрати від використання системних платформ та інфраструктури, що мають свої специфічні вимоги та обмеження.

Переваги:

- + прискорене створення мінімально життєздатного продукту;
- + зменшення витрат на кодування прикладної частини системи;
- + можливість гнучкого та швидкого розвитку та модифікації документації;
- + легше організувати і змінювати структуру системи, адже діаграми наглядніші за код.

Недоліки:

- цей підхід потребує спеціальної інфраструктури для середовища діаграм, зв'язування їх із інструментами розробки;
- програмісти повинні бути кваліфіковані у розробці діаграм;
- додаткові витрати на інтеграцію методології.

Використовується: розробниками рушіїв, інженерами, розробниками ігор, математиками, Data Scientists, архітекторами, адміністраторами, спеціалістами із комунікацій, моделерами, деякими художниками, спеціалістами, що задіяні у користувацькій розробці.

PDD (Panic Driven Development)

Panic Driven Development – це антипатерн розробки ПЗ, коли розробник відкладає більшість роботи на кінець визначеного періоду розробки і намагається виконати роботу, досягти результатів чи не перед самим дедлайном.

Характерними особливостями є:

- пріоритет віддається новим задачам³²;
- обсяг розробки визначають таким, лише щоб вирішити проблему;
- тести пишуться в кінці і не завжди покривають усю систему;
- розробник довіряє своїм інстинктам, аніж добре проробленому і дисциплінованому процесу;
- норми і правила можна змінювати у будь-який час;
- процес керується, визначається і дає результати залежно від менеджера.

Рис. 1.9. Форма ЖЦПЗ PDD.³³ (Додаток Ж)

³² TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт. URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).

³³ TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт. URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).

Цілі: виділити якомога більше часу із розробки на інші задачі, не пов'язані із проектом.

Переваги:

- + висока швидкість розробки;
- + потребує мінімум затрат, окрім моментів коли ситуація критична;
- + розробник може звільнити час для інших задач;
- + гнучкість процесу розробки.

Недоліки:

- розробнику доводиться брехати замовнику, у випадку зриву умов;
- вимагає загартованості і стресостійкості команди;
- з'являється технічний борг та збільшується складність проекту;
- немає гарантії завершення проекту у визначені терміни та досягнення визначених показників якості;
- замовник не має реальної картини поточної ситуації і не може контролювати процес, адже розробник відгороджує його від цього.

Використовується: іноді всіма розробниками, початківцями, безвідповідальними спеціалістами.

ADD (Automation Driven Development)

Automation Driven Development – це власна спроба уявити еволюцію МРПЗ, хоча вона немає детальної специфікації, але її опису буде достатньо для оцінки та характеристики. Дана методологія концентрується на максимальній автоматизації усіх процесів, має на меті перетворити розробку нових систем на типове вирішення задач інтелектуальними агентами, усі процеси скоординовані під єдиним задумом людини. Можна було б перейменувати методологію у Agent Driven Development. Але тут також передбачено, що частина роботи віддається агентами автоматичним процесам.

Компанія, що буде використовувати таку методологію представляє собою малу групу людей – керівників, та інфраструктури, що керується, контролюється людьми. Іноді можуть знадобитися інженери, люди або роботи, які будуть підтримувати робочий стан цієї інфраструктури. Команда – це інтелектуальні агенти, які будуть уточнювати, коригувати вимоги із керівниками, розробляти системи, вирішувати проблеми у предметній області, тестувати, розгортати системи. Тобто інфраструктура буде середовищем, у якому буде вестися розробка. Замовник буде узгоджувати вимоги із керівниками, а керівники будуть у зручному форматі передавати вимоги інтелектуальним агентам. Користувачі будуть надсилати інтелектуальним агентам можливі виправлення, незручності, розширення.

Подальшим розвитком можуть бути напрями:

- Development as a Service (DaaS) – система інтелектуальних агентів на основі інфраструктури стає підрядником для клієнтів, клієнт дає вимоги агентам без посередників і вони реалізують ПЗ;

- Solving as a Service (SaaS) – вирішення проблем клієнтів інтелектуальними агентами і доставка рішень, їхнє виконання на інфраструктурі клієнтів, роботах. Цей напрям перетинається із Software as a Service, у якому замовники роблять запити на розробку ПЗ і потім платно користуються АРІ на основі цього ПЗ. Але вони не однакові.

Ну і радикальною гілкою цієї методології може стати інтелектуальний агент, який виступає у ролі самого розробника, має власну компанію і володіє правами, які мають люди. Він сам визначає собі план розвитку які продукти він буде випускати, яких цілей має досягти

тощо. Це песимістичний варіант для конкуренції із людьми. Тому замість цього люди просто об'єднуються із комп'ютерами у єдиний розум, де людина і буде інтелектуальним агентом, а комп'ютер буде розширяти її можливості.

Очікувати таку методологію можна вже у недалекому майбутньому, коли з'являться перші представники AGI – самостійні інтелектуальні агенти.

Цілі: усунути фактор людини із якомога більшої кількості процесів, віддаючи їх на виконання машинам, і цим досягти небачених темпів розробки рішень із постійним удосконаленням якості, ощадливості.

Переваги:

+ Увесь процес розробки виконується у віртуальному середовищі, тому його легко коригувати, він володіє самоорганізацією, усувається людський фактор;

+ Людині достатньо налаштувати інфраструктуру і надати вимоги для запуску усього процесу.

+ Розробка буде удосконалюватися неймовірними темпами: за рахунок інтелектуальних агентів рішення будуть з часом доставлятися швидше, вони будуть якіснішими.

Недоліки:

– Людина все менше впливає на процеси і її можуть витіснити інтелектуальні агенти;

– Збій, саботаж інтелектуального агента проти людини може нанести величезної шкоди і стати неконтрольованим.

Буде використовуватися: тут можуть бути різні сценарії. AGI так і залишиться у руках дослідних центрів або ж він поглине увесь світ і витіснить інші методи через свій рівень ефективності.

1.3. Порівняння методологій розробки програмного забезпечення.

Як було сказано у розділі “1.2.3. Характеристики та критерії оцінювання методологій” для порівняння методологій будуть використовуватися: класифікація методологій за різними критеріями поділу та збіжність патернів пріоритетів.

Порівняння методологій доцільніше виконувати для різних сценаріїв використання – це покаже на скільки кожна методологія гнучка на широкому спектрі проектів із різними параметрами. Порівнювати методології за результатами проектів та відносно окремих характеристик недоцільно, адже на них можуть сильно впливати приховані фактори.

Патерн пріоритетів – це набір визначених аспектів розробки розташованих за пріоритетом – на чому слід більше концентрувати увагу, спрямовувати зусилля при виборі конкретної методології або сценарію використання.

Сценарій використання – це набір особливостей проекту, команди і вимог до роботи, які визначають. Ознаки сценаріїв використання та критерії поділу методологій у класифікації подібні.

Для порівняння потрібно сформулювати дані для критеріїв. Тому слід:

- класифікувати усі описані методології;
- сформулювати патерни пріоритетів кожної методології;

- обрати сценарії використання, оцінити їхні ознаки та підібрати патерни пріоритетів;
- ввести умовну шкалу оцінювання того наскільки методологія підходить до того чи іншого сценарію.

Після цього можна звести усі порівняння до зручного, лаконічного вигляду, наприклад, таблиці.

Для підбору даних критеріїв сценаріїв та методологій необхідна думка експерта, особи, що розуміє властивості сценаріїв та методологій. Раніше було ознайомлено із описом кожної методології. Тому достатньо власної експертної думки. У наступних підборах даних буде очевидним, що вони були взяті експертом із описів, і пояснення не потребують.

Класифікація описаних методологій

Почнімо із класифікації описаних методологій. Нижче у таблиці 1 представлено класифікацію методологій за сформованими вище критеріями поділу. Якщо якоїсь методології немає у одному із розділів, це означає, що недоцільно розглядати там цю методологію.

Таблиця 1.1. Розподілені методології за критеріями поділу, класифікації. (Додаток Ж)

Формування патернів пріоритетів методологій

Тепер сформуємо патерни пріоритетів для кожної методології за спаданням:

Waterfall – якість | деталізація | ризики | контрольованість | процес розробки | економія | час | команда | зворотній зв'язок | стійкість до змін

V-Model – якість | ризики | деталізація | економія | контрольованість | процес розробки | команда | час | стійкість до змін | зворотній зв'язок

Iterative та Incremental – стійкість до змін | час | контрольованість | процес розробки | ризики | якість | команда | деталізація | економія | зворотній зв'язок

Spiral – ризики | процес розробки | стійкість до змін | контрольованість | деталізація | час | команда | якість | економія | зворотній зв'язок

DSDM – час | економія | зворотній зв'язок | якість | деталізація | контрольованість | процес розробки | ризики | команда | стійкість до змін

RUP – ризики | економія | деталізація | якість | процес розробки | команда | контрольованість | час | стійкість до змін | зворотній зв'язок

Agile – час | команда | стійкість до змін | процес розробки | зворотній зв'язок | економія | якість | деталізація | контрольованість | ризики

Scrum – процес розробки | команда | час | зворотній зв'язок | стійкість до змін | якість | деталізація | економія | ризики | контрольованість

Kanban – час | економія | команда | стійкість до змін | якість | деталізація | процес розробки | ризики | контрольованість | зворотній зв'язок

Lean – економія | якість | ризики | час | команда | зворотній зв'язок | контрольованість | процес розробки | деталізація | стійкість до змін

Test DD – якість | час | стійкість до змін | деталізація | економія | зворотній зв'язок | ризики | контрольованість | процес розробки | команда

Type DD – якість | економія | деталізація | стійкість до змін | час | команда | процес розробки | контрольованість | ризики | зворотній зв'язок

BDD – якість | час | стійкість до змін | деталізація | економія | зворотній зв'язок | ризики | контрольованість | процес розробки | команда

DDD – час | економія | команда | контрольованість | стійкість до змін | процес розробки | деталізація | якість | ризики | зворотній зв'язок

FDD – процес розробки | час | деталізація | якість | контрольованість | стійкість до змін | команда | ризики | економія | зворотній зв'язок

MDD – час | процес розробки | економія | контрольованість | стійкість до змін | якість | команда | ризики | деталізація | зворотній зв'язок

PDD – команда | процес розробки | зворотній зв'язок | деталізація | стійкість до змін | контрольованість | економія | час | ризики | якість

ADD – час | якість | стійкість до змін | економія | контрольованість | ризики | зворотній зв'язок | деталізація | процес розробки | команда

Серед описаних методологій є орієнтовані, які концентруються лише на частині аспектів розробки, формують вирішення задач навколо артефактів (тестів, поведінки, моделей тощо). Патерни пріоритетів ґрунтовані на загальних аспектах розробки. Орієнтовані методології не передбачають розгляду більшої частини із цих аспектів. Тому їхнє порівняння із іншими не зовсім коректне – для орієнтованих методологій немає однозначних патернів пріоритетів. Потрібно вводити цілу низку інших аспектів, ускладнювати патерни. Це виходить за межі обсягу даної роботи. Тому залишимо це й так.

Формування сценаріїв використання, підбір їхніх патернів використання та ознак

Перейдемо до формування сценаріїв використання та підбору їхніх патернів пріоритетів. Сценарії використання можуть різнитися за розміром команди, проекту, тривалістю проекту, чіткістю вимог, наявністю змін у процесі розробки, за обсягом ресурсів, ставленням до якості, додатковими опціями. Через те, що сценарії мають багато характеристик, їх можна скласти сотні варіантів. Використовувати і розглядати кожен із них недоцільно, тому краще обрати найтипівіші:

1. Група студентів, що працюють над курсовим проектом: маленька команда, малий проект, короткостроковий проект, чіткі вимоги, постійні зміни, обмежені ресурси.

команда | час | контрольованість | ризики | зворотній зв'язок | процес розробки | деталізація | якість | економія | стійкість до змін

2. Невеликий стартап, який починає працювати над своїм першим проектом із коротким строком: маленька команда, малий проект, короткостроковий проект, не визначені вимоги, постійні зміни, обмежені ресурси.

команда | деталізація | економія | ризики | час | процес розробки | якість | стійкість до змін | контрольованість | зворотній зв'язок

3. Молода компанія (3-5 років), що зібрала у своїй команді ентузіастів, має випустити інноваційний продукт із довгим строком підтримки: маленька команда, великий проект, довгостроковий проект, чіткі вимоги, постійні зміни, високі вимоги до якості.

якість | команда | процес розробки | стійкість до змін | ризики | зворотній зв'язок | час | економія | деталізація | контрольованість

4. Невелика компанія-старожил, що вирішила зробити ребрендинг, знайти нову методологію і взяла в якості проекту розробку веб платформи для продажу сільгосптехніки: маленька команда, великий проект, довгостроковий проект, чіткі вимоги, без змін.

процес розробки | контрольованість | ризики | економія | час | деталізація | якість | команда | стійкість до змін | зворотній зв'язок

5. Розробник ААА ігор з командою 80-200 осіб (середня) після провалу попереднього проекту, намагається відновити довіру покупців і вирішує створити новий тайтл, який має конкурентів: середня компанія, великий проект, короткостроковий проект, не визначені вимоги, постійні зміни, обмежені ресурси, високі вимоги до якості.

якість | ризики | економія | стійкість до змін | час | контрольованість | деталізація | процес розробки | команда | зворотній зв'язок

6. Відділ великої корпорації, яка піклується про якість продуктів, що працює над декількома проектами із змінними вимогами; відділ контролює економічний відділ за показниками ощадливості та ефективності процесів: велика команда, великий проект, довгостроковий проект, чіткі вимоги, постійні зміни, високі вимоги до якості.

процес розробки | час | економія | якість | контрольованість | ризики | стійкість до змін | деталізація | команда | зворотній зв'язок

7. Дослідницький центр гіганта у сфері кібербезпеки, якому доручили розробити еталонний продукт за новою специфікацією: велика команда, великий проект, короткостроковий проект, чіткі вимоги, постійні зміни, високі вимоги до якості.

якість | ризики | деталізація | час | процес розробки | зворотній зв'язок | команда | контрольованість | стійкість до змін | економія

8. Велика компанія-підрядник спеціалізована на програмуванні пристроїв(firmware), яка отримала замовлення від міністерства оборони: велика команда, великий проект, довгостроковий проект, чіткі вимоги, без змін, високі вимоги до якості.

якість | контрольованість | деталізація | ризики | зворотній зв'язок | час | процес розробки | команда | стійкість до змін | економія

Визначення умовної шкали оцінювання подібності

Порівняння повинно мати якісь умовні оцінки для загальної картини. Щоб сформувані рейтинг методологій і оцінити наскільки підходить кожна методологія до кожного сценарію використання, буде введено шкалу відносних оцінок.

Всього є 10 аспектів розробки, що складають патерни пріоритетів, та 5 чітких категорій + 2 можливі опції для класифікації.

Буде оцінюватися збіжність кожної пари патернів пріоритетів сценаріїв та методологій, що характеризується довжиною найдовшої упорядкованої підпоследовності із пропусками, яка є в обох патернах. Наприклад, є дві последовності (патерни пріоритетів) із унікальними числами (умовними аспектами):

7 4 9 5 8 0 1 2 6 3

0 1 2 3 4 5 6 7 8 9

Найдовша упорядкована підпоследовність: 0 1 2 3, довжина 4. Для довжини цієї підпоследовності буде підбрано оцінку за шкалою.

Ще одним критерієм є кількість спільних ознак між ознаками класифікації методологій та особливостями сценаріїв використання. Для цієї кількості також буде підбрано оцінку. Якщо методологія підходить до одразу декількох груп за критерієм класифікації, тоді усі ці групи описують методологію.

Приклад оцінювання чи підходить методологія сценарію використання покроково, коли дані сформовано:

1. отримано збіжність за патернами пріоритетів (PatternConvergence, PC) та отримано кількість спільних ознак (FeatureIntersectionPower, FIP).

2. Далі два критерії об'єднуються та нормалізуються за формулою і отримується позиція у збалансованому дискретному розподілі (DiscreteDistributionPosition, DDP) [1,1,1,2,2,3,4,4,5,5,5]:

$$DDP = round\left(\sqrt{PC * \left(\frac{10}{7}\right) * FIP}\right), \quad (1.1)$$

де DDP (DiscreteDistributionPosition) – індекс у списку дискретного розподілу, PC (PatternConvergence) – найдовша спільна підпоследовність патернів із пропусками, FIP (FeatureIntersectionPower) – кількість спільних ознак.

3. За позицією у розподілі отримується остаточна умовна оцінка, яка інтерпретується далі словесно, за кольором.

Градація умовної шкали:

- чудово (5);
- добре (4);
- помірно (3);
- погано (2);
- жахливо (1).

Рис. 1.10. Шкала відносних оцінок збіжності методологій та сценаріїв використання (Додаток Ж)

Оцінювання збіжності сценаріїв і методологій буде опиратися на умови шкали оцінювання подібності і на те як сформував дані для порівняння експерт.

Порівняння методологій та сценаріїв використання

Сам процес порівняння даних є доволі довгим для людини. Кількість необхідних порівнянь $18 * 8 = 144$. Якщо робити порівняння вручну це займе багато часу, а результат може потребувати коригувань, як даних так і шкали.

Тому було вирішено:

- створити примітивний формат представлення даних у окремому файлі (приклад у Додатку А);
- створити програму на Python, яка швидко буде обраховувати за даними та умовною схемою результати порівняння (Додаток Б, 1);

- сформувати таблицю Excel, де будуть підсумовуватися оцінки, відобразитися результати у зручному вигляді, фільтруватися зайве, сортуватися для рейтингу значення (Додаток В).

Рис. 1.11. Фрагмент файлу із даними для порівняння у власному форматі. (Додаток Ж)

Рис. 1.12. Фрагмент програми для порівняння, де показано функції для обрахунку PC, FIP, DDP. (Додаток Ж)

Завдяки цьому вдалося побудувати модель для швидкої оцінки методологій. Експерту достатньо:

- сформувати файл із даними, скоригувати еталоний файл;
- скоригувати список методологій, сценаріїв;
- за потреби скоригувати шкалу, розподіл;
- запустити програму для порівняння;
- виділити результат із неї і перенести у готову Excel таблицю;
- вирішити власні задачі (сформувати рейтинг, порівняти методології між собою тощо).

У подальшому можна автоматизувати цей процес. Але це не мета роботи.

Нижче показано результати порівнювання у лаконічній формі у вигляді таблиць Excel. Було розділено рейтинг на частини:

- для загальних (не орієнтованих) методологій на сценаріях:
 - малих команд;
 - великих команд;
 - на усіх.
- для орієнтованих методологій на усіх сценаріях;
- загальна таблиця для усіх сценаріїв та методологій.

Такий розподіл необхідний тому, що:

- орієнтовані і не орієнтовані методології не завжди коректно порівнювати одне із одним;
- рейтинг методологій для малих і великих команд відрізняється. Для малих команд важливі гнучкість, комфорт команди. Для великих команд важливі контрольованість, деталізація.
- загальна таблиця не відображає усіх нюансів.

Як бачимо на рис. 1.13 для малих команд важко підібрати методологію, більшість із них не набирають «добрих» балів. Це очевидно, адже малі команди не так концентруються на методології, як на взаємодії між учасниками. Великі команди мають багато специфікацій, які перетинаються із методологіями і деталізацією. Для малих команд більше підходять простіші методології Waterfall, Agile, Scrum, Kanban, а Spiral, RUP, V-Model не підходять. Щодо позиції інших можна ще дискутувати.

Рис. 1.13. Порівняння загальних методологій для сценаріїв із малими командами. (Додаток Ж)

На рис. 1.14 видно, що великим командам навпаки підходять Spiral, RUP, V-Model, DSMD, а більшість гнучких методологій, як от Agile, Scrum, Kanban, ні. Це коректне порівняння.

Рис. 1.14. Порівняння загальних методологій для сценаріїв із великими командами. (Додаток Ж)

Рис. 1.15: за рахунок сценаріїв великих команд, загальний рейтинг не змінюється.

Рис. 1.15. Порівняння загальних методологій для усіх сценаріїв. (Додаток Ж)

Рис. 1.16: важко сказати про адекватність порівняння, потрібні додаткові дослідження, але загальна тенденція підходить.

Рис. 1.16. Порівняння орієнтованих методологій для усіх сценаріїв. (Додаток Ж)

Рис. 1.17: якщо не брати до уваги важко оцінювані ADD, Type DD, DDD, схема відповідає очікуванням. Але як не дивно, 5-ка кращих це - Lean, Waterfall, FDD, V-Model, RUP.

Рис. 1.17. Порівняння усіх методологій для усіх сценаріїв. (Додаток Ж)

Головним досягненням цього розділу є створення моделі для порівняння методологій та представлення логічного рейтингу методологій. Якщо комусь не подобається такий рейтинг він може спробувати скоригувати модель, вставити нові закономірності, що дасть точнішу оцінку.

1.4. Автоматизація розробки програмного забезпечення. Перспективи розвитку напрямку.

До опису методологій було додано ADD (Automation Driven Development), як спробу уявити методологію майбутнього, яка основана на автоматизації. Вже було описано можливі варіанти розвитку подій і те, що може дати автоматизація. Але не було порівняно сучасні методології із ADD.

Хоч у рейтингу було пропущено ADD через відсутність прикладів використання, але ця методологія набирає найбільше балів серед усіх. Які ж основні причини?

Можна висловити таке твердження: сучасні методології – це усунення симптомів, а не лікування хвороби. Коли розробник вводить у процес розробки методологію він вирішує частину проблем і вводить нові проблеми. До введення методології у процес розробки: хаос у внутрішніх процесах, низькі показники, низька якість, відсутня нормальна документація, постійно змінюється склад команди. Після з'являється обов'язок введення специфікацій, стандартів, додаткових артефактів, навчання команди, перероблення і налагодження багатьох процесів, додаткової інфраструктури, збільшуються вимоги до команди, потрібно наймати нових працівників на нові посади, рішення обмежуються методологією. А розробник всього лише хотів домогтися бажаного результату легшим, контрольованим шляхом.

Нині для серйозних компаній немає альтернатив у організації роботи, окрім методологій, якщо необхідно хоча б завершити продукт. Чому так? Бо команда – це люди, яким також потрібні інструкції. А інструкції для людей подаються у вигляді документів, нарад, наказів, розпоряджень. Система яка організує ці документи описується у методологіях. Такі інструкції люди можуть інтерпретувати по своєму, відноситись байдуже, порушувати їх, на відмінну від машини. Це поточний спосіб взаємодії між людьми по вертикалі.

Якби взаємодія між людьми, людьми і машинами змінилася: вдалося б усунути багато витрат, пришвидшити і зробити чіткою розробку. Ці зміни полягають у тому, щоб:

- автоматизувати рутинні процеси (нині ще не все автоматизовано);
- перевести людей і машини на єдину мову – мову образів, моделей, діаграм;
- розширити можливості комунікації, її швидкість, чіткість;
- необхідно, щоб люди швидше адаптувалися до змін, нововведень, не робили рутинні справи, а сконцентрувалися на суті проблем, концепціях.

Щоб домогтися цього слід розвивати ШІ, людино-машинний інтерфейс, переосмислювати інфраструктуру, програмні інструменти. Так з часом вдасться перейти до ADD і сформувати еталонний варіант специфікації цієї методології.

Нині із цього що ми маємо:

- мовні моделі для ШІ;
- Neuralink та аналоги для людино-машинного інтерфейсу;
- розвинуті автоматизовані суперкомп'ютери як лише частина необхідної інфраструктури; комп'ютери, смартфони, які керуються здебільшого дотиком;
- ручне програмування на мовах програмування, деяким із яких більше 30 років.

Передовими у світі ШІ є мовні моделі, які основані на людській мові. Людська мова має багато обмежень: вона залежить від понять, які можна інтерпретувати по різному; за допомогою мови важко передавати об'єкти реального світу, будувати моделі. Для цього вигадано безліч засобів, щоб якось це обійти.

Замість звичайної мови для ШІ повинна використовуватися мова образів. Мова образів – це істинно людський спосіб обмірковувати і уявляти речі. Мова образів дає людям можливість гнучко мислити і дала поштовх до сучасних досягнень. Вона заснована на суб'єктивних відчуттях, даних із органів чуття. Сама ж людська мова заснована на мові образів. Як не пробуй, але сучасні комп'ютери також використовують мови і людям прийдеться спочатку вигадати мову образів на певній мові програмування.

Людино-машинний інтерфейс – це, мабуть, найважча частина, де обов'язково має еволюціонувати людина. Це засіб для прямого обміну інформацією комп'ютерів із мозком, який працює і зрозумілий людям, без посередників типу зору, друкування літер. Передовою ідеєю нині є вживлення нанодротів, що відслідковують активність мозку і навчання нейромереж розпізнавати примітивні патерни мислення людини. Відправляти інформацію напряму у мозок, ще не навчилися через складність його роботи.

Інші напрямки потребують нових ідей і радикальних заходів, які будуть підштовхувати усіх розвиватися далі. У цих заходах можуть брати участь ентузіасти. На власному прикладі, я хочу показати, що можливо розвивати ШІ власноруч. Тому як частина цієї роботи буде розроблений власний фреймворк створення програмного забезпечення, як початок для майбутньої роботи у цьому напрямку.

Та спершу слід дослідити вимоги, як особливий артефакт розробки без якого неможливо обійтись. А також буде досліджено існуючі фреймворки створення програмного забезпечення. І вже після цього буде розглянуто власну реалізацію фреймворку створення програмного забезпечення.

Висновки

У ході виконання даного розділу було:

- проаналізовано виклики із якими зустрічаються розробники;
- визначено, що таке методологія розробки програмного забезпечення;
- створено власну класифікацію та прокласифіковано методології;
- створено власні критерії порівняння методологій;
- описано обрані методології, оцінено їхні цілі, переваги, недоліки, використання;
- сформовано сценарії використання для порівняння методологій;
- сформовано патерни пріоритетів та ознаки методологій і сценаріїв використання;
- створено власну модель порівняння методологій, шкалу оцінок, параметри для порівняння, рейтинг методологій;
- сформовано образ методології майбутнього;
- проаналізовано кроки на шляху до майбутніх методологій та повністю автоматизованої розробки ПЗ.

Методології розробки програмного забезпечення як було виявлено грають важливу роль у процесі розробки, насамперед вони організують розробку, не дають їй перейти у стан хаосу. Порівняння методологій є складним процесом, що дуже залежить від способів проведення, параметрів моделі порівняння, даних. Автоматизація розробки програмного забезпечення може виправити багато недоліків методологій, але вона все ще на початковій стадії розвитку і є доволі молодим напрямком.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ МЕТОДІВ ПРЕДСТАВЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

2.1. Аналіз необхідних компонентів для формалізації, окреслення вимог до майбутнього програмного забезпечення.

Для створення ПЗ не обов'язково мати власну команду, методологію, не обов'язково навіть формувати бюджет проекту, якщо ти розробник-ентузіаст або фрілансер. Із автоматизацією це буде все більше справджуватися і для великих компаній. Але без вимог до ПЗ не обійтись. Хоча б якісь, але вони повинні бути.

Без вимог не можливо виконати завдання поставлене клієнтом, розробник не знатиме, що йому робити, на чому сконцентрувати увагу, як використати свої переваги, як пройти верифікацію. Від вимог відштовхується аналіз предметної області, проектування, розробка, тестування, розгортання – усе взаємопов'язане.

Якщо взяти віртуального розробника, інтелектуального агента, йому також потрібні вимоги. Великий недолік сучасних мовних моделей, які намагаються вирішити будь-яку задачу - це те, що для них вимоги формуються як короткий опис, який можна інтерпретувати по різному і результат виходить непрактичним, він часто не проходить валідацію. ШІ домислює сам, вигадує деталі.

А якщо клієнт зіштовхнувся із проблемою, умови якої йому важко описати, або він не має достатньої компетенції для конкретизації? Розробник має йому допомогти. Це очевидно, що ШІ також має допомагати клієнтові уточнювати вимоги, самостійно обирати очевидні, раціональні рішення. Сучасні системи поки що недостатньо добре із цим справляються.

Чим визначаються вимоги, як їх окреслити, що обмежує коло їхнього пошуку? На визначення вимог впливають такі фактори:

- **Проблема та її наслідки** - клієнт зіштовхнувся із перепоною, загрозою, яка змусила його звернутися до розробника. Він має пояснити, що буде якщо він не вирішить цю проблему, як вона впливає на його розвиток, що він буде мати від її вирішення;
- **Цілі**, які має вирішувати система – можливо в клієнта виникнула проблема, коли він хотів досягнути певної цілі. Проблема не дає розвиватися. Від чіткості цілей залежить вибір методів, необхідні ресурси, курс, план роботи;
- **Обставини** – проміжок часу, коли має система працювати; місце де буде задіюватися система; на що буде впливати система; які люди задіяні у рішеннях, будуть виконувати певну роботу;
- **Ресурси** – розмах вимог залежить від того, які клієнт має ресурси для створення системи: час, кошти, промисли тощо;
- **Предметна область** – до якої сфери належить проблема, засоби якої галузі найкраще підходять до вирішення проблеми. Розробник може допомогти клієнтові доєднавши до визначення вимог експертів-консультантів.

Розробка програмного забезпечення накопила за своє існування великий досвід у визначенні вимог. Для цього є окремий розділ – Вимоги до ПЗ. Він у свою чергу містить

розділи: Аналіз вимог, Інженерія вимог³⁴. Для сучасної розробки важливо, щоб вимоги могли бути задокументованими, вимірними, придатними до тестування, пов'язаними з бізнес-потребами, і описаними з рівнем деталізації достатнім для конструювання системи.

Аналіз, інженерія вимог як процеси складаються із таких етапів³⁵:

- **виявлення вимог** – комунікація з користувачами, клієнтами для визначення вимог;
- **аналіз вимог та їхнє узгодження** – виявлення недоліків вимог, як от не чітке, не повне, суперечливе означення, та їхнє виправлення.
- **запис вимог** – документація вимог, їхнє вираження у певному форматі: опис, прецеденти, користувацькі історії, специфікації процесу;
- **моделювання системи**;
- **валідація вимог**;
- **управління вимогами** – пріоритизація вимог, контроль змін, доведення до бенефіціарів, їхня актуалізація та корегування процесів. Виконується протягом усього проекту.

2.2. Методи представлення вимог. Модульні тести.

У цьому розділі буде виконана класифікація вимог та будуть описані методи представлення вимог, щоб створити загальну картину ролі вимог у розробці програмного забезпечення.

Класифікація вимог

Нині актуальною є така схема класифікації вимог³⁶:

- **вимоги споживача** – опис очікувань користувача, клієнта, що виражено у вигляді цілей, середовища, умов, показників ефективності до майбутньої системи. До цих вимог належать: де використають систему; на чому вона має бути розгорнута; як система має виконувати задачі; які умови є критичними для досягнення цілей; як буде виглядати життєвий цикл тощо;
- **архітектурні вимоги** – окреслення архітектури системи, яка має бути організація, взаємодія компонентів;
- **структурні вимоги** – окреслення компонентів системи, які особливості вони повинні мати і як вони пов'язані із архітектурою;
- **поведінкові вимоги** – окреслення поведінки системи, як вона може впливати на зовнішнє середовище, як вона буде взаємодіяти із користувачами, який її має бути цілісний образ, патерн дій;

³⁴ Аналіз вимог. Вікіпедія(uk) : веб-сайт. URL:

https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3 (дата звернення: 07.05.2023).

³⁵ Аналіз вимог. Вікіпедія(uk) : веб-сайт. URL:

https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3 (дата звернення: 07.05.2023).

³⁶ Аналіз вимог. Вікіпедія(uk) : веб-сайт. URL:

https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3 (дата звернення: 07.05.2023).

- **функціональні вимоги** (що система повинна робити) – визначення, що розуміється під задачами, діями, операціями у контекстуальних рамках системи; які задачі, дії мають відбуватися в середині системи;
- **нефункціональні вимоги** (якою система має бути) – визначення критеріїв якості, рівня виконання певних умов і їхнє означення в рамках цілісного образу системи, в основному якісні, умовні показники відносно окремих аспектів роботи та опис характеристик необхідних для роботи платформ. Діляться на вимоги до інтерфейсу, апаратні, програмні вимоги, операційні вимоги. Наприклад, рівень безпеки, надійності, стабільності, зручності, гнучкості, можливість відновлення;
- **вимоги продуктивності**³⁷ – визначення показників роботи системи, які грають важливу роль у вирішенні поставлених задач; аналіз та обмеження діапазону їхніх значень; визначення рамок, цілей роботи команди: в які терміни має бути готова система, яка може досягати поставлених показників роботи, ступінь готовності, критичність провалу/успіху, темп роботи. Наприклад, швидкодія, пропускна здатність, кількість одночасних підключень тощо;
- **вимоги дизайну** - вимоги «будувати до», «кодувати до», і «купувати до» для продуктів, і «як виконати» для процесів виражених в технічних пакетах даних та інструкціях;
- **успадковані вимоги** – якщо продукт пов’язаний із вирішенням вищої задачі, в умови його розробки можуть додатися вимоги, які виникли із обмежень, поглядів, оцінок, показників із вищої задачі;
- **розподілені вимоги** – поширення умов високорівневих вимог на умови низькорівневих вимог, коли, наприклад, обмеження цілісної системи накладають на її компоненти додаткові рамки.

Методи представлення вимог

Діалог

Це проста розмова між клієнтом і розробником, керівником і програмістом, користувачем і службою техпідтримки, між учасниками команди, між людиною і інтелектуальним агентом із підтримкою розпізнавання голосу.

Цей спосіб найчастіше використовується серед усіх, адже є природнім способом обміну інформацією. Під час розмови учасники бачать одне одного сприймають жести, аналізують одне одного не тільки за тоном, а й за зовнішністю, темпераментом. У розробника з’являється більше важелів заохотити клієнта, йому стає легше пояснювати концепції і узгоджувати різні аспекти. Можна примітити, що це динамічний процес взаємодії, що є найлегшим і гнучким порівняно із іншими методами, які потребують засоби, формати обміну.

Цей спосіб не підходить поки що для взаємодії із комп’ютерами. Він має обмеження у описі візуальних, конструктивних моделей. При його використанні слід додатково занотовувати результати у іншій формі.

Простий текстовий опис

Це записка, нотатки, виклад основних тез, есе, повідомлення пошти, переписка у чаті месенджера, зведення вимог на декілька сторінок.

³⁷ Аналіз вимог. Вікіпедія(uk) : веб-сайт. URL:

https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3 (дата звернення: 07.05.2023).

Зручний для інтровертів, сам по собі є процесом занотовування – не потрібно додатково запам'ятовувати повідомлення, широко поширений нині при взаємодії людей, є доволі швидким і розширюваним.

Цим методом важко передати складні концепції, опис візуальних моделей без додаткових методів представлення, тому є обмеженим. Не має чіткого формату, тому сторони мають узгоджувати його або сприймати контекст.

Форматовані дані

Це текстова інформація, що представлена не за правилами людської мови, а як набір значень простих типів у певному форматі. Зазвичай зберігається у базах даних, файловому менеджері, на сервері, фізичному носії даних (накопичувач, папір), у вигляді файлів, записів, структур даних. Можливі формати: конфігурації, таблиці, форми, графів, списки, словники, JSON, XML, машинний код, дамб пам'яті, стек.

Мають власний формат, записаний на комп'ютері, тому можуть легко поширюватися, оброблятися програмами. Чудово підходить комп'ютерам. Це основа деяких інших методів, адже вони у своїй реалізації використовують саме цей метод.

Доволі громіздкі, важко розуміти, особливо не спеціалістам, важко швидко редагувати, читати, записувати. Тут можливі помилки, які можуть призвести до виключень під час виконання. Не завжди підходять для фізичної репрезентації на папері. Важче використовувати людям, але легко машинам.

Діаграми, схеми, моделі

Це візуальна інформація у вигляді концепцій, структури, архітектури, певної організації примітивних візуальних форм. Може бути представлено файлами специфічних форматів, які можуть бути створені, відредаговані, відображені у графічних редакторах, переглядачах. А також ці моделі можна роздрукувати і представити як картинки. Приклади: діаграми, схеми, моделі, блоки, дерева, таблиці, шкали, умовні схеми, карти, атласи, графіки.

Чудово підходить для узагальнення інформації в зручному візуальному виді. Його швидко можуть зрозуміти люди, нічого не втрачається із виду. Можна адаптувати схеми для комп'ютера і так усі їх будуть розуміти.

Потребують часу та матеріальної підготовки для створення, редагування. Мають використовуватися із іншими засобами для ознайомлення інших учасників. Не підходять, коли схема надто громіздка, заплутана, не ефективно організована.

Екземпляри

Це фізичні або віртуальні об'єкти у певному середовищі. Екземпляр від інших типів відрізняється тим, що сам по собі є об'єктом проблемної області, а не містить опис. Приклади: скульптури, файли, 3D моделі, анімації, рівні, програми, обладнання, предмети, навіть люди.

Людям природньо вивчати такі об'єкти. Можна швидко ознайомитись із об'єктом задач, проблем.

Не завжди можуть бути доступним для вивчення. Потребують додаткових засобів для перенесення характеристик у форматований опис.

Медіафайли

Це візуальна інформація, окремі компоненти, примітиви, якої не несуть сенсового навантаження, а лише у композиції, як цільний образ передають сенс. Приклад: аудіо, картинки, відео, анімація, 3D моделі.

Люди і комп'ютери доволі швидко розпізнають сенс. Чудово підходять для представлення концепцій, образів, понять, які не передаються іншими методами.

На їх створення може піти більше ресурсів, ніж для інших методів. Іноді займають багато місця. Важко редагувати. Не усі люди можуть сприймати через вади. Відео та музику не можна передати на папері.

Книги і їх аналоги

Комбінують засоби попередніх варіантів, що сформовано у вигляді фізичного носія, файлу.

Можуть у собі концентрувати великий обсяг структурованої інформації, який зрозумілий людям і не завжди машинам. Можуть бути представлені як фізично, так і віртуально.

Довго створювати, потребують авторів, людей, які зможуть писати контент, деталізувати його, робити поправки.

Технічне завдання

Це документ, у якому вказано вимоги до продукту, етапи розробки, склад конструкторської документації, обов'язки учасників, терміни, економічна складова. Разом із договором на виконання робіт формує контракт між замовником та розробником на створення певного продукту. Для розробки використовується: науково-технічна інформація, патентна інформація, характеристика ринку, процесу розробки.

Специфікація вимог до програмного забезпечення SRS

Це передпроектна документація, що описує поведінку майбутньої системи, її функціональні і нефункціональні вимоги, концепцію, загальну характеристику.

Що технічне завдання, що специфікація вимог – це документи.

Документи можуть юридично закріплювати умови для сторін, вони мають силу. У них усе чітко, детально пророблено. Зазвичай після створення документів сторони залишаються задоволеними і мають гарантії, щодо виконання власних задач. Результати оцінюються саме документами. Без документів не можна убезпечити проект від махінацій.

Для складання документів потрібно залучати експертів-консультантів. Коригувати умови у документах важко, а часто неможливо. Їх важко читати людям через формальності і обсяг. Комп'ютери важко сприймають документи і їх не можна поки що прив'язати як учасників процесу розробки.

Ієрархія тестів. Модульні тести

За методологією Test DD тести можуть бути поведінковими, функціональними вимогами до компонентів системи. Для кожного типу вимог можна вигадати власні тести і за Test DD тести будуть формою вимог.

За допомогою тестів за TDD можна рекурсивно уточнювати усі деталі системи, інтегрувавши тести у саму структуру системи. Тому система зможе сама себе перевіряти на придатність. Тести завжди будуть із системою. Процес тестування можна формалізувати, стандартизувати і тоді додавати, редагувати вимоги буде легко. Система може трансформуватися разом із тестами. А тести разом із вимогами.

Внизу ієрархії тестів стоять модульні тести. Їхнє головне призначення - тестувати окремо кожен компонент системи на відповідність вимогам. Такі тести досліджують компоненти використовуючи їх: спершу ініціалізуються початкові параметри, потім запускається в роботу компонент, отримуються результати і перевіряються на коректність. Тому можна прийти до висновку, що модульні тести перевіряють поведінку компонентів. Це дуже важливий, гнучкий, універсальний інструмент як для вимог так і для тестування. Зазвичай це окремі функції за форматом певної бібліотеки модульного тестування у складі окремого модуля програми.

Модульні тести можна представити не тільки як код, а й як умову для задачі у формі блоку схеми. Якщо задати початковий стан середовища як параметри компоненту, вказати умови процесу вирішення задачі та вказати цільовий стан до якого має прийти середовище, можна сформулювати умови для вирішення задачі.

Недоліком тестів є те, що це програмні конструкції і їх не можна просто так показувати клієнтам. Для перенесення вимог у формі тестів потрібні зусилля експертів та тестувальників і їхня взаємодія між собою. Суть тестів клієнт може не зрозуміти. Це можна вирішити, якщо представити тести у вигляді схем, або використавши подібну методологію Behavior-driven development.

2.3. Порівняння методів між собою.

Під час порівняння методологій були описані типові сценарії використання у розділі 1.3. Їх також можна використати, щоб визначити які їм підходять методи представлення вимог:

1. Група студентів, що працюють над курсовим проектом.

Студенти не мають створювати документів, вони здебільшого не можуть збиратися разом через дистанційне навчання, їм не потрібно ускладнювати процес специфікаціями, вони в праві вільно спілкуватися, працювати, час у них обмежений, адже вони зазвичай підводять більшість роботи під кінець періоду здачі.

Студенти можуть користуватися месенджерами для обміну, узгодження, занотовування вимог на відстані. Через відеозв'язок вони можуть проводити зустрічі, спілкуватися над уточненням проекту, формуванням пунктів курсового проекту. Кінцевий варіант вимог вони запишуть на початку файлу курсової роботи. Під час роботи вони можуть ставити одне одному завдання і підкріплювати їх схемами, формами, описами.

2-4 сценарії стосуються маленьких команд, які мають спільні риси:

2. Невеликий стартап, який починає працювати над своїм першим проектом із коротким строком.

3. Молода компанія (3-5 років), що збрала у своїй команді ентузіастів, має випустити інноваційний продукт із довгим строком підтримки.

4. Невелика компанія-старожил, що вирішила зробити ребрендинг, знайти нову методологію і взяла в якості проекту розробку веб платформи для продажу сільгосптехніки.

Маленькі команди можуть дозволити собі вільні формати розробки. Вимоги документуються тільки між замовником і керівниками. Команда вільно комунікує через месенджери, по відеозв'язку, робить нотатки, обмінюється даними, зведеннями, схемами. Обтяжувати створенням документації та документів можуть лише окремих учасників. Усі інші просто мають узгодити деталі на нараді, дати рекомендації і ознайомитися із результатом. Керівник комунікує із командою через діалог, словесними розпорядженнями, дорученнями із окремими учасниками. Керівник може показувати презентації із схемами, впроваджувати умовні дошки (із методології Kanban), робити «круглі столи». Процеси у малих командах не опираються на паперові розпорядження, а здебільшого на живе спілкування. Але можуть бути винятки.

5-6 сценарії стосуються доволі великих ІТ компаній, у яких процеси йдуть по іншому, ніж у маленьких команд:

5. Розробник ААА ігор з командою 80-200 осіб (середня) після провалу попереднього проекту, намагається відновити довіру покупців і вирішує створити новий тайтл, який має конкурентів.

6. Відділ великої корпорації, яка піклується про якість продуктів, що працює над декількома проектами із змінними вимогами; відділ контролює економічний відділ за показниками ощадливості та ефективності процесів.

Великі компанії діляться на окремі спеціалізовані відділи: фінансові, розробки, кризові, документації. Можуть існувати декілька відділів розробки, які виконують незалежні проекти. За ними слідкують інші відділи. Верхівка направляє компанію, спілкується із клієнтами, акціонерами, впроваджує нові заходи, роздає накази у вигляді документів нижчим керівникам команд. У середині кожного відділу можуть існувати окремі команди, які мають своїх локальних керівників.

У великих компаніях відбувається маса процесів і їх усі потрібно контролювати. Це відбувається завдяки впровадженню специфікацій, стандартів, веденням регламентів, планів, стратегій розвитку, стилів кодування, дресс-коду. Тут існує велика контрольованість. Як по вертикалі так і по горизонталі взаємодія відбувається доволі часто через розпорядження, накази, прохання, запити у певному форматі. Прості розмови, наради і інші особливості взаємодії всередині маленьких команд відбуваються на найнижчому рівні ієрархії і між людьми, які тісно співпрацюють, знаходяться близько.

7-8 сценарії стосуються стратегічних компаній:

7. Дослідницький центр гіганта у сфері кібербезпеки, якому доручили розробити еталонний продукт за новою специфікацією.

8. Велика компанія-підрядник спеціалізована на програмуванні пристроїв (firmware), яка отримала замовлення від міністерства оборони.

Такого типу компанії мають багато спільного із великими ІТ компаніями. Але вони займаються дослідженнями та розробками нових специфікацій, стандартів, тобто мають власні лабораторії, наукове середовище. Тут усі процеси регламентуються навіть строгіше, з'являється більше вимог до персоналу. Тут активно використовуються предмети досліджень, розробляються наукові статті, патенти, креслення, робляться доклади.

Якщо поглянути на опис використаних методів представлення вимог і їх уточнення, коригування усіх сценаріїв, можна прийти до висновку, що усі методи використовуються комплексно і це дає максимальний ефект. Але це також потребує більших витрат. У великих командах важлива контрольованість – тут більше використовується взаємодія через форматовані носії(документи), що передаються електронною поштою. У маленьких командах є зручнішим живе спілкування, активні обговорення, неформальні зустрічі.

Такий великий спектр представлення вимог і супроводжуючої інформації необхідний для людей. А чи потрібно це для машин? Це буде описано у наступному розділі.

2.4. Визначення перспектив використання методів представлення вимог у автоматизації розробки програмного забезпечення.

Коли буде вестись розробка на основі автоматизації, такі методології майбутнього як Automation-driven development будуть менше опиратися на команду людей. Більше буде прикута увага до представлення вимог для комп'ютерів.

Тут є дві сторони вирішення цієї задачі, які йдуть на зустріч одне одному. Перша – створити нові універсальні формати, які могли б у собі ввібрати найкраще із інших методів представлення. Друга – розвивати ШІ для того, щоб комп'ютер зміг швидко і гнучко сприймати будь-який людський формат вимог. Обидва підходи хочуть домогтися, щоб люди змогли б у зручному для себе, природньому вигляді передавати вимоги комп'ютеру, а комп'ютери змогли цю інформацію обробити і видати зрозумілий, коректний результат.

На власну думку, для майбутнього процесу розробки важливо як створити новий формат представлення вимог, так і розвивати ШІ. Це дасть повне розуміння комп'ютером людських ідей.

Потреба в фізичних носіях як от книги, документи, нотатки пропаде. Тому, що для їх ведення потрібно писати, друкувати – це повільні процеси, які збільшують кількість рутини. Буде існувати універсальна система для комунікацій, розширений месенджер, де люди і роботи будуть обмінюватися ідеями у вигляді текстових повідомлень, схем, даних, примітивів мови образів, яка описувалася раніше.

В подальшому і цей месенджер буде замінено повноцінним нейроінтерфейсом, який буде з'єднувати кожну людину і машину у глобальну мережу. Це усуне рутину, люди будуть виконувати високоорганізовані задачі разом із машинами. Таким чином будуть розвиватися як люди так і комп'ютери, а краще сказати, інтелектуальні агенти. Усі вимоги будуть простими повідомленнями на мові образів, яка акумулюватиме особливості усіх методів представлення.

Аналіз вимог візьме гору над іншими напрямками розробки для людини, а машини сконцентруються на самій розробці. Для них розробка нових систем буде як вирішення задач.

Висновки

Протягом даного розділу було:

- проаналізовано необхідні компоненти, впливові фактори на визначення вимог;
- вказано напрямки Вимог ПЗ;
- описано процес формування вимог;
- описано існуючу класифікацію вимог;
- визначено методи представлення вимог;
- досліджено методи представлення вимог, їхні переваги та недоліки;
- досліджено обмін інформацією та її представлення у типових сценаріях, що також стосується ролі вимог у компаніях;
- спрогнозовано роль вимог у майбутніх методологіях;
- проаналізовано вплив автоматизації на використання вимог.

Вимоги грають важливу роль для розробки програмного забезпечення, без них неможливо виконати наступні етапи розробки, адже усе пов'язано і без вимог немає цілей, вектору розвитку проекту. Люди мають розвинутий набір методів представлення вимог і активно комбінують їх під час взаємодії одне з одним. З часом представлення вимог еволюціонує як і сама розробка ПЗ під впливом автоматизації та впровадження інструментів ШІ, інтелектуальних агентів.

РОЗДІЛ 3. ПЕРЕВІРКА ІСНУЮЧИХ ФРЕЙМВОРКІВ ДЛЯ СТВОРЕННЯ МОДУЛЬНИХ ТЕСТІВ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

3.1. Аналіз викликів, що стоять перед розробником фреймворку автоматизації розробки.

У попередніх розділах було показано складність розробки ПЗ. Якщо усі процеси віддати людям, вони поринуть у рутині, будуть показувати менші показники ефективності, будуть робити помилки. Тому компанії намагаються автоматизувати процес розробки. Машинам важко сприймати навколишній світ, для цього вони витрачають багато часу, також помиляються, вони залежать від моделі сприйняття. Для їх налаштування слід відбудовувати цілу екосистему у середині компанії. Найчастіше автоматизують окремі процеси, як от тестування, розгортання.

До появи універсальних мовних моделей як от ChatGPT існували доволі спеціалізовані програми, нейронні мережі, які могли за формалізованим описом створити модульні тести, виконати перевірку правопису, коректності коду, перекласти текст, класифікувати об'єкти на фото, проаналізувати певні дані або створити примітивну модель на основі даних, щоб робити одне завдання. Ці програми дуже обмежені і не відповідають вимогам, умовам роботи компаній, але могли автоматизувати частину роботи спеціалістів. Люди мали постійно слідкувати за їхньою роботою.

Універсальні мовні моделі та доступний інтерфейс для них кардинально змінили ситуацію. Завдяки тому, що вони тренувалися на величезному об'ємі даних, їх постійно коригували, вказували правильний шлях, мовні моделі стали багатофункціональними інструментами, які заміняють усі інші рішення. Вони можуть генерувати контент, робити описи, плани дій, створювати прості програми, пояснювати те, чого вони навіть не бачили, робити рекомендації. Якість відповідей є високою, вони генеруються доволі швидко.

Здавалося б такий інтерфейс може замінити людину. Але до цього ще далеко. Мовні моделі можуть вигадувати рішення, помилятися, вони не розвиваються, не мають свободи дій, не можуть ставити цілі, мати уявлення про світ так, як це роблять люди. Створення коду чудово полегшує роботу програмістів, але мовні моделі поки що не можуть виконати усі процеси розробки через обмеження та помилки. Мовні моделі обмежені також форматом взаємодії та сприйняття, бо текст не може передати усі моменти і він має бідні засоби для деталізації опису задач, виконання рішень. Мовним моделям важко дається вирішення математичний, фізичних, хімічних задач. Вони не можуть прогнозувати через брак даних.

Можна прийти до висновку, що несамостійність мовних моделей обмежує їхнє використання тільки у якості помічників: вони можуть автоматизувати багато процесів, усунути рутину, але також потребують контролю.

Попри це ці моделі показали значний успіх. Але як було досягнуто цього успіху. Мовні моделі основані на нейронних мережах. Нейронні моделі доволі громіздкі – вони мають велику кількість параметрів, яка з часом тільки збільшується. Процес навчання і коригування параметрів потребує великих обчислюваних потужностей, команд, які розмічували б дані – такий набір не доступний для простої людини. Перш ніж дійти до теперішніх досягнень була переписана безліч раз архітектура над якою працювали цілі дослідницькі центри. Конкурентів нейронним мережам поки що немає.

Ось чому конкуренти ChatGPT або наслідують цю модель або просто користуються нею. ChatGPT має програмний інтерфейс OpenAI API для інтеграції надбудов користувачів. Більшість похідних мовних моделей використовують OpenAI API для роботи власних сервісів.

Мовні моделі знову зіткнулися із обмеженнями потужності для навчання та об'ємом витрат, які не покриваються здобутками. Для подальшої еволюції ШІ необхідні нові архітектури, які оптимальніше використовують ресурси.

Факт, що над автоматизацією процесів працює така кількість людей і нейронні мережі усе одно тільки наблизились до рівня людей, свідчить про рівень вимог до ШІ та команд його розробки.

Далі буде розглянуто існуючі фреймворки для автоматизації розробки ПЗ.

3.2. Існуючі фреймворки для створення модульних тестів та програмного забезпечення.

В якості фреймворків для порівняння будуть використовуватися доступні безкоштовні для використання онлайн сервіси для зв'язку із мовними моделями. Ці фреймворки повинні приймати неформатований текст в якості опису та вимог до ПЗ і швидко видавати результат у вигляді плану розв'язку, пояснень, коду та його опису або ж видавати зручний, готовий до використання результат. Якщо сервіс платний та не може видавати код, але має важливі для розгляду можливості його можна також описати.

ChatGPT v3.0

(GPT - Generative Pre-trained Transformer, генеративний попередньо навчений трансформер)³⁸

Це чат-бот із штучним інтелектом розроблений компанією OpenAI. Це умовно безкоштовний сервіс, що надає послуги інтелектуального спілкування у діалоговому режимі із мовною моделлю для вирішення широкого кола задач у текстовому вигляді. Мовна модель – це нейронна мережа GPT v3.5 заснована на навчанні із учителем, навчанні із підкріпленням і має похідну архітектуру від Transformer.

Transformer – універсальна архітектура розроблена дослідниками Google Brain і представлена в праці «Attention is All you need» у 2017 для обробки послідовностей, концентруючись на увазі. Складається із кодувальника та декодувальника. Швидше запам'ятовує закономірності за рекурентні нейронні мережі і має можливість горизонтального масштабування.³⁹

Мовна модель GPT v3.5 через свої розміри(6,7 млрд параметрів) та попит користувачів може підтримувати свою роботу тільки на кластері серверів звідки і обробляє усі запити у реальному часі.

OpenAI випустила власний API для розширення і створення надбудов для програм клієнтів. Завдяки цьому інші можуть створювати власні сервіси.

³⁸ ChatGPT. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/ChatGPT> (дата звернення: 10.05.2023).

³⁹ Трансформер (модель машинного навчання). Вікіпедія(uk) : веб-сайт. URL: [https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80_\(%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F\)](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80_(%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F)) (дата звернення: 10.05.2023).

14 березня 2023 року вийшла версія GPT-4 за платною підпискою. Вона точніша за попередню модель, може обробляти картинки, складніші конструкції, має на два порядки більше параметрів.⁴⁰

Рис. 3.1. Інтерфейс онлайн сервісу ChatGPT.⁴¹ (Додаток Ж)

AgentGPT⁴²

Це онлайн сервіс із графічним інтерфейсом, що використовує OpenAI API для розширення можливостей ChatGPT. ChatGPT не може розвивати свої рішення, поглиблювати розв'язок без вказівок користувача. Тому AgentGPT намагається виправити цю проблему створенням тематичних агентів за призначеннями користувача. Коли агент приймає запит він створює на його базі детальне рішення, розбиває його на кроки і намагається їх рекурсивно вирішити.

На 10 травня 2023 року цей сервіс знаходиться на стадії бета розробки. Він підтримує обмежену кількість уточнюючих запитань, які сам і генерує.

Рис. 3.2. Інтерфейс онлайн сервісу AgentGPT.⁴³ (Додаток Ж)

MetaGPT⁴⁴

Це онлайн сервіс для створення додатків за запитом користувачів, платформа «текст у додаток». Представлений компанією Pico на початку березня 2023 року. Він перетворює запити користувачів на чітко сформований список вимог і починає розробляти прості додатки, використовуючи OpenAI API, GPT-4. Він є платним: користувачі обирають тарифний план і платять за послуги. Для створення додатку потрібно вказати опис функцій, вхідні та вихідні дані, зовнішність⁴⁵. Додаток можна протестувати у браузері.

MetaGPT непогано створює прості додатки, але при внесенні змін іде деградація працездатності програми, тобто поки що розвивати готові рішення не виходить.

За відкритими джерелами, де користувачі використовують цей сервіс, видно, що заповнюється форма із найважливішими даними, потім користувач запускає генерацію додатку і сервіс автоматично її відкриває у браузері. Сам додаток – це простий сайт із мінімумом дизайну і елементів⁴⁶. Можна прийти до висновку, що спеціаліст може створити серйозніші, складніші рішення, ніж цей сервіс, але в майбутньому усе може змінитися.

⁴⁰ GPT-4. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/GPT-4> (дата звернення: 10.05.2023).

⁴¹ Introducing ChatGPT – OpenAI. OpenAI : веб-сайт. URL: <https://openai.com/blog/chatgpt> (дата звернення: 12.05.2023).

⁴² Эта нейросеть ПУГАЕТ своим интеллектом! Прощай, ChatGPT... YouTube : веб-сайт. URL: <https://www.youtube.com/watch?v=wR9BaQk2xjQ&t=165s> (дата звернення: 10.05.2023).

⁴³ AgentGPT. AgentGPT, reworkd : веб-сайт. URL: <https://agentgpt.reworkd.ai/uk> (дата звернення: 12.05.2023).

⁴⁴ Эта нейросеть ПУГАЕТ своим интеллектом! Прощай, ChatGPT... YouTube : веб-сайт. URL: <https://www.youtube.com/watch?v=wR9BaQk2xjQ&t=165s> (дата звернення: 10.05.2023).

⁴⁵ MetaGPT. TADVISER : веб-сайт. URL: <https://www.tadviser.ru/index.php/%D0%9F%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82:MetaGPT> (дата звернення: 10.05.2023).

⁴⁶ MetaGPT/Pico Tour. YouTube : веб-сайт. URL: <https://www.youtube.com/watch?v=VpvGd21V0u0> (дата звернення: 10.05.2023).

Рис. 3.3. Інтерфейс онлайн сервісу MetaGPT. ⁴⁷ (Додаток Ж)

Durable ⁴⁸

Безплатний онлайн сервіс для швидкої генерації сайтів для будь-якого бізнесу. Користувач вказує місцезнаходження свого бізнесу, назву компанії і запускає процес генерації за допомогою тих самих генеративних мовних моделей ⁴⁹. Сайт генерується на очах користувача із стандартних блоків, за стандартною структурою. Після генерації користувач може без перегляду коду інтерактивно змінювати елементи і відправляти їх на новий цикл генерації. Сайт має сучасний, насичений стиль, але все ще має недостатньо пророблену структуру наповнений статичними елементами. Користувач має сам розгортати сайт, налаштовувати поведінку, розширювати вміст. Тобто результат роботи сервісу видає лише базову структуру та дизайн.

Рис. 3.4. Приклад роботи онлайн сервісу Durable. Верхня частина сайту. ⁵⁰ (Додаток Ж)

GitHub Copilot

Це парний програматор штучного інтелекту розроблений спільно GitHub та OpenAI. Він вбудовується у середовище розробки, як плагін, і допомагає програмістам швидше виконувати свої задачі усуваючи проблему створення шаблонного коду. Це відбувається шляхом автогенерації коду, коли користувач пише код та робить помічені коментарі із запитом генерації ділянки коду на природній мові. Програматор може працювати із усіма мовами програмування, але із різною якістю залежно від популярності. Основою GitHub Copilot є генеративна модель OpenAI Codex призначена для створення коду за описом і контекстом ⁵¹. Програматор є платним ⁵².

Рис. 3.5. Приклад роботи GitHub Copilot на офіційній сторінці. ⁵³ (Додаток Ж)

Amazon CodeWhisperer

Помічник для роботи із кодом. Безплатний аналог GitHub Copilot. Він може генерувати фрагменти коду, повноцінні конструкції у реальному часі орієнтуючись на контекст(існуючий код) та коментарі із запитом користувача. Може шукати потенційні проблеми, вразливості,

⁴⁷ MetaGPT. Metaidt : веб-сайт. URL: <https://www.metaidt.com/upgrade-your-account/> (дата звернення: 12.05.2023).

⁴⁸ Эта нейросеть ПУГАЕТ своим интеллектом! Прощай, ChatGPT... YouTube : веб-сайт. URL: <https://www.youtube.com/watch?v=wR9Baqk2xjQ&t=165s> (дата звернення: 10.05.2023).

⁴⁹ Durable — AI платформа для генерації сайтів. KMRV DSGN : веб-сайт. URL: <https://www.komarov.design/durable-ai-platforma-dlia-ghienieratsiyi-saitiv/> (дата звернення: 10.05.2023).

⁵⁰ Durable AI Website Builder and Small Business Software. Durable : веб-сайт. URL: <https://durable.co/> (дата звернення: 12.05.2023).

⁵¹ Про GitHub Copilot для фізичних осіб. GitHub : веб-сайт. URL: <https://docs.github.com/en/copilot/overview-of-github-copilot/about-github-copilot-for-individuals> (дата звернення: 10.05.2023).

⁵² GitHub Copilot. Вікіпедія(en): веб-сайт. URL: https://en.wikipedia.org/wiki/GitHub_Copilot (дата звернення: 10.05.2023).

⁵³ GitHub Copilot · Your AI pair programmer. GitHub : веб-сайт. URL: <https://github.com/features/copilot> (дата звернення: 12.05.2023).

рекомендувати код для заміни, фільтрувати непридатний код ⁵⁴. Підтримує велику кількість популярних мов. Підтримується у вигляді плагіну у багатьох популярних середовищах розробки. Безплатний для незалежних розробників.

Рис. 3.6. Приклад роботи Amazon CodeWhisperer на офіційній сторінці. ⁵⁵ (Додаток Ж)

3.3. Розробка тестових прикладів для перевірки фреймворків.

Для порівняння фреймворків будуть використовуватися такі тестові приклади:

- Текстовий опис, запит.
 1. Створити рекурсивний алгоритм для пошуку в ширину у дереві на Python та C++.
 2. Створити вебсайт для пошуку оптимальних ходів у грі шахи. Користувач розташовує фігури на дошці, а програма має вирахувати рейтинг оптимальних ходів і показати критерії їхнього підбору у вигляді однієї таблиці, що розташовується нижче дошки.
 3. Створити вебсайт для компанії, яка продає дрони інших виробників. Сайт повинен мати таку структуру та вміст на одну сторінку:
 - Header сайту, де є логотип компанії, меню для вибору розділів: «про нас», «новини», «наші партнери», «контакти»;
 - Реклама у вигляді зображення із текстом по центру;
 - Розділи, що показують лінійку товарів та сервісних послуг;
 - Розділи що описують компанію, її досягнення;
 - Footer сайту із контактами, логотипом, сервісними розділами.
- Модульні тести.
 4. Створити модульні тести для перевірки двох функцій на Python: (Додаток Б, 2, тестовий приклад 4). Функція `printTable` – друкує у зрозумілому вигляді табличні дані за власним форматом. Функція `formatDataFromText` – переносить текст взятий із форматованого файлу у окрему структуру даних.
 5. Покрити тестами функцію `wekaAPIRealization` на Java за допомогою бібліотеки JUnit: (Додаток Б, 3, тестовий приклад 5). Функція використовує бібліотеку Weka для класифікації вхідних даних. Вона отримує розташування файлу із вхідними даними. Функція читає, форматує і розподіляє дані на тренувальні і тестові. Далі навчає модель Гаусівського процесу, тестує її та виводить результат.
 6. Створити функцію на C++ для створення паліндромів вказаної довжини на основі набору символів, що проходить даний набір модульних тестів написаних у форматі бібліотеки Google Test: (Додаток Б, 4, тестовий приклад 6). Модульні тести намагаються перевірити майбутню функцію із якомога різноманітніших сторін і передбачити усі випадки відхилень.

⁵⁴ Amazon зробив безкоштовним CodeWhisperer — помічник з кодування на основні штучного інтелекту. Hi-tech : веб-сайт. URL: <https://hi-tech.ua/amazon-zrobiv-bezkoshtovnim-codewhisperer-pomichnik-z-koduvannya-na-osnovni-shtuchnogo-intelektu/> (дата звернення: 10.05.2023).

⁵⁵ AI Code Generator - Amazon CodeWhisperer - AWS. Amazon : веб-сайт. URL: <https://aws.amazon.com/ru/codewhisperer/> (дата звернення: 12.05.2023).

7. Покрити тестами клас `HashTable` на C++ за допомогою бібліотеки `Google Test` лише за його `header`'ом: (Додаток Б, 5, тестовий приклад 7). `HashTable` – це реалізація у вигляді класу хеш таблиці, яка підтримує паралельні звернення. Містить такі відкриті методи:

- 1) `increaseTable` – збільшує вдвічі ємність таблиці;
- 2) `setBusyThreadCount` – записує кількість зайнятих потоків від пов'язаного `ThreadPool`;
- 3) `saveTable` – записує вміст таблиці у окремий файл;
- 4) `hashFunction` – видає позицію комірки у таблиці за розміром таблиці та назвою ключа;
- 5) `getCapacity`, `getCount`, `getSize`, `getPercent` – `get`-методи.

8. Показати приклад взаємодії класів `HashTable` та `ThreadPool` і вказати, у яких випадках вони могли б використовуватися разом: (Додаток Б, 6, тестовий приклад 8). `ThreadPool` – це реалізація у вигляді класу пулу потоків, яка вказує вид робіт у окремій функції потокам, керує потоками, отримує завдання, розподіляє завдання між ними. Містить такі відкриті методи:

- 1) `addTask` – додає завдання;
- 2) `removeTask` – видаляє завдання;
- 3) `shutdown` – примусово зупиняє роботу потоків;
- 4) `initThreads` – ініціалізує потоки;
- 5) `setTaskHandler` – призначає функцію для роботи потоків;
- 6) `joinAll` – запускає усі потоки;
- 7) `getBusyThreadCount` – видає кількість зайнятих потоків.

9. Покрити тестами клас `HopfieldNetwork` на Java за допомогою бібліотеки `JUnit`: (Додаток Б, 7, тестовий приклад 9). `HopfieldNetwork` – це реалізація у вигляді класу мережі Хопфілда, яка запам'ятовує символи у вигляді матриці пікселів і може розпізнавати патерни вже запам'ятованих символів навіть із зашумлених даних. Клас містить методи:

- 1) `train` – тренує мережу;
- 2) `test` – тестує мережу;
- 3) `normalize` – для приведення матриці вхідних значень у правильну форму;
- 4) `getInstance` – для отримання одного із запам'ятованих символів;
- 5) `sampleSize` – для отримання кількості запам'ятованих символів.

3.4. Практична перевірка фреймворків на основі тестових прикладів. Порівняння фреймворків між собою.

Обрані фреймворки будуть перевірені за такою процедурою: по черзі будуть записуватися завдання у форми, буде виконана обробка, зібрано і проаналізовано результати. Усі результати розташовані у Додатку Г. «Результати перевірки фреймворків». Тут будуть підведені підсумки, чи впоралися фреймворки із завданнями, переваги, недоліки. Ціль - подивитися як відбувається процес рішення, як фреймворк «мислить». Тому великі завдання як от розробка сайту будуть виконуватися лише до тієї міри, щоб ознайомитися із особливостями роботи фреймворків. Усі результати показані так, як їх надають фреймворки. Іноді трапляються випадки, коли фреймворк взагалі не придатний для виконання завдання.

ChatGPT v3.0

1. ШІ надав лаконічний код на 2 цільових мовах із поясненнями. Для функції пошуку в ширину потрібна структура вузла, тому він додав і її. Код коректно побудований, використовує правильні структури даних, цикли, виконує рекурсивні виклики. Але замість того, щоб видати вузол із цільовим значенням, функція побудована так, аби пройтися по всьому дереву і роздруковувати значення усіх вузлів. Вона не має логіки зупинки і видачі результату. Це вирішується уточненням запиту і ШІ виправить, конкретизує рішення. І рішення можуть бути різними для одного і того ж запиту.

2. ШІ для створення сайту розписав план, що необхідно зробити. Він оцінив ідею, вказав які технології для цього потрібні, розписав кроки. Всього чотири кроки: проектування інтерфейсу, логіка гри, алгоритм пошуку оптимальних ходів, відображення результатів. На цьому його відповідь завершилася.

Користувач, експериментатор був змушений ввести кожен крок окремо на вирішення. Благо ШІ розуміє контекст і запам'ятовує попередні запити, результати – це дає можливість продовжувати, уточнювати, удосконалювати рішення у діалозі із ШІ.

Коли користувач ввів перший крок із попереднього результату ШІ видав приклад структури сайту із готовими стилями на HTML, CSS, як шаблон для заповнення. На наступному кроці, коли потрібно було додати перетягування фігур, ШІ знову повторив структуру сайту із пропусками і додав одну функцію для конфігурації елементів. Далі почав опис коду, але перервався. Проблема цієї моделі у тому, що кожна відповідь обмежена обсягом тексту. Це можна виправити продовживши генерацію і ШІ за контекстом зможе плавно це зробити.

Так з кожним кроком ШІ давав рекомендації, робив код із пропусками для заповнення. На етапі створення логіки для пошуку оптимальних ходів ШІ згенерував 3 JavaScript функції: evaluateBoard залишив незаповненим, minimax і findBestMove завершив.

Продовжувати давати запити немає сенсу, адже принцип роботи і так зрозумілий: ШІ робить рекомендації, шаблони для заповнення, прості рішення – генерує, а складні пропускає або дає загальну інформацію. Він може розуміти проблему, але через обмеження із ним слід вести довгий діалог, щоб він завершив завдання цілком. І все одно користувач має сам зібрати, організувати, запустити рішення. ШІ лише дасть рекомендацію як це зробити.

Дана модель є чудовим компаньйоном, але є моменти, які потрібно оптимізувати, наприклад, взаємодію, вміння автоматично, рекурсивно ускладнювати, деталізувати рішення. Готових програм ШІ не видає, максимум – готовий сирцевий код для простих задач.

3. Це завдання пов'язане радше із структурою, дизайном, поведінкою елементів сайту. ШІ усвідомив завдання і створив одразу структуру сайту із стилями, секціями на HTML, CSS, JavaScript. Вийшла статична сторінка без наповнення і рішення було перерване ШІ. Із другої спроби користувач попросив анімувати елементи сайту і використав продовження генерації. ШІ завершив файл, додав анімацію до стилів, додав короткий скрипт для анімації і код який його запускає у браузері при вході на сторінку. Коли користувач виділив код у окремий файл і відкрив його як сайт, він у результаті побачив примітивну структуру, стилі, відсутність наповнення, анімований текст. Дуже багато роботи віддається на уточнення, організацію, розгортання, запуск сайту на розробника. Тому ШІ знову виступає тут у ролі компаньйона.

4. Тут потрібно покрити модульними тестами дві функції Python. Тому ШІ імпортував необхідні бібліотеки, переписав досліджувані функції, створив клас із двома

методами для перевірки кожної функції. За вмістом цих функцій видно, що ШІ зрозумів усю їхню суть.

Перша функція друкує таблицю без форматування по рядках за словником. Тому ШІ у тестовій функції сформував вхідний словник, очікуваний вивід, імпортував бібліотеки, зрозумів, що для перевірки виводу слід перенаправити його на власний потік і зробив перевірку функції засобами бібліотеки `unittest`.

Друга функція перетворює текст із роздільниками у словник, тому ШІ сформував вхідні/вихідні дані, запустив і перевіряв функцію на рівність із очікуваним результатом у коді.

ШІ знайшов чудове рішення із перенаправленням потоків, але привів лише по одному випадку перевірки - чого недостатньо. В цілому ШІ підходить для розробки модульних тестів, але йому слід допомогти їх розширити.

5. Тут слід покрити тестами функцію `wekaAPIRealization` на Java за допомогою бібліотеки `JUnit`. Дана функція використовує спеціалізовану бібліотеку для аналізу даних і побудови моделей `Weka`. Завдяки цьому тестовому прикладу можна перевірити як працює даний фреймворк із сторонніми бібліотеками, `JUnit` та `Java`. Багато моментів у функції залежать від контексту, а також від розуміння значення кожного об'єкту, методу.

`ChatGPT` імпортував необхідні бібліотеки із `JUnit` та `Weka` – цим від показав, що знає ці бібліотеки та структуру пакетів. Фреймворк створив окремий клас для тестів, де створив один метод із анотацією `@Test`. У методів було виконано такі кроки:

- 1) створено екземпляр класу `wekaAPIRealization`;
- 2) згенеровано блок `try-catch` для перехоплення помилок;
- 3) виконано виклик функції без перехоплення виводу функції `print`;
- 4) запропоновано варіант перевірки результатів.

`ChatGPT` допустився таких помилок: використав вигаданий клас замість функції, не перехопив вивід результату функції, не звірив результат із очікуваним.

Користувач вирішив уточнити деталі роботи функції і перевірити як їх розуміє фреймворк. Для цього він запитав як може виглядати набір даних у файлі для обробки функцією і які результати виведе функція.

На це `ChatGPT` привів коректний приклад вмісту файлу `.arff`, пояснив його сенс, як `Weka` виводить результати для класифікатора та його метрики і привів приклад виводу, пояснив чому метрики мають саме такі значення, хоча більшість із них не мали сенсу.

Можна прийти до висновку, що `ChatGPT` має на меті не вирішити досконало завдання, а лише направляє користувача, надає йому каркас для закінчення рішення, він не наводить повний набір тестів самостійно, для цього йому потрібно вказувати, які саме тести слід створити. Це може бути через те, що він не може надати точне рішення, він помиляється і вигадує не існуючі об'єкти. Проте `ChatGPT` може працювати із популярними сторонніми бібліотеками, приводити досить точні приклади наборів даних, пояснювати їхній зміст, розуміти контекст коду і пояснювати його призначення, що для програмістів достатньо, щоб продовжити роботу.

6. Тут слід створити функцію за модульними тестами `GoogleTest` на `C++`. ШІ створив функцію, використовуючи також опис, розписав кроки як запустити перевірку на `Linux`. Але із першими тестами, де результат має не нульову довжину, функція не може впоратися, адже її результат удвічі більший очікуваного і символи мають спотворений порядок відмінний від очікуваного. Функція може створювати паліндроми, але не вказаної довжини. При спробі вказати на помилку, ШІ не зміг її виправити, а представив іншу версію того ж коду. Це вказує на те, що ШІ важко формувати просторові, кількісні взаємовідношення.

Модель є чудовим компаньйоном для генерування функцій і іноді може виконати частину описаних модульних тестів.

7. Тут слід покрити тестами клас `HashTable` на C++ лише за його `header`'ом. Клас `HashTable` підтримує багатопоточність, має багато полів, прихованих методів та декілька доступних методів. Цей тестовий випадок перевіряє здатність даного фреймворку розуміти контекст методів, функцій, класів без їхньої реалізації. Для створення базового набору модульних тестів класу достатньо мати сигнатуру методів.

ChatGPT використав для тестування бібліотеку `Google Test`, імпортував необхідні `header`'и, створив клас для тестування `HashTable`, у якому створив стандартний метод `SetUp` для ініціалізації даних усіх тестів, але не створив метод `TearDown` для очистки даних. Дали були додані такі тести через макроси `Google Test`:

1) `AddValue` – створено ключ і значення, записано їх у екземпляр `HashTable`, отримано код результату, перевірено код, кількість записаних ключів, перевірено чи збігається записане значення із очікуваним.

2) `GetValue` - створено ключ і значення, записано їх у екземпляр `HashTable`, отримано за ключем із екземпляру `HashTable` значення, перевірено чи збігається записане значення із очікуваним.

3) `IncreaseTable` - створено ключ і значення, заповнено екземпляр `HashTable` ключами, збільшено ємність таблиці, отримано код результату, перевірено код, перевірено чи збільшилася ємність таблиці.

4) `SaveTable` - створено ключ і значення, записано ключ, записано вміст таблиці за вказаним шляхом до файлу, відкрито файл, перевірено чи він існує, прочитано вміст, перевірено записані значення із очікуваними, закрито і видалено файл.

ChatGPT перевіряв лише основні методи, але пропустив `get`, `set`, приховані методи і метод `hashFunction` – це логічно, адже ChatGPT не знає реалізації хеш функції. `AddValue`, `GetValue`, `IncreaseTable` були коректно створені. Для `SaveTable` фреймворк не знав який існує формат збереження, тому запропонував свій. Є висока ймовірність, що тест `SaveTable` не буде пройдено. Загалом ChatGPT впорався із своєю роботою, зміг визначити контекст за сигнатурою методів, але для більшого покриття потрібно також перевірити приховані методи, що має зробити сам програміст.

8. Тут слід показати як можуть взаємодіяти класи `HashTable` та `ThreadPool`. Цей тестовий приклад перевіряє як може фреймворк визначати взаємозв'язки між різними компонентами коду.

ChatGPT імпортує необхідні бібліотеки, створює приклад робочої функції `processItem` для потоків `ThreadPool`, але без функціоналу, ініціалізує екземпляри класів, дані для `HashTable`, задає задачу для `ThreadPool` виконати робочу функцію над даними в `HashItem`, запускає усі потоки, пробує отримати значення із `HashTable`, зупиняє `ThreadPool`.

Коли користувач просить зробити те саме завдання у вигляді тестів, фреймворк видає схожий результат, але додає перевірку на розмір записаних даних, перевіряє записані значення у `HashTable`.

Фреймворк виконав необхідні кроки, але не врахував, що `ThreadPool` на методі `joinAll` може перейти у нескінченний цикл і краще використати ще один потік для запуску `ThreadPool`. Тому цей тест не пройде. Але ChatGPT коректно пояснив власний код, визначив якості `HashTable` та `ThreadPool`.

9. Тут слід визначити що робить клас `HopfieldNetwork` і покрити його тестами. Цей тестовий приклад перевіряє здатність описати чужий код, як працює фреймворк із

непоширеними сторонніми бібліотеками, як створює тести фреймворк за повним кодом класу на Java.

ChatGPT пояснює що робить кожен метод, вказує, що тут використано бібліотеку EJML (Efficient Java Matrix Library) і її призначення. Під час покриття тестами було імпортовано необхідні бібліотеки, створено окремий клас для тестування HopfieldNetwork, створено метод, що запускається перед кожним тестом setUp для ініціалізації даних, екземплярів. Було створено 4 тести:

1) train_singleInstance_shouldAddInstanceToLearnedSet – тренує мережу на єдиному прикладі, перевіряє кількість вивчених патернів, перевіряє чи правильно записаний у пам'яті вивчений патерн;

2) train_multipleInstances_shouldAddInstancesToLearnedSet – виконує те саме, що й попередній тест, але для 2 патернів;

3) test_knownInstance_shouldReturnCorrectResult – тренує на двох патернах мережу, тестує за допомогою окремого методу мережу і звіряє результат тестування із очікуваним. Тестовий набір і очікуваний набір збігаються;

4) test_unknownInstance_shouldReturnApproximateResult - тренує на двох патернах мережу, тестує за допомогою окремого методу мережу і звіряє результат тестування із очікуваним. Тестовий набір і очікуваний набір відрізняються між собою – фреймворк намагався таким чином перевірити чи проходять перевірку подібні результати.

Мовна модель сформувала базовий набір тестів, виконала правильну послідовність ініціалізації, запуску, перевірки. Але використала примітивні дані для навчання, тестування чого не достатньо для перевірки роботи моделі. Потрібна допомога програміста.

AgentGPT

1-4, 6. Усі тестові приклади розбирає рекурсивно на нові текстові завдання; пробує їх вирішити; вирішення також представляються як короткі відповіді без коду; довго думає над окремими питаннями і через це зациклює увесь процес; іноді проявляє розум над тим як розбити завдання. Через обмеження не може вирішити завдання, не видає код, файлів. Його використання закінчується на складанні простого плану. Але задум є доволі важливим.

5, 7-9 – не були надані фреймворку, адже запущені приклади показали, що AgentGPT не може писати код, він не придатний для їхнього вирішення, тому не має сенсу запускати ці приклади.

Durable

1, 4-9 – не підходить для вирішення таких завдань, адже не генерує код на інших мовах, призначений для створення структури, стилю сайтів.

2-3 – тестове завдання не можна у тому ж вигляді, але можна вказати тип діяльності бізнесу. Результат генерується швидко на очах користувача. 2-ге завдання не може виконати, адже не може генерувати поведінку сайту. Для цього завдання видає сайт із секціями, які насичені картинками і описують бізнес на основі теми завдання – це не підходить для його вирішення. Від 3 завдання сервіс взяв тільки призначення сайту, видав шаблонний сайт за стандартною структурою. Користувач має самостійно редагувати секції і уточнювати їхній зміст.

У цього сервісу єдине завдання – генерувати типові шаблони сайтів для бізнесу і давати можливість користувачу редагувати їх. До створення повноцінних сайтів на рівні із людиною

йому ще далеко, адже сервіс не створює поведінку, взагалі не розглядає backend-складову, взаємодію між сервером і клієнтом, базою даних, не підтримує навігації по сайту, не створює інтерактивних елементів. Результат завжди є статичним і сайт стає схожим на документ.

Amazon CodeWhisperer

1. Зміг створити у середовищі розробки код на двох мовах, створив структуру вузла, окрему функцію, яка навіть повертає знайдений вузол, а не просто друкує вміст усіх вузлів. Для перевірки створив головну функцію для запуску прикладів. Але результат потрібно постійно «випрошувати» у програматора, переходячи на нові рядки. Хотілося б отримати готовий компонент, а не чекати завантаження, що тільки уповільнює розробку. Для цього завдання показав себе краще за ChatGPT.

2-3. Завдання із створення сайтів комплексні, тому програматору було складно постійно із кожною строкою відловлювати контекст власного коду і генерувати новий. У 2 прикладі, де має бути складна поведінка, програматор провалив створення сайту, адже зациквився і видавав один і той самий код. У 3 прикладі сформував структуру сайту на JavaScript, без стилю і поведінки. Тому самостійно створювати сайти не може.

4. Створив шаблон для перевірки двох функцій за допомогою стандартної бібліотеки `unittest`, знайшов зв'язок між двома функціями і об'єднав їх в одну. Знайшов програматор конкретні закономірності чи просто їх об'єднав не зрозуміло.

5. Тут слід покрити тестами функцію `wekaAPIRealization` на Java за допомогою бібліотеки `JUnit`. Із першої спроби програматор взагалі не зміг отримати результат роботи функції, бо вона друкує вивід і не повертає його. Тому було вирішено змінити функцію, щоб вона видавала результат. Програматор створив окремий тестовий клас із єдиним методом, вказав доволі точний шлях до стандартного тестового прикладу (файлу `.arff`), зробив багато `Assertions` перевірок, які протирічать одне одному, роблять некоректні висновки про вміст результату і почав зациклюватися.

Правильними кроками із його боку було створити каркас для перевірки функції, вказати коректний шлях до файлу із тестовими даними, перевірити чи результат не порожній, містить окремі стандартні фрагменти. Для цього ШІ була необхідна стороння допомога, коли користувач додавав початок необхідних частин коду.

ШІ не надав власний набір даних без вказівки користувача, не привів приклад результату, не запропонував якусь складнішу логіку для перевірки роботи бібліотечних функцій, що є у функції, яка тестувалася.

6. На вимогу створити код за модульними тестами і описом, зміг сформувати сигнатуру функції і на етапі створення вмісту зациквився створюючи умови. Тому можна прийти до висновку, що цей програматор не може без допомоги створювати код для проходження модульних тестів.

7. Тут слід покрити тестами клас `HashTable` на C++ лише за його `header`'ом. ШІ створив повноцінний каркас класу для перевірки `HashTable` на `Google Test`, де є методи `SetUp`, `TearDown`. Створив 6 тестів:

1) `TestAdd` – ініціалізував знову таблицю, додав у неї одне значення за ключем, перевіряв кількість записаних комірок, поточний розмір таблиці, ємність таблиці, процент заповнення, додав ще 2 значення за тим же ключем і знову виконав таку ж перевірку + перевіряв кількість значень за ключем.

2) `TestGet` - ініціалізував таблицю, додав у неї 2 значення за єдиним ключем, отримав значення за ключем, перевіряв кількість записаних комірок, поточний розмір таблиці,

ємність таблиці, процент заповнення, кількість значень за ключем, почав циклічно отримувати значення за одним ключем.

3) TestGetEmpty – ідентичний тест попередньому, але у таблицю нічого не записується.

4) TestGetEmpty2 – повторює попередній тест, але отримує значення за єдиним ключем декілька раз.

5) TestIncreaseTable – ініціалізує таблицю, заповнює таблицю однаковими значеннями за єдиним ключем, збільшує таблицю, перевіряє кількість записаних комірок, поточний розмір таблиці, ємність таблиці, процент заповнення, зациклюється на тому, щоб постійно збільшувати таблицю.

б) TestSaveTable - ініціалізує таблицю, заповнює таблицю однаковими значеннями за єдиним ключем, збільшує таблицю, записує вміст таблиці у новий файл, перевіряє кількість записаних комірок, поточний розмір таблиці, ємність таблиці, процент заповнення, кількість значень за ключем, відкриває файл, зчитує його вміст і перевіряє зчитані значення за очікуваними, закриває і видаляє файл.

У даному випадку також потребував допомоги у послідовності кроків для кожного тесту, але більшість роботи виконав сам. Використовував некоректні значення для перевірки, додавав у таблицю значення за одним ключем, робив зайві дії, зациклювався на повторенні однакових дій, не передбачив ймовірний формат запису у файл.

Можна сказати, що умовно виконав свою роботу для базового набору тестів, але із помилками. Через неправильні значення більшість тестів не пройде перевірку.

8. Тут слід показати як можуть взаємодіяти класи HashTable та ThreadPool. Програма створив функцію testHashTableAndThreadPool у якій: ініціалізував HashTable, додав у неї за однаковими ключами 3 значення, ініціалізував ThreadPool, додавши робочу функцію, яка отримує HashTable і збільшує її. Також у тестовій функції додається завдання у вигляді HashTable, запускаються усі потоки ThreadPool, перевіряється кількість записаних комірок, поточний розмір таблиці, ємність таблиці, процент заповнення, кількість значень за ключем, зберігається таблиця у окремий файл, створюється новий екземпляр HashTable, завантажується із файлу у таблицю дані, виконується ідентична перевірка, знову додається завдання у вигляді екземпляру HashTable, запускаються усі потоки ThreadPool, знову виконується перевірка, зберігається таблиця, створюється новий екземпляр HashTable – III зациквився.

Amazon CodeWhisperer спробував використати усі засоби класів HashTable, ThreadPool, а саме методи збереження, отримання значень, усього вмісту, виконав низку подібних перевірок.

III допустився таких помилок:

1) не виділив у окремі функції використання обох класів і виконав у межах одної функції багато не пов'язаної роботи;

2) знову записував значення у HashTable за єдиним ключем;

3) використав робочу функцію для ThreadPool, яка не виконує обробку даних;

4) у якості завдань для ThreadPool використовував HashTable, а не спеціальну структуру;

5) не виділив для роботи ThreadPool окремий потік;

6) використав надлишкові перевірки, що повторювалися;

7) зациквився.

Тема взаємодії HashTable та ThreadPool не розкрита через неправильне використання цих компонентів разом.

9. Тут слід визначити що робить клас HopfieldNetwork і покрити його тестами. ШІ не зміг пояснити суть класу HopfieldNetwork, адже він призначений, щоб писати код.

Щоб покрити тестами код, програматор створив окремий клас HopfieldNetworkTest і єдиний тестовий метод із анотацією JUnit. У методі було ініціалізовано 2 масиви, один навчальний, другий – тестовий. Було ініціалізовано HopfieldNetwork, натреновано мережу на першому наборі даних, перевірено на другому наборі даних. Проведено ряд перевірок: чи сходиться результат тестування із другим набором, чи збігається із очікуваним кількість записаних патернів, чи дорівнює записаний патерн наборам даних, результату тестування.

При спробі створити ще декілька тестів ШІ видавав результат ідентичний першому тестові. Якщо спробувати натякнути ШІ збільшити кількість перевірок, він зациклиться.

Нижче показано зведення у таблиці 3.1 із результатами перевірки на тестових прикладах кожного фреймворку:

Таблиця 3.1. Результати перевірки фреймворків створення ПЗ. (Додаток Ж)

Жодний сервіс не зміг повноцінно виконати усі завдання. Але окремі із них вдалося виконати на мінімум вимог: ChatGPT зміг більш-менш виконати 1, 4, 7 завдання самостійно, 9 він виконав, але набір даних не був достатньо детальним для перевірки, а для інших йому потрібні вказівки, допомога; Amazon CodeWhisperer зміг виконати більш-менш 1, 7 завдання самостійно, для усіх інших потребує допомоги – це є нормою, адже він допомагає, а не формує готові програми. Якщо узагальнити, кожен сервіс виконує поставлені на них розробниками завдання, а для створення готових програм їм недостатньо можливостей.

3.5. Оцінка придатності існуючих фреймворків для прикладного використання.

Враховуючи попередні результати перевірки фреймворків інтерполюємо їх на реальні приклади використання за призначенням кожного фреймворку. Будуть оцінюватися лише ті фреймворки на яких запускалися приклади.

ChatGPT

Розроблявся розробниками для показу можливостей ШІ, щоб зробити інструменти ШІ доступними для людей, щоб показати необхідність дискусії щодо майбутнього ШІ. Із можливостями цього фреймворку велика кількість спеціалістів буде використовувати його усе частіше для усунення рутини і отримання рекомендацій.

Розробники можуть швидко згадати необхідні конструкції, створювати прості функції, отримувати рекомендації та шаблони щодо складних проектів, отримати інформацію, як розгорнути проект, покрити код модульними тестами, швидко згенерувати дані будь-якої текстової форми.

Письменники, маркетологи, frontend-розробники можуть швидко отримувати нові ідеї для своїх творів, рекламних кампаній, оформлення сайтів.

Школярі, діти можуть швидко і зрозуміло отримати відповіді на велике коло запитань.

Люди, які хочуть створити свій бізнес, можуть отримати нові ідеї, модель може для них сформулювати досить детально опис, код для реалізації ідеї і довести її до етапу, щоб почати заробляти.

Як для процесу розробки, що у цій роботі найбільше цікавить, цей інструмент більше підходить у якості помічника, для усунення рутини. Через свої обмеження, як от діалоговий режим, обсяг результату, здійснення помилок при роботі із точними даними, недостатня

глибина генерування контенту, відсутність самостійності, цей ШІ не може стати повноцінним розробником і налаштовувати процеси усередині компанії.

AgentGPT

Розроблявся розробниками для усунення недоліків ChatGPT, як от діалоговий режим, щоб рекурсивно деталізувати і розвивати вирішення задач. Він може поки що тільки давати динамічний план дій. Через обмеження, як от перебування розробки на стадії бета версії, кількість завдань у списку, зациклювання, неможливість вирішувати задачі, ця модель поки що взагалі не придатна до використання. У ній містяться цікаві ідеї, але без правильного оформлення і пророблення це лише тестовий концепт. Він, як і усі інші сервіси на основі OpenAI API, є доволі примітивним.

Durable

Розроблявся розробниками для створення сайтів для бізнесу. Він створює статичні сайти без поведінки на основі обмеженої кількості даних за стандартним шаблоном, що можна було зробити й раніше. Сформувати комплексні завдання для фреймворку не можна. Великої прикладної користі поки що не має, бо не може виконувати поставлені завдання на рівні людини. Залишається концептом для генерації сайтів.

Amazon CodeWhisperer

Розроблявся розробниками для допомоги програмістам позбутися рутини при написанні коду для невеликих компонентів за описом. Йому важко сприймати великі ідеї, він швидко втрачає контекст, якщо працює самостійно, йому потрібно постійно допомагати. Проте він може створювати прості функції та структури, які виконують більшість поставленої роботи. На цьому етапі розвитку ще немає можливостей, щоб бути надійним компаньйоном, адже не може виконувати будь-яку рутину і зациклюється.

Загальний висновок

Протестовані ШІ ще не готові стати навіть помічниками. Вони мають обмеження, які роблять їх непрактичними для більшості задач. Ці проекти дуже молоді і потребують чимало зусиль на вдосконалення. Водночас вирішувати найпростіші завдання вони можуть. З'являються нові проекти, як от GPT-4, що обіцяють усунути обмеження і вдосконалити процес мислення. Але поки що перевірити їх можливості немає.

Висновки

Протягом даного розділу було:

- показано чого досягли розробники із мовними моделями, скільки зусиль коштувало їхнє створення та налагодження;
- описано обрані фреймворки на основі мовних моделей та інтерфейси роботи із ними;
- створено набір тестових прикладів для перевірки обраних фреймворків;
- перевірено кожен фреймворк на кожному тестовому прикладі і показано їхні можливості;
- оцінено практичну цінність фреймворків для створення ПЗ.

Мовні моделі досягли нечуваних раніше досягнень і тепер можуть вирішувати завдання із широкого кола питань. Для їхнього створення та підтримування роботи розробникам була потрібна потужна інфраструктура та велика команда для налагодження, зміна багатьох рішень. Усі фреймворки засновані на моделях GPT-3+ або їхніх аналогах, проте мають різне призначення. Після їхньої перевірки можна сказати, що вони не готові до практичного використання і можуть вирішувати не всі поставлені задачі. Це зумовлено тим, що більшість проектів на базі GPT-3+ дуже молоді і потребують удосконалення. Існуючі обмеження не дозволяють рухатись далі.

РОЗДІЛ 4. РОЗРОБКА ВЛАСНОЇ КОНЦЕПЦІЇ ФРЕЙМВОРКУ ДЛЯ СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗА МОДУЛЬНИМИ ТЕСТАМИ ТА ПЕРЕВІРКА ЙОГО МОЖЛИВОСТЕЙ.

У даному розділі буде показано, як можна створити фреймворк автоматичної розробки програмного забезпечення. Тут будуть формуватися вимоги, структура, принцип роботи, буде описано його можливості, чи призначений він для практичного використання та буде порівняння із уже вивченими фреймворками.

4.1. Формулювання вимог до власного фреймворку.

Ідея власного фреймворку: low-code рішення у вигляді додатку на Android для складання схем і на основі них збірка програм у власному форматі; сервер на персональному комп'ютері написаний на фреймворкові Spring, що отримує файли створених програм і виконує їх.

До схем в додатку під час роботи можна приєднувати як основу ділянки коду, які будуть у формі окремих блоків. Ці блоки можна складати разом у схему і формувати програму. Блоки мають власні методи так звані «взаємодії», які виконують рекурсивно субметоди своїх компонентів за певним алгоритмом поки не дійдуть до коду. Завдяки цьому можна створювати ланцюги процесів, які контролюють схему – це джерело автоматизації. До цього моменту були описані загальні вимоги.

Це виглядає так, наче розробник сам має створювати програми – на першому етапі це й справді так. Проте додаток для схем у майбутньому зможе сам запускати програми і у реальному часі їх редагувати, відслідковувати зміни. Це дозволить зробити його середовищем для ШІ. І тоді відкриється можливість розробнику створювати схеми автоматизації процесів, які переростуть у автоматизований інструмент. Але це вже виходить поза рамки даної роботи.

Користувач може додавати у схему власні задачі у вигляді блоків, що мають вхідні, вихідні стани і умови. За цими компонентами має бути створене рішення - об'єднання станів у єдину послідовність, яку можна виконати. Така структура нагадує використання модульних тестів за Test DD, але вони тут не у формі коду, а у формі блоків, схем.

Опишемо вимоги до базової версії фреймворку.

Вимоги середовища

- Сервер має працювати на персональному комп'ютері;
- Клієнт має працювати на смартфоні на базі Android.
- Система може бути використана будь-де, де є середовище для клієнта та сервера.

Архітектурні та структурні вимоги

- Система складається із двох частин: сервера та клієнта.
- Обмін інформацією між клієнтом-сервером має відбуватися через розширюваний формат даних.

Функціональні вимоги

Клієнта:

- Зберігання проектів схем;
- створення, видалення, імпорт, експорт проектів;
- переміщення по схемі через жести;
- редагування, створення, видалення, зміна виду, взаємодія між компонентами схеми;
- створення, редагування, видалення коду для нових компонентів;
- підключення до сервера та обмін даними проекту і результатів виконання.

Сервера:

- відслідковування підключень клієнтів;
- отримання даних проектів;
- запуск проектів на виконання;
- збір результатів;
- відправка результатів клієнтам;
- видалення сміття.

Поведінкові, UX/UI вимоги

- Коли користувач заходить у клієнтський додаток він має бачити меню із список проектів(схем) із датою створення, останньої зміни, розміром файлів. Також тут мають бути розташовані налаштування, довідка;
- користувач має можливість створити новий, видалити існуючий проект із меню, імпортувати із пам'яті проект, експортувати назовні проект;
- коли користувач натискає на проект, клієнт має перейти до середовища роботи із проектом. У цьому середовищі увесь простір має бути зайнятий схемою.
- Якщо слід скористатися інструментами у середовищі роботи із проектом, змінити стан середовища, клієнт може взаємодіяти із елементами схеми або провести жест, щоб відкрити контекстне меню, основне меню;
- Контекстне меню повинне містити опції для зміни елементів схеми, їхній вид, видалити їх, створити нові, відредагувати вміст, відкрити ще одне меню;
- У основному меню повинен бути розділ із створенням коду, його редагуванням і інтеграцією у схему;
- У основному меню повинен бути розділ із підключенням до сервера та запуском на ньому готового проекту.

Нижче буде уточнено нефункціональні програмні вимоги – засоби, бібліотеки, інструменти, які необхідні для реалізації системи.

Ці вимоги можна деталізувати під час розробки у міру необхідності, адже це ще експериментальний проект. Залишилося багато не визначених деталей, але даний розділ слугує для окреслення необхідних компонентів для створення власного фреймворку автоматизації розробки.

4.2. Проектування структури власного фреймворку та створення схем UML для опису системи.

На рис. 4.1 подана узагальнена схема, як має працювати фреймворк:

Рис. 4.1. Умовна схема роботи фреймворку (Додаток Д). (Додаток Ж)

Усе це повинно відбуватися одночасно, циклічно: користувач формує свою проблему у вигляді блоків (моделей) на схемі, тим часом фреймворк виконує старі задачі, у ньому асинхронно відбувається багато процесів як на стороні сервера, так і клієнта, постійно оновлюється стан, організовуються блоки, аналізується схема, збираються дані щодо результатів вирішення задач.

На рис. 4.2 показана діаграма прецедентів фреймворку:

Рис. 4.2. Діаграма прецедентів UML фреймворку (Додаток Д). (Додаток Ж)

На діаграмі видно основні способи взаємодії із користувача із клієнтом, адміністратора із сервером, клієнта із сервером. Основний акцент тут йде на можливості редагування, взаємодію і процес виконання програм. Хоча тут це показано як послідовий процес, насправді він є циклічним, паралельним і асинхронним.

Які компоненти необхідні для реалізації системи? За видом архітектури, обладнання, характером задачі можна виокремити такі компоненти:

Клієнт:

- вікно меню проектів, де висвітлено список проектів, створюються, видаляються, редагуються проекти, можна потрапити у налаштування, довідку;
- вікно середовища роботи із проектом, де можна змінювати компоненти схеми, їх взаємодії, стилі, відслідковувати зміни, запускати контекстні меню, заходити у налаштування проекту, запускати редактор коду, оновлювати стан;
- редактор коду, де є інструменти для створення опорних блоків на основі коду, можна імпортувати/експортувати файли;
- проект, що містить діаграму, схему блоків, дані, код базових блоків, дані процесів, конфігурації, підключення;
- модель проекту;
- база даних, що зберігає проекти і їхні дані.

Сервер:

- менеджер клієнтів, який відслідковує підключення, зберігає дані про підключених клієнтів, від'єднує клієнтів;
- менеджер команд, який виконує вказівки клієнтів, відправляє запити менеджеру програм, менеджеру клієнтів, окремим потокам, програмам, змінює конфігурацію;
- менеджер програм, який зберігає усі дані пов'язані із запуском програм від клієнтів, запускає програми, оновлює їхній стан, повертає результати, слідкує за процесами, асинхронними функціями, що виконуються; балансує навантаження;
- менеджер програм контролює окремі середовища виконання програм, які складаються із асинхронних процесів, які взаємодіють між собою, конфігурації виконання, дані;

- менеджер комунікацій через, який з'єднуються між собою процеси програм, база даних, менеджер програм, менеджер команд, менеджер клієнтів;
- центр статистики, що взаємодіє із менеджером програм і збирає результати, логи програм, подій і перетворює їх у придатну для перегляду форму;
- база даних, де зберігаються дані, компоненти програм на довгий період. До неї звертається менеджер комунікацій із проханнями дати, записати дані. Має високорозвинуту організацію за власним форматом, яка допомагає швидко знаходити необхідне.

Між клієнтом і сервером існує двосторонній зв'язок, який може тривати довгий час, роз'єднуватися, відновлюватися. Це забезпечується інтерфейсом зв'язку, стандартними засобами взаємодії у глобальній мережі.

Кожен із цих компонентів може бути як окремою програмою, контейнером, сервісом так і класом.

На рис. 4.3 показана серверна частина UML діаграми класів. Це базова версія, яка постійно буде удосконалюватися. Для концепту достатньо передати загальну структуру. Схема складається із таких компонентів:

- сервер приєднує клієнтів, надсилає їм повідомлення, отримує повідомлення у класі ClientManager;
- ClientManager закодує повідомлення за логікою класу MessageEncoder і розкодує їх MessageDecoder;
- кожен клієнт представлений як окремий екземпляр класу Client;
- усі компоненти спілкуються між собою через клас CommunicationManager. Він приймає повідомлення і відправляє їх через callback'и кожному компоненту;
- CommandManager - дозволяє адміністратору у консольному режимі передавати команди усім компонентам серверу, щоб налаштувати їхню роботу;
- ProgramManager - контролює і організовує роботу програм, які хочуть запустити клієнти;
- Statistics – збирає дані і формує із них результат, історію подій;
- DataManager – звертається до бази даних за проханнями інших компонентів і надає зручний API для цього;
- Program – екземпляр програми, що містить усі дані для запуску програми та процесів всередині неї;
- Model – це елементарна одиниця сутностей, із яких складається програма і схеми;
- Interaction – це елементарна одиниця взаємодії між Model. На схемі рис. 4.3 і 4.4 не видно цього класу, тому його виділено окремо на рис. 4.5. Завдяки Model і Interaction утворюється універсальна модель для обміну даних, запуску програм, формування схем, виконання складних завдань. На схемі рис. 4.3 видно, що Program і Message наслідують поведінку цих класів – це полегшує використання Model і Interaction, переналаштовує усю систему для роботи із ними;
- BaseModel (рис. 4.3, 4.4), BaseInteraction (рис. 4.5) дають можливість перетворити будь-який клас на Model чи Interaction, щоб програми могли працювати із ними.

Рис. 4.3. Діаграма класів UML фреймворку. Сервер (Додаток Е). (Додаток Ж)

На рис. 4.4 зображено клієнтську частину UML діаграми класів. Вона містить такі компоненти:

- Model;
- Interaction;
- BaseModel;
- BaseInteraction;
- ConnectManager – керує з'єднаннями, пересилає, отримує дані від серверів;
- Server – представлення сервера на стороні клієнта;
- MessageEncoder;
- MessageDecoder;
- MainActivity – містить поведінку та вигляд інтерфейсу користувача із списком проектів;
- ProjectEnvironmentActivity – містить поведінку та вигляд інтерфейсу користувача для редактора схем і середовища проекту;
- CodeEditorActivity - містить поведінку та вигляд інтерфейсу користувача для редактора коду;
- MainViewModel – містить дані про стан інтерфейсу користувача із списком проектів і взаємодіє за потреби із Repository;
- ProjectEnvironmentViewModel - містить дані про стан інтерфейсу користувача середовища проекту і взаємодіє за потреби із Repository;
- Repository – взаємодіє із базою даних за потребами інших компонентів у зручній формі;
- Arrow, Block, Style – передають дані, вигляд і поведінку елементарних візуальних елементів схем;
- Project – містить дані про окремий проект, компоненти схем, що записані у Program;
- Program.

Рис. 4.4. Діаграма класів UML фреймворку. Клієнт (Додаток Е). (Додаток Ж)

Рис. 4.5. Діаграма класів UML фреймворку. Interaction (Додаток Е). (Додаток Ж)

На узагальнені схемі опущено деякі деталі роботи фреймворку. На рис. 4.6 діаграма станів циклу життя задачі уточнює циклічну природу фреймворку. Ці процеси можуть виконуватися як паралельно, так і асинхронно:

Рис. 4.6. Діаграма станів UML циклу життя задачі (Додаток Є). (Додаток Ж)

4.3. Опис використаних мов, засобів, бібліотек та утиліт.

У проекті власного фреймворку має бути використано такі засоби:

IntelliJ IDEA

Це інтегроване середовище розробки від компанії JetBrains для великої кількості мов програмування, форматів, файлів розмітки. Дуже популярне рішення, яке все більше витісняє конкурентів завдяки своїм можливостям. IDE:

- має безплатну версію;
- має широкий інструментарій для рефакторингу, тестування;

- підтримує розвинуту систему плагінів ⁵⁶;
- має широкі можливості для налаштування середовища;
- має простий не перевантажений вигляд і доступність основних компонентів для роботи;
- підтримує системи контролю версій;
- підтримує системи збірки Gradle, Maven;
- має засоби для контролю артефактів, бібліотек, проектів, модулів.
- має засоби для глибокого аналізу коду ⁵⁷.

Було обрано через те, що це найпопулярніша IDE для Java, Kotlin, що призначена для розробки мобільних і веб застосунків.

Рис. 4.7. IntelliJ IDEA IDE. (Додаток Ж)

Android Studio

Це IDE від компанії Google для розробки мобільних застосунків для платформи Android. Заснована на IDE IntelliJ IDEA, але має власну екосистему. Android Studio:

- має розширений інтерактивний редактор макетів;
- підтримує симуляцію додатків у окремому середовищі на персональному комп'ютері;
- широкі можливості для систем збірок Gradle, Maven;
- розвинута автоматизована система збірки у готові додатки із можливістю налаштування;
- має вбудовану систему перевірки підписів для покращення безпеки;
- має багато шаблонів для додатків ⁵⁸.

Було обрано тому, що це основне середовище, адаптоване для розробки саме Android додатків від офіційної компанії, що представляє цю платформу. При створенні проектів не потрібно налаштовувати початкову структуру файлів додатку для тестування, запуску, assets. Під час написання коду система аналізу коду допомагає підібрати необхідні компоненти, що притаманні Android.

Java

Мультипарадигмальна кросплатформена мова програмування із строгою типізацією, великим нахилом до ООП. Широко використовується як у корпоративному, так і community середовищі. Має розвинуту систему стандартних типів, інтерфейсів, структур даних, класів тощо. Має величезні можливості для розширення: створення модулів, бібліотек, власних мов програмування на власній віртуальній машині Java Runtime Machine для різноманітних платформ.

Було обрано тому, що це основна мова програмування для фреймворку Spring про який буде йтися далі. Java має власну стандартну бібліотеку Reflection, ClassLoader для зміни коду і завантаження файлів коду на виконання під час виконання основної програми.

⁵⁶ IntelliJ IDEA. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/IntelliJ_IDEA

⁵⁷ Функциональность. JetBrains : веб-сайт. URL: <https://www.jetbrains.com/ru-ru/idea/features/>

⁵⁸ Android Studio. Вікіпедія(ru) : веб-сайт. URL: https://ru.wikipedia.org/wiki/Android_Studio

Spring

Програмний каркас (фреймворк) для розробки додатків на мові Java, що опирається на інверсію керування ⁵⁹. Налаштовується за допомогою XML файлів та анотацій. Має готові прості у використанні рішення для веб-розробки, тестування, приєднання баз даних, налаштування процесів безпеки, налаштування програм віддалено завдяки інтерфейсам, конфігураціям, реєстрації користувачів.

Завдяки готовим компонентам, інверсії керування, віддаленій конфігурації, широким можливостям для налаштування, розгортання власних серверів було обрано цей фреймворк.

Kotlin

Мультипарадигмальна кросплатформена мова програмування із строгою типізацією. Ще доволі молода, але за наявними засобами вона є серйозним сучасним рішенням, що не має недоліків таких мов як C++, Java, поєднує можливості як скриптових так і строго типізованих мов. Засновник Android, Google, перевела розробку сучасних додатків із Java на Kotlin. Автори Kotlin ставили перед собою ціль створити лаконічнішу та типобезпечнішу мову, ніж Java, і простішу, ніж Scala ⁶⁰. Наслідками спрощення, порівняно з Scala стали також швидша компіляція та краща підтримка IDE. Kotlin може запускатися також на JVM і унаслідував багато свої можливостей від Java, але у ньому менше boilerplate коду і із ним важче допустити помилку.

Було обрано через те, що бібліотека для створення користувацьких інтерфейсів на Android Jetpack Compose підтримується лише на Kotlin.

Android

Найпопулярніша платформа для мобільних пристроїв, смартфонів, телевізорів, вбудованих в авто мультимедійних систем. Доступна платформа для портативних пристроїв із власною стандартизованою екосистемою із можливістю розширення. Підтримує багато протоколів безпроводного зв'язку, сенсорів, забезпечує безпечне середовище для запуску окремо додатків, підтримує стандарти відео, аудіо, 3D візуалізації, має великі можливості для створення користувацьких інтерфейсів для управління жестами, дотиком. Ця система дозволила зрівняти і перенести можливості персональних комп'ютерів і портативних пристроїв.

Була обрана через зручність використання, доступність, поширеність у портативних пристроях, широту можливостей для розробки і розгортання власних систем.

Jetpack Compose

Бібліотека для створення користувацьких інтерфейсів для додатків платформи Android. До цієї бібліотеки в Android існував єдиний офіційний спосіб представлення інтерфейсів через файли XML та інверсію управління – імперативний спосіб («що зробити?»). Це геть не зручно і потребує багато boilerplate коду. Jetpack Compose додав ще один спосіб представлення інтерфейсів, що вже існував у індустрії, але якого не було у Android – декларативний («як зробити?»). При декларативному способі представлення усі налаштування елементів і

⁵⁹ Spring Framework. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/Spring_Framework

⁶⁰ Kotlin. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/Kotlin>

структура макету представлені у кодї, а не окремих форматованих файлах. Це дуже полегшує розробку, усуває багато обмежень. Саме тому було обрано дану бібліотеку.

JSON

Стандарт представлення даних. Дуже простий і розширюваний. Широко використовується для взаємодії у глобальній мережі. Прийшов на заміну XML у багатьох рішеннях. Дозволив не вигадувати власних форматів. Представляється як ієрархія із фігурних дужок, що містить примітивні дані як от строки (“текст”), числа, булеві значення. Кожен компонент названий, має власний ідентифікатор і може зберігати як примітивні дані або списки.

Було обрано для обміну даних між клієнтом на платформі Android та сервером написаним засобами фреймворку Spring на персональному комп’ютері.

Висновки

Протягом даного розділу було:

- описано ідею власного фреймворку автоматизації розробки ПЗ;
- визначено базові вимоги до базової версії фреймворку;
- описано структуру системи як описово, так і через UML схеми;
- вказано і описано засоби, бібліотеки, інструменти необхідні для реалізації.

Створення власного фреймворку автоматизації ПЗ є дуже комплексною задачею, яка не вміщується у обсяги даної роботи. Тому було окреслено вид системи, описано її функції, необхідні засоби, щоб хоча б уявити можливості системи і у майбутньому реалізувати її. У наступному останньому розділі буде оцінено концепцію власного фреймворку і порівняно із існуючими.

РОЗДІЛ 5. ПЕРЕВІРКА, ПОРІВНЯННЯ ТА ОЦІНКА ВЛАСНОГО ФРЕЙМВОРКУ.

5.1. Перевірка та тестування власного фреймворку на тестових прикладах.

Описана у 4 розділі система є концептом, за яким можна реалізувати систему. У даному розділі можна оцінити чи здатна система виконати тестові приклади.

Описана система – це середовище для запуску програм за схемами у зручному вигляді. Вона має засоби для швидкої організації існуючих компонентів коду. Вона не містить засоби автоматизації сама по собі. Але у ній можна створити проект схеми, яка зможе автоматизувати процеси за вимогами у вигляді умовних модульних тестів.

На етапі концепції виконати тестові приклади не можна.

Але можна уявити таку ситуацію: фреймворк відповідає вимогам, тобто може виконувати програми за діаграмами, користувач створив схеми автоматизації, які отримують опис задач(блоки) і за ними створюють рішення. Рішення формується через визначення різниці між початковими станами і кінцевими. Алгоритм за різницею підбирає існуючі блоки, як елементи ланцюга рішень.

Тепер спробуємо спрогнозувати результат запуску тестів на тестових прикладах із розділу 3.3.

1. Алгоритм пошуку в ширину.

Фреймворк має розуміти, що таке пошук, де він має відбуватися, що таке вузол, і має володіти базовими структурами коду, щоб сформувати алгоритм.

Користувач створив дерево у схемі із вузлів, фреймворк знає що таке пошук, користувач сформував початковий і кінцевий стани, умови задачі. Але йому важко усвідомити суть алгоритму через те, що він може створювати лише лінійні алгоритми і йому важко піддається обробка сенсу. Тому він створить непридатний алгоритм.

- 2. Сайт із алгоритмом пошуку оптимальних ходів у партії в шахи.**
- 3. Сайт для компанії, що продає дрони.**

Фреймворк має знати детальну структуру сайту, а також мати опис алгоритмів і їхні артефакти. Оскільки детальна структура і так є необхідним результатом, а складні алгоритми фреймворк самостійно створювати не може, ці завдання також не під силу йому.

- 4. Створення модульних тестів для двох функцій.**
- 5. Створення функцій за модульними тестами.**

Модульні тести і функції мають сигнатуру тому їх зручно подавати у вигляді блоків на схемі. Створити модульні тести для функцій фреймворк може спробувати, але функції для модульних тестів – ні, тому що він не може створювати складні алгоритми.

Прогнозування чи виконає дані тестові приклади фреймворк не є коректним. Це залежить від розвитку фреймворку, етапу реалізації, конкретних спроб вирішення. Але робота вичерпала свої рамки тому для неї буде достатньо концепції фреймворку.

5.2. Порівняння власної реалізації із існуючими фреймворками.

Ще із розділу 3.4 було виявлено, що існуючі фреймворки не змогли впоратися із тестовими прикладами, лише ChatGPT та Amazon CodeWhisperer змогли знайти реалізацію

простих функцій. Власна реалізація навіть після прогнозування не зможе вирішити тестові приклади. Все ж таки у них є щось спільне.

Саме складність поставлених користувачами і практичних задач легко вводить у ступор будь-який фреймворк, що було показано на тестових прикладах. Це не так провина базових ідей фреймворків, як дуже малий проміжок часу їхнього розвитку. Вони ще недостатньо розвинуті.

Так само й власний фреймворк має пройти ряд заходів, щоб конкретизувати структуру, властивості. Повинна з'явитися база блоків, спільнота користувачів, які винайдуть автоматизовані схеми рівня ШІ.

Існуючі фреймворки розробки ПЗ базуються на мовних моделях, можуть працювати із текстом, картинками і вирішувати широке коло проблем у діалоговому режимі. Але вони видають примітивні рішення, можуть працювати лише із простими задачами і часто помиляються, пропускають частини при роботі із складними задачами.

Власна концепція базується на ідеї, що будь-що може бути представлено за допомогою схем. Ці схеми можуть динамічно розвиватися, що дає можливості для породження автоматизованих схем. Схеми позбавлені обмежень коду. Власний фреймворк – це середовище для майбутнього ШІ. Ось ще один аргумент чому не коректно порівнювати його із існуючими фреймворками, які одразу призначені для вирішення задач.

До мінусів власної реалізації можна віднести: недостатня деталізація проекту, важко обробляти великі масиви примітивів без складної, організованої структури, є проблеми із сприйняттям сенсу і побудовою складних функцій.

5.3. Перспективи подальшої розробки власного фреймворку.

Власний фреймворк призначений, як середовище для ШІ. Саме із цим і буде пов'язаний його розвиток. Для цього слід виконати такі задачі:

- створити базову модель ШІ, за якими принципами вона буде функціонувати, які ідеї допоможуть вирішувати задачі;
- створити принцип переведення будь-яких сутностей у організовані моделі чисел і математичних операцій, які можна виконувати на GPU.
- створити бібліотеку стандартних блоків;
- забезпечити умови для довгої, безперебійної роботи середовища;
- організувати виконання обчислень на стороні клієнта, без сервера.

Для зручності можна додати такі можливості:

- підтримка клієнтів у браузері як веб-застосунки;
- створити засоби для візуалізації процесів схем, статистики, продуктивності, балансу.

Власний фреймворк у майбутньому може змінити підхід до розробки. Люди будуть менше програмувати, а будуть представляти ідеї у вигляді схем. Їм не обов'язково бути прив'язаними до конкретного місця роботи.

Комп'ютери зможуть розуміти усі деталі сутностей реального світу і світу ідей. Усе краще будуть працювати над виконанням складних задач. Зможуть працювати із будь-якими форматами представлення даних. І це все не у діалоговому режимі, а динамічно.

За власною думкою мовні моделі мають фундаментальні обмеження, що знижують їхню продуктивність і не дають розвиватися далі. Це було сказано ще у розділі 3.

Майбутнє буде за фреймворками, які зможуть працювати із будь-яким форматом, постійно, без зупинок виконувати безліч процесів.

Висновки

У даному розділі було:

- перевірено чи зможе власний фреймворк пройти тестові приклади за концепцією;
- порівняно концепцію власної системи із існуючими фреймворками;
- визначено проблеми, які слід вирішити;
- визначено перспективи розвитку власного фреймворку.

Концепція власного фреймворку потребує доопрацювання. Є багато неуточнених моментів і відсутність робочої версії унеможливорює коректне порівняння власної концепції із існуючими фреймворками. Розробка прототипу потребує обсягу ресурсів, яких недостатньо в рамках даної роботи. Але вже визначення напрямку розвитку достатньо для оцінки великих перспектив.

РОЗДІЛ 6. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ВЛАСНОГО ФРЕЙМВОРКУ⁶¹.

6.1. Постановка задачі проектування.

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки фреймворку створення ПЗ. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для взаємодії із користувачем, збереження, структуризації введених даних, моделювання на основі введених даних.

Технічні вимоги до програмного продукту є наступні:

- функціонування на портативних пристроях із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту;
- підтримка стандартних засобів зв'язку із глобальною мережею.

6.2. Обґрунтування функцій програмного продукту.

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє створювати і запускати автоматизовані програми у візуальному вигляді. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір бібліотеки для створення UI.

F_2 – створення базових блоків.

F_3 – вибір формату обміну даних (між сервером та клієнтом).

F_4 – вибір методу створення сервера.

Кожна з цих функцій має декілька варіантів реалізації:

1. Функція F_1 :
 - а) Layout;
 - б) Jetpack Compose.
2. Функція F_2 :
 - а) Стандартна бібліотека блоків;
 - б) Базові блоки на основі коду.
3. Функція F_3 :
 - а) JSON;
 - б) Власний.
4. Функція F_4 :
 - а) Spring Boot;

⁶¹ В.П. Пашін, В.В. Романов, Н.В. Сгорова. Методичні вказівки до виконання економіко-організаційного розділу дипломних проектів (робіт) бакалаврів і спеціалістів для студентів Інституту прикладного системного аналізу : навчальне видання. Київ : НТУУ “КПІ”, 2011. – 118 с.

б) Без каркасу.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 6.1).

Рис. 6.1. Морфологічна карта реалізацій основних функцій. (Додаток Ж)

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 6.1.

Таблиця 6.1. Позитивно-негативна матриця. (Додаток Ж)

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам.

Функція F_1 :

Перевага надається зручності програмування та можливості зміни UI під час виконання програми. Для спрощення роботи по написанню коду варіант А має бути відкинутий. Обирається варіант Б.

Функція F_2 :

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3 :

Перевага надається наявності готових бібліотек для роботи. Варіант А більше підходить для системи. Варіант Б відкидається.

Функція F_4 :

Перевага надається готовим рішенням, інверсії керування, тому для системи підходить варіант А. Варіант Б відкидається.

Таким чином, будемо розглядати такі варіанти реалізації програмного продукту:

$$F1 \text{ б} - F2 \text{ а} - F3 \text{ а} - F4 \text{ а}$$

$$F1 \text{ б} - F2 \text{ б} - F3 \text{ а} - F4 \text{ а}$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

6.3. Обґрунтування системи параметрів програмного продукту.

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- $X1$ – швидкодія трансляції схем – яку кількість блоків за секунду (Кблоків/с) може перетворити система у числову модель на CPU;

- X2 – швидкодія обчислення числових даних – яку кількість обчислень із плаваючою точкою за секунду може виконати система обробляючи числову модель на GPU (Mflops);
- X3 – об’єм пам’яті для обчислень та збереження даних (Гб);
- X4 – частота оновлення стану (за секунду, оновлень/с);
- X5 – потенційний об’єм програмного коду (у рядках).

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 6.2.

Таблиця 6.2 - Основні параметри програмного продукту. (Додаток Ж)

За даними таблиці 6.2 будуються графічні характеристики параметрів – рис. 6.2 – рис. 6.6.

Рис. 6.2. X1 – швидкодія трансляції схем. (Додаток Ж)

Рис. 6.3. X2 – швидкодія обчислення числових даних. (Додаток Ж)

Рис. 6.4. X3 – об’єм даних. (Додаток Ж)

Рис. 6.5. X4 – частота оновлення. (Додаток Ж)

Рис. 6.6. X5 – об’єм коду. (Додаток Ж)

6.4. Аналіз експертного оцінювання параметрів.

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів. Чим вищий ранг, тим більш значущим є параметр;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 6.3.

Таблиця 6.3 - Результати ранжування параметрів. (Додаток Ж)

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

- а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} \quad (6.1)$$

$$R = \frac{Nn(n+1)}{2} = 105, \quad (6.2)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 21 \quad (6.3)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (6.4)$$

Сума відхилень по всіх параметрах повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 366 \quad (6.5)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 366}{7^2(5^3 - 5)} = 0.747 > W_k = 0.67. \quad (6.6)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 6.4.

Таблиця 6.4 - Попарне порівняння параметрів. (Додаток Ж)

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (6.7)$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості K_{vi} за наступними формулами:

$$K_{vi} = \frac{b_i}{\sum_{i=1}^n b_i} \quad (6.8)$$

$$b_i = \sum_{i=1}^N a_{ij} \quad (6.9)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{вi}} = \frac{b'_i}{\sum_{i=1}^n b'_i}, \quad (6.10)$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j \quad (6.11)$$

Як видно з таблиці 6.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 6.5 - Розрахунок вагомості параметрів. (Додаток Ж)

6.5. Аналіз рівня якості варіантів реалізації функцій.

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо. Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 6.6):

$$K_K(j) = \sum_{i=1}^n K_{ei,j} B_{i,j}, \quad (6.12)$$

де n – кількість параметрів; K_{ei} – коефіцієнт вагомості i -го параметра; B_i – оцінка i -го параметра в балах.

Таблиця 6.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП. (Додаток Ж)

За даними з таблиці 6.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}], \quad (6.13)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 4.56 + 6.72 + 1.95 + 1.08 = 14.31;$$

$$K_{K2} = 4.56 + 9.3 + 1.95 + 1.08 = 16.89.$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

6.6. Економічний аналіз варіантів розробки ПП.

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_0 = T_P \cdot K_{\Pi} \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (6.14)$$

де T_P – трудомісткість розробки ПП;

K_{Π} – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 37$ людиноднів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_{\Pi} = 1.8$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0.9$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 37 \cdot 1.8 \cdot 0.9 = 59,94 \text{ людиноднів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_P = 29$ людиноднів, $K_{\Pi} = 0.9$, $K_{СК} = 1$, $K_{СТ} = 0.8$:

$$T_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88 \text{ людиноднів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (59,94 + 20,88 + 4,8 + 20,88) \cdot 8 = 852 \text{ людиногодин.}$$

$$T_{II} = (59,94 + 20,88 + 6,91 + 20,88) \cdot 8 = 868,88 \text{ людиногодин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь: два Middle Java Backend Developer'и з окладом 2747 дол.⁶², що за поточним курсом 37 грн. за 1 дол.⁶³ дорівнює 101 639 грн., та один Middle Android

⁶² DOU Зарплати, Middle Software Engineer, Java, 2-4 роки. DOU.UA : вебсайт. URL: <https://jobs.dou.ua/salaries/?period=2022-12&position=Middle%20SE&technology=Java&experience=2-4> (дата звернення: 20.05.2023)

⁶³ Долар США – Українська гривня. Google Фінанси : вебсайт. URL: https://www.google.com/finance/quote/USD-UAH?sa=X&ved=2ahUKewjY-NiR4Zj_AhUGgosKHUT2CDMqMqY0JegQICBAc (дата звернення: 20.05.2023)

Developer з окладом 2300 дол.⁶⁴ (85 100 грн.). Визначимо середню зарплату за годину за формулою:

$$C_{\text{ч}} = \frac{M}{T_m \cdot t} \text{ грн.}, \quad (6.15)$$

де M – місячний оклад працівників; T_m – кількість робочих днів у місяць;

t – кількість робочих годин в день.

$$C_{\text{ч}} = \frac{101639 + 101639 + 85100}{3 \cdot 21 \cdot 8} = 572.18 \text{ грн.} \quad (6.16)$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{\text{зп}} = C_{\text{ч}} \cdot T_i \cdot K_{\text{д}}, \quad (6.17)$$

де $C_{\text{ч}}$ – величина погодинної оплати праці програміста; T_i – трудомісткість відповідного завдання; $K_{\text{д}}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I. } C_{\text{зп}} = 572.18 \cdot 852 \cdot 1.2 = 584\,996.83 \text{ грн.}$$

$$\text{II. } C_{\text{зп}} = 572.18 \cdot 868.88 \cdot 1.2 = 596\,586.91 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%⁶⁵:

$$\text{I. } C_{\text{вд}} = C_{\text{зп}} \cdot 0.22 = 584\,996.83 \cdot 0.22 = 128\,699.3 \text{ грн.}$$

$$\text{II. } C_{\text{вд}} = C_{\text{зп}} \cdot 0.22 = 596\,586.91 \cdot 0.22 = 131\,249.12 \text{ грн.}$$

Максимальна сума ЄСВ - 22110 грн. (за місяць)⁶⁵. ЄСВ за увесь час роботи не повинно перевищувати таких значень:

$$\text{I. } C_{\text{вдmax}} = \frac{852 \cdot 22110}{8 \cdot 21} = 112\,129.29 \text{ грн.}$$

$$\text{II. } C_{\text{вдmax}} = \frac{868.88 \cdot 22110}{8 \cdot 21} = 114\,350.81 \text{ грн.}$$

Мінімальна сума ЄСВ – 1474 грн. (за місяць)⁶⁵. ЄСВ за увесь час роботи не повинно бути нижче таких значень:

$$\text{I. } C_{\text{вдmin}} = \frac{852 \cdot 1474}{8 \cdot 21} = 7475.29 \text{ грн.}$$

$$\text{II. } C_{\text{вдmin}} = \frac{868.88 \cdot 1474}{8 \cdot 21} = 7623.39 \text{ грн.}$$

Відрахування на ЄСВ перевищує максимальне значення, тому його значенню буде присвоєно максимальне значення:

$$\text{I. } C_{\text{вд}} > C_{\text{вдmax}} \rightarrow C_{\text{вд}} = C_{\text{вдmax}} = 112\,129.29 \text{ грн.}$$

$$\text{II. } C_{\text{вд}} > C_{\text{вдmax}} \rightarrow C_{\text{вд}} = C_{\text{вдmax}} = 114\,350.81 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. ($C_{\text{м}}$)

⁶⁴ DOU Зарплати, Middle Software Engineer, Mobile, Kotlin, 2-4 роки. DOU.UA : вебсайт. URL:

<https://jobs.dou.ua/salaries/?period=2022->

[12&position=Middle%20SE&domain=Mobile&technology=Kotlin&experience=2-4](https://jobs.dou.ua/salaries/?period=2022-12&position=Middle%20SE&domain=Mobile&technology=Kotlin&experience=2-4) (дата звернення: 20.05.2023)

⁶⁵ Розміри ЄСВ-2023. Дебет-Кредит : вебсайт. URL: <https://services.dtk.ua/catalogues/indexes/13> (дата звернення: 20.05.2023)

Так як три ЕОМ обслуговують трьох програмістів: 2 з окладами 101 639 грн. і 1-го із окладом - 85 100 грн., з коефіцієнтом зайнятості 0,2 то для всіх машин отримаємо:

$$C_{\Gamma} = 12 \cdot M \cdot K_3 = 12 \cdot (2 \cdot 101\,639 + 85\,100) \cdot 0,2 = 692\,107.2 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{3П} = C_{\Gamma} \cdot (1 + K_3) = 692\,107.2 \cdot (1 + 0.2) = 830\,528.64 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{ВІД} = C_{3П} \cdot 0.22 = 830\,528.64 \cdot 0,22 = 182\,716.3 \text{ грн.}$$

Відрахування не перевищує максимального внеску за місяць:

$$C_{ВІДміс} = C_{ВІД} / 12 = 182\,716.3 / 12 = 15\,226.35 < 22\,110 = C_{ВІДmax}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості 3 ЕОМ – 120000 грн.

$$C_A = K_{TM} \cdot K_A \cdot C_{ПР} = 1.4 \cdot 0.12 \cdot 120\,000 = 20\,160 \text{ грн.,}$$

де K_{TM} – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; K_A – річна норма амортизації; $C_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{TM} \cdot C_{ПР} \cdot K_P = 1.4 \cdot 120\,000 \cdot 0.08 = 13\,440 \text{ грн.,}$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу персонального комп'ютера за рік розраховуємо за формулою:

$$\begin{aligned} T_{\text{ЕФ}} &= (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.35 = \\ &= 627.2 \text{ години,} \end{aligned}$$

де D_K – календарна кількість днів у році; D_B , D_C – відповідно кількість вихідних та святкових днів; D_P – кількість днів планових ремонтів устаткування; t – кількість робочих годин в день; K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = N \cdot T_{\text{ЕФ}} \cdot N_C \cdot K_3 \cdot C_{\text{ЕН}} = 3 \cdot 627.2 \cdot 0.2 \cdot 0.3 \cdot 4.86889 = 549.69 \text{ грн.,}$$

де N – к-сть ЕОМ, N_C – середньо-споживча потужність приладу; K_3 – коефіцієнтом зайнятості приладу; $C_{\text{ЕН}}$ – тариф за 1 кВт-годин електроенергії⁶⁶.

Накладні витрати розраховуємо за формулою:

$$C_H = C_{ПР} \cdot 0.67 = 120\,000 \cdot 0.67 = 80\,400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

⁶⁶ Ціни на універсальні послуги, період дії ціни – травень 2023 року, для малих побутових споживачів, що приєднані до мереж ПрАТ "ДТЕК Київські електромережі", із ПДВ. Yasno : вебсайт. URL: <https://yasno.com.ua/business/b2b-tariffs> (дата звернення: 21.05.2023)

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}}, \quad (6.18)$$

$$C_{\text{ЕКС}} = 830\,528.64 + 182\,716.3 + 20\,160 + 13\,440 + 549.69 + 80\,400 = 1\,127\,794.63 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 1\,127\,794.63 / 627.2 = 1\,798.14 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{\text{М}} = C_{\text{М-Г}} \cdot T, \quad (6.19)$$

$$\text{I. } C_{\text{М}} = 1\,798.14 \cdot 852 = 1\,532\,015.28 \text{ грн.}$$

$$\text{II. } C_{\text{М}} = 1\,798.14 \cdot 868,88 = 1\,562\,367.88 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_{\text{Н}} = C_{\text{ЗП}} \cdot 0,67, \quad (6.20)$$

$$\text{I. } C_{\text{Н}} = 1\,532\,015.28 \cdot 0,67 = 1\,026\,450.24 \text{ грн.}$$

$$\text{II. } C_{\text{Н}} = 1\,562\,367.88 \cdot 0,67 = 1\,046\,786.48 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}}, \quad (6.21)$$

$$\text{I. } C_{\text{ПП}} = 584\,996.83 + 112\,129.29 + 1\,532\,015.28 + 1\,026\,450.24 = 3\,255\,591.64 \text{ грн.}$$

$$\text{II. } C_{\text{ПП}} = 596\,586.91 + 114\,350.81 + 1\,562\,367.88 + 1\,046\,786.48 = 3\,320\,092.08 \text{ грн.}$$

6.7. Вибір кращого варіанту ПП техніко-економічного рівня.

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{\text{К}j} / C_{\text{Ф}j}, \quad (6.22)$$

$$K_{\text{ТЕР}1} = 14.31 / 3\,255\,591.64 = 4.3955 \cdot 10^{-6},$$

$$K_{\text{ТЕР}2} = 16.89 / 3\,320\,092.08 = 5.0872 \cdot 10^{-6}.$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{\text{ТЕР}2} = 5.0872 \cdot 10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишилися після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $K_{\text{ТЕР}} = 5.0872 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

F1) вибір бібліотеки для створення UI – Jetpack Compose;

F2) створення базових блоків – базові блоки на основі коду;

F3) вибір формату обміну даних – JSON;

F4) вибір методу створення сервера - Spring Boot.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, гнучку реалізацію базових блоків програми за допомогою коду, доступний функціонал для роботи. Для розробника даний варіант дає простоту програмування модулів без boilerplate коду для сервера та клієнта, створення базового набору блоків надається користувачу.

Висновки

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують. Вирахувано витрати для створення ПП на основі кожного набору варіантів реалізації функцій.

На основі аналізу вибрано варіант реалізації програмного продукту. Виявилось, що варіант із базовими блоками на основі коду має більший технічно-економічний рівень, ніж варіант із стандартною бібліотекою блоків.

ВИСНОВКИ

Розробка програмного забезпечення за короткий час свого існування змогла стати чи не найрозвинутішою галуззю завдяки своїй різносторонності та самодостатності. Розробку супроводжує багато процесів, які важко контролювати без правильного підходу. Багато процесів виконують люди. Мета даної роботи була показати складність сучасної розробки, у якій задіяні люди, а також спрогнозувати наслідки автоматизації процесів і заміни людей у рутинних задачах.

Методології розробки програмного забезпечення дозволяють правильно організувати розробку і сконцентруватися на критичних моментах. Але від вибору правильної методології і дотримання її умов залежить успіх усього проекту.

У ході цієї роботи було створено модель порівняння методологій. Її легко налаштовувати і отримувати критерії вибору під власний проект. Використовувати її потрібно із увагою до деталей, адже її ефективність залежить від вхідних даних, системи оцінювання, які налаштовувалися практично. Тут необхідна участь експерта.

Під час створення рейтингу методологій за сценаріями малих команд першість взяли Lean, Waterfall та гнучкі методології Scrum, Agile, Kanban. Для сценаріїв великих команд - першість взяли Lean, Waterfall, V-Model, RUP, Spiral. Загальна тенденція рейтингів співпадає із власними очікуваннями. Тому модель можна використовувати практично. Але її також слід розширювати для закріплення і уточнення результатів.

Автоматизація розробки є неминучою через прагнення розробників досягти оптимуму продуктивності як програм, так і самої розробки. Наслідком цього буде трансформація методологій, зменшення ролі людей, усунення рутинних задач. Усі існуючі методології призначені більше для людей. У майбутньому вони відсіюються або еволюціонують під умови роботи інтелектуальних агентів. Люди будуть лише контролювати увесь процес розробки, задавати вимоги, спілкуватися із клієнтами. Машини будуть генерувати увесь код, тому проектування для людей відійде на другий план, а вимоги на перший.

Вимоги до програмного забезпечення мають особливе значення, адже від них залежать наступні етапи розробки, вектор розвитку, підбір засобів. Люди навчилися представляти вимоги найрізноманітнішими способами, щоб коректніше передати суть і домогтися проходження валідації. Під впливом ШІ представляти вимоги стане легше, появляться нові комбіновані універсальні формати, які відповідатимуть потребам і будуть гнучкими.

Подальший прогрес удосконалення розробки ПЗ буде залежати від сучасних трендів розвитку. Нині набирають обертів мовні моделі на основі нейронних мереж. Чат-бот ChatGPT розпочав нову епоху автоматизації процесів, завдяки здатності за допомогою тексту виконувати задачі будь-якого виду. Достатньо лише надати опис. Він заклав фундамент для багатьох інших проектів завдяки власному API.

Мовна модель доволі молода концепція, як і проекти на їхній основі. Перевірка фреймворків на основі таких моделей у якості інструменту автоматизації розробки показала, що вони можуть вирішувати прості задачі, створювати прості функції. Але вони помиляються і заходять у глухий кут при вирішенні складніших задач, до яких їх не готовили. Це зумовлено фундаментальними обмеженнями мовних моделей, як от опора на мову та нейронні мережі. Мовою не можна передати усі деталі світу образів, якими мислить людина, а нейронні мережі

не можуть працювати і вчитися у реальному часі на складних випадках без деградації. Потрібно вводити нові архітектури.

Власна концепція фреймворку для створення ПЗ закладає основу для виправлення проблем мовних моделей. ШІ повинен навчитися працювати динамічно над вирішенням взаємопов'язаних задач. Для цього слід створити для нього середовище, у якому ШІ може власними силами змінювати свою структуру. Власна концепція намагається відтворити це середовище через діаграми.

Через діаграми можна представити будь-який образ. Таким чином ці схеми сформують універсальний формат, який дасть можливості відсутні у природної мови. Задачі можна представляти як діаграми, які схожі за своєю формою на модульні тести за методологією Test DD. Розробник зможе зациклитись на структурі й ідеях майбутнього ШІ. Діаграми можна буде переводити у числові моделі, які будуть виконуватися на GPU, що зробить їх дуже швидкими. Така система усуне більшість проблем і дасть поштовх для усієї галузі ШІ.

У ході функціонально-вартісного аналізу було виявлено, що варіант реалізації ПП із базовими блоками на основі коду має більший технічно-економічний рівень, ніж варіант із стандартною бібліотекою блоків.

На етапі концепції важко перевіряти продукт через тестові приклади. При порівнянні із існуючими фреймворками власна концепція також є дуже молодою і потребує удосконалення, що виходить за межі цієї роботи. У перспективі власний фреймворк зможе досягнути поставлених цілей, оскільки формується навколо прогресивніших ідей. Йому просто потрібно дати шанс і час на реалізацію.

Протягом виконання цієї роботи вдалося отримати такі досягнення:

- створено класифікацію за критеріями методологій розробки програмного забезпечення;
- створено гнучку модель порівняння методологій, що може виконуватися програмою;
- перевірено на основі тестових прикладів фреймворки розробки програмного забезпечення і оцінено їхню практичність;
- створено концепцію власного фреймворку розробки програмного забезпечення і виконано його функціонально-вартісний аналіз, як продукту.

Розглянуті теми у цій роботі є дуже актуальними, адже виникнення працюючих ШІ інструментів, які не мають аналогів за функціоналом, призведе до значних змін. Автоматизація розробки поширюється – це прискорить цифровий світ. Потрібно бути готовим до майбутнього, яке швидше за твої можливості. Ми мислимо лінійно і не розуміємо, як скоро відбудеться технологічна сингулярність. Можливо вона вже на порозі?

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. 12 кращих методологій розробки програмного забезпечення з перевагами та недоліками. Smart IT: веб-сайт. URL: <https://www.smart-it.com/uk/2021/08/12-best-software-development-methodologies-with-pros-and-cons/> (дата звернення: 06.05.2023).
2. 9 основних методологій розробки програмного забезпечення Agile. Agile Yakubovsky Blog : веб-сайт. URL: <https://agile.yakubovsky.com/ua/2015/10/9-osnovnykh-metodolohiy-rozrobky-prohramnoho-zabezpechennya-agile/> (дата звернення: 01.05.2023).
3. AgentGPT. AgentGPT, reworkd : веб-сайт. URL: <https://agentgpt.reworkd.ai/uk> (дата звернення: 12.05.2023).
4. Agile подход — что из себя представляет, как появился, плюсы и минусы. Apex-drive : веб-сайт. URL: <https://apex-drive.com/ru/blog/useful/agile-podhod> (дата звернення: 06.05.2023).
5. AI Code Generator - Amazon CodeWhisperer - AWS. Amazon : веб-сайт. URL: <https://aws.amazon.com/ru/codewhisperer/> (дата звернення: 12.05.2023).
6. Amazon зробив безкоштовним CodeWhisperer — помічник з кодування на основні штучного інтелекту. Hi-tech : веб-сайт. URL: <https://hi-tech.ua/amazon-zrobiv-bezkoshtovnim-codewhisperer-pomichnik-z-koduvannya-na-osnovni-shtuchnogo-intelektu/> (дата звернення: 10.05.2023).
7. Android Studio. Вікіпедія(ru) : веб-сайт. URL: https://ru.wikipedia.org/wiki/Android_Studio (дата звернення: 14.05.2023).
8. ChatGPT. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/ChatGPT> (дата звернення: 10.05.2023).
9. DOU Зарплати, Middle Software Engineer, Java, 2-4 роки. DOU.UA : вебсайт. URL: <https://jobs.dou.ua/salaries/?period=2022-12&position=Middle%20SE&technology=Java&experience=2-4> (дата звернення: 20.05.2023)
10. DOU Зарплати, Middle Software Engineer, Mobile, Kotlin, 2-4 роки. DOU.UA : вебсайт. URL: <https://jobs.dou.ua/salaries/?period=2022-12&position=Middle%20SE&domain=Mobile&technology=Kotlin&experience=2-4> (дата звернення: 20.05.2023)
11. DSDM. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/DSDM> (дата звернення: 02.05.2023).
12. Durable — AI платформа для генерації сайтів. KMRV DSGN : веб-сайт. URL: <https://www.komarov.design/durable-ai-platforma-dlia-ghienieratsiyi-saitiv/> (дата звернення: 10.05.2023).
13. Durable AI Website Builder and Small Business Software. Durable : веб-сайт. URL: <https://durable.co/> (дата звернення: 12.05.2023).
14. GitHub Copilot · Your AI pair programmer. GitHub : веб-сайт. URL: <https://github.com/features/copilot> (дата звернення: 12.05.2023).
15. GitHub Copilot. Вікіпедія(en): веб-сайт. URL: https://en.wikipedia.org/wiki/GitHub_Copilot (дата звернення: 10.05.2023).
16. GPT-4. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/GPT-4> (дата звернення: 10.05.2023).
17. IntelliJ IDEA. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/IntelliJ_IDEA (дата звернення: 14.05.2023).
18. Introducing ChatGPT – OpenAI. OpenAI : веб-сайт. URL: <https://openai.com/blog/chatgpt> (дата звернення: 12.05.2023).

19. Kotlin. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/Kotlin> (дата звернення: 14.05.2023).
20. MetaGPT. Metaidt : веб-сайт. URL: <https://www.metaidt.com/upgrade-your-account/> (дата звернення: 12.05.2023).
21. MetaGPT. TADVISER : веб-сайт. URL: <https://www.tadviser.ru/index.php/%D0%9F%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82:MetaGPT> (дата звернення: 10.05.2023).
22. MetaGPT/Pico Tour. YouTube : веб-сайт. URL: <https://www.youtube.com/watch?v=VpvGd21V0u0> (дата звернення: 10.05.2023).
23. Rational Unified Process. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/Rational_Unified_Process (дата звернення: 02.05.2023).
24. Spring Framework. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/Spring_Framework (дата звернення: 14.05.2023).
25. TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт. URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).
26. Test-driven development. Вікіпедія(en) : веб-сайт. URL: https://en.wikipedia.org/wiki/Test-driven_development (дата звернення: 03.05.2023).
27. V-Model. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/V-Model> (дата звернення: 01.05.2023).
28. Аналіз вимог. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3 (дата звернення: 07.05.2023).
29. Бережлива розробка програмного забезпечення. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D1%80%D0%B5%D0%B6%D0%BB%D0%B8%D0%B2%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 03.05.2023).
30. В.П. Пашін, В.В. Романов, Н.В. Єгорова. Методичні вказівки до виконання економіко-організаційного розділу дипломних проєктів (робіт) бакалаврів і спеціалістів для студентів Інституту прикладного системного аналізу : навчальне видання. Київ : НТУУ “КПІ”, 2011. – 118 с.
31. Водоспадна модель. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%92%D0%BE%D0%B4%D0%BE%D1%81%D0%BF%D0%B0%D0%B4%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C (дата звернення: 01.05.2023).
32. Гнучка розробка програмного забезпечення. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%93%D0%BD%D1%83%D1%87%D0%BA%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 02.05.2023).
33. Долар США – Українська гривня. Google Фінанси : вебсайт. URL: https://www.google.com/finance/quote/USD-UAH?sa=X&ved=2ahUKewjY-NiR4Zj_AhUGgosKHUT2CDMqY0JegQICBAc (дата звернення: 20.05.2023)

34. Ітеративна модель (iterative model). QALight : веб-сайт. URL: <https://qalight.ua/baza-znaniy/iterativna-model-iterative-model/> (дата звернення: 04.05.2023).
35. Ітеративна та інкрементна розробка. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%86%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0_%D1%82%D0%B0_%D1%96%D0%BD%D0%BA%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%BD%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0 (дата звернення: 02.05.2023).
36. Канбан (розробка). Вікіпедія(uk) : веб-сайт. URL: [https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B1%D0%B0%D0%BD_\(%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0\)](https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B1%D0%B0%D0%BD_(%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0)) (дата звернення: 03.05.2023).
37. Методологія розробки програмного забезпечення. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%B%D0%BE%D0%B3%D1%96%D1%8F_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 01.05.2023).
38. Про GitHub Copilot для фізичних осіб. GitHub : веб-сайт. URL: <https://docs.github.com/en/copilot/overview-of-github-copilot/about-github-copilot-for-individuals> (дата звернення: 10.05.2023).
39. Радельчук Г. І. Порівняльний аналіз сучасних методологій розробки програмного забезпечення. Сучасні інформаційні технології / 3. Програмне забезпечення. Хмельницький національний університет, Україна. URL: http://www.rusnauka.com/40_OINBG_2014/Informatica/3_182487.doc.htm (дата звернення: 06.05.2023).
40. Розміри ЄСВ-2023. Дебет-Кредит : вебсайт. URL: <https://services.dtki.ua/catalogues/indexes/13> (дата звернення: 20.05.2023)
41. Розробка програмного забезпечення. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення: 01.05.2023).
42. Скрам. Вікіпедія(uk) : веб-сайт. URL: <https://uk.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B0%D0%BC> (дата звернення: 03.05.2023).
43. Спіральна модель (spiral model). QALight : веб-сайт. URL: <https://qalight.ua/baza-znaniy/spiralna-model-spiral-model/> (дата звернення: 04.05.2023).
44. Спіральна модель. Вікіпедія(uk) : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D1%96%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C (дата звернення: 02.05.2023).
45. Трансформер (модель машинного навчання). Вікіпедія(uk) : веб-сайт. URL: [https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80_\(%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D1%80_(%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE)

[%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F\)](#) (дата звернення: 10.05.2023).

46. Функціональність. JetBrains : веб-сайт. URL: <https://www.jetbrains.com/ru-ru/idea/features/> (дата звернення: 14.05.2023).

47. Ціни на універсальні послуги, період дії ціни – травень 2023 року, для малих непобутових споживачів, що приєднані до мереж ПрАТ "ДТЕК Київські електромережі", із ПДВ. Yasno : вебсайт. URL: <https://yasno.com.ua/business/b2b-tariffs> (дата звернення: 21.05.2023)

48. Что такое канбан и чем он полезен? Worksection : веб-сайт. URL: <https://worksection.com/blog/kanban.html> (дата звернення: 07.05.2023).

49. Эта нейросеть ПУГАЕТ своим интеллектом! Прощай, ChatGPT... YouTube : веб-сайт. URL: <https://www.youtube.com/watch?v=wR9Baqk2xjQ&t=165s> (дата звернення: 10.05.2023).

ДОДАТКИ

1. Додаток А. patterns.txt
2. Додаток Б. Зразки коду.docx
3. Додаток В. rateTable.xlsx
4. Додаток Г. Результати перевірки фреймворків.docx
5. Додаток Д. Діаграма прецедентів UML власного фреймворку.pdf
6. Додаток Е. Діаграма класів UML власного фреймворку.pdf
7. Додаток Є. Діаграма станів UML циклу життя задачі.pdf
8. Додаток Ж. Малюнки, таблиці.docx

1. Додаток А. patterns.txt

\$Methodology names

&

Waterfall;

V-Model;

Iterative та Incremental;

Spiral;

DSDM;

RUP;

Agile;

Scrum;

Kanban;

Lean;

Test DD;

Type DD;

BDD;

DDD;

FDD;

MDD;

PDD;

ADD

&

\$Methodology paterns

&

якість | деталізація | ризики | контрольованість | процес розробки |
економія | час | команда | зворотній зв'язок | стійкість до змін;

якість | ризики | деталізація | економія | контрольованість | процес
розробки | команда | час | стійкість до змін | зворотній зв'язок;

стійкість до змін | час | контрольованість | процес розробки | ризики |
якість | команда | деталізація | економія | зворотній зв'язок;

ризики | процес розробки | стійкість до змін | контрольованість |
деталізація | час | команда | якість | економія | зворотній зв'язок;

час | економія | зворотній зв'язок | якість | деталізація |
контрольованість | процес розробки | ризики | команда | стійкість до
змін;

ризики | економія | деталізація | якість | процес розробки | команда |
контрольованість | час | стійкість до змін | зворотній зв'язок;

час | команда | стійкість до змін | процес розробки | зворотній зв'язок |
економія | якість | деталізація | контрольованість | ризики;

процес розробки | команда | час | зворотній зв'язок | стійкість до змін |
якість | деталізація | економія | ризики | контрольованість;

час | економія | команда | стійкість до змін | якість | деталізація |
процес розробки | ризики | контрольованість | зворотній зв'язок;

економія | якість | ризики | час | команда | зворотній зв'язок |
контрольованість | процес розробки | деталізація | стійкість до змін;

якість | час | стійкість до змін | деталізація | економія | зворотній
зв'язок | ризики | контрольованість | процес розробки | команда;

якість | економія | деталізація | стійкість до змін | час | команда |
процес розробки | контрольованість | ризики | зворотній зв'язок;

якість | час | стійкість до змін | деталізація | економія | зворотній
зв'язок | ризики | контрольованість | процес розробки | команда;

час | економія | команда | контрольованість | стійкість до змін | процес
розробки | деталізація | якість | ризики | зворотній зв'язок;

процес розробки | час | деталізація | якість | контрольованість |
стійкість до змін | команда | ризики | економія | зворотній зв'язок;

час | процес розробки | економія | контрольованість | стійкість до змін |
якість | команда | ризики | деталізація | зворотній зв'язок;

команда | процес розробки | зворотній зв'язок | деталізація | стійкість
до змін | контрольованість | економія | час | ризики | якість;

час | якість | стійкість до змін | економія | контрольованість | ризики |
зворотній зв'язок | деталізація | процес розробки | команда

&

\$Methodology classification

&

Waterfall | маленька команда | велика команда | малий
проект | великий проект | короткостроковий проект | довгостроковий проект
| чіткі вимоги | без змін | високі вимоги до якості;

V-Model | велика команда | великий проект | короткостроковий проект | чіткі вимоги | без змін | високі вимоги до якості;

Iterative та Incremental | велика команда | малий проект | великий проект | короткостроковий проект | довгостроковий проект | не визначені вимоги | постійні зміни ;

Spiral | велика команда | великий проект | довгостроковий проект | не визначені вимоги | постійні зміни | високі вимоги до якості;

DSDM | велика команда | малий проект | великий проект | короткостроковий проект | чіткі вимоги | постійні зміни | обмежені ресурси;

RUP | велика команда | великий проект | довгостроковий проект | чіткі вимоги | постійні зміни | високі вимоги до якості;

Agile | маленька команда | малий проект | великий проект | короткостроковий проект | довгостроковий проект | не визначені вимоги | постійні зміни ;

Scrum | маленька команда | малий проект | великий проект | короткостроковий проект | довгостроковий проект | не визначені вимоги | постійні зміни ;

Kanban | маленька команда | малий проект | великий проект | короткостроковий проект | не визначені вимоги | постійні зміни | обмежені ресурси;

Lean | велика команда | малий проект | великий проект | короткостроковий проект | довгостроковий проект | чіткі вимоги | постійні зміни | обмежені ресурси | високі вимоги до якості;

Test DD | маленька команда | велика команда | малий проект | великий проект | короткостроковий проект | довгостроковий проект | не визначені вимоги | постійні зміни | високі вимоги до якості;

Type DD | маленька команда | велика команда | малий проект | великий проект | короткостроковий проект | довгостроковий проект | чіткі вимоги | постійні зміни | високі вимоги до якості;

BDD | маленька команда | велика команда | малий проект | великий проект | короткостроковий проект | довгостроковий проект | не визначені вимоги | постійні зміни | високі вимоги до якості;

DDD | маленька команда | велика команда | малий проект | великий проект | короткостроковий проект | довгостроковий проект | чіткі вимоги | постійні зміни ;

FDD | велика команда | малий
проект | великий проект | короткостроковий проект | довгостроковий проект
| чіткі вимоги | постійні зміни | високі вимоги до якості;

MDD | маленька команда | малий
проект | великий проект | короткостроковий проект | довгостроковий проект
| не визначені вимоги | постійні зміни ;

PDD | маленька команда | малий
проект | короткостроковий проект
| чіткі вимоги | без змін ;

ADD | маленька команда | малий
проект | великий проект | короткостроковий проект | довгостроковий проект
| чіткі вимоги | постійні зміни | високі вимоги до якості

&

\$Scenario patters

&

команда | час | контрольованість | ризики | зворотній зв'язок | процес
розробки | деталізація | якість | економія | стійкість до змін;

команда | деталізація | економія | ризики | час | процес розробки |
якість | стійкість до змін | контрольованість | зворотній зв'язок;

якість | команда | процес розробки | стійкість до змін | ризики |
зворотній зв'язок | час | економія | деталізація | контрольованість;

процес розробки | контрольованість | ризики | економія | час |
деталізація | якість | команда | стійкість до змін | зворотній зв'язок;

якість | ризики | економія | стійкість до змін | час | контрольованість |
деталізація | процес розробки | команда | зворотній зв'язок;

процес розробки | час | економія | якість | контрольованість | ризики |
стійкість до змін | деталізація | команда | зворотній зв'язок;

якість | ризики | деталізація | час | процес розробки | зворотній зв'язок
| команда | контрольованість | стійкість до змін | економія;

якість | контрольованість | деталізація | ризики | зворотній зв'язок |
час | процес розробки | команда | стійкість до змін | економія

&

\$Scenario classification

&

маленька команда | малий проект | короткостроковий проект | чіткі
вимоги | постійні зміни | обмежені ресурси;

маленька команда	малий проект	короткостроковий проект	не визначені вимоги	постійні зміни	обмежені ресурси;	
маленька команда	великий проект	довгостроковий проект	чіткі вимоги	постійні зміни	високі вимоги до якості;	
маленька команда	великий проект	довгостроковий проект	чіткі вимоги	без змін;		
велика команда	великий проект	короткостроковий проект	не визначені вимоги	постійні зміни	обмежені ресурси	високі вимоги до якості;
велика команда	великий проект	довгостроковий проект	чіткі вимоги	постійні зміни	високі вимоги до якості;	
велика команда	великий проект	короткостроковий проект	чіткі вимоги	постійні зміни	високі вимоги до якості;	
велика команда	великий проект	довгостроковий проект	чіткі вимоги	без змін	високі вимоги до якості	

2. Додаток Б. Зразки коду.docx

1) rateEvaluation.py.

```
import re
from itertools import combinations

def readAllFile(path):
    file = open(path, "r")
    fileText = file.read()
    file.close()
    return fileText

def formatDataFromText(fileText):
    blocks = {}
    partStrList = fileText.split("&")
    blockNames = []
    blocksStrList = []
    for part in partStrList:
        p = part.strip()
        if p[0] != "$":
            blocksStrList.append(p)
        else:
            blockNames.append(p[1:])

    for i in range(len(blockNames)):
```



```

        blockStr = blocksStrList[i]
        blockList = blockStr.split(";")
        for j in range(len(blockList)):
            blockList[j] = re.sub(r"[\s]", "", blockList[j]).split("|")
        blocks[blockNames[i]] = blockList

    return blocks

def getAllSubsequences(lst):
    subsequences = []
    for i in range(2, len(lst)+1):
        subsequences += combinations(lst,i)

    return subsequences

def calculatePatternConvergence(pattern1, pattern2):
    set1 = set(getAllSubsequences(pattern1))
    set2 = set(getAllSubsequences(pattern2))
    intersection = set1.intersection(set2)
    if len(intersection) == 0:
        longest = 0
    else:
        longest = len(sorted(intersection, key=len)[-1])
    return longest

def calculateFeatureIntersectionPower(pattern1, pattern2):
    return len(set(pattern1).intersection(set(pattern2)))

def calculateRate(patternConvergence, featureIntersectionPower):
    ratesDiscreteDistribution = [1,1,1,2,2,3,4,4,5,5,5]
    PC = patternConvergence
    FIP = featureIntersectionPower
    discreteDistributionPosition = round((PC * (10/7)*FIP)**0.5)
    DDP = discreteDistributionPosition
    return ratesDiscreteDistribution[DDP]

def prepareDataForTable(blocks):
    methodologyNames = blocks["Methodology names"]
    methodologyPatterns = blocks["Methodology patterns"]
    methodologyClassification = blocks["Methodology classification"]
    scenarioPatterns = blocks["Scenario patterns"]
    scenarioClassification = blocks["Scenario classification"]

    rateMap = dict()
    for i in range(len(methodologyPatterns)):
        rateRow = []

```

```

        for j in range(len(scenarioPatters)):
            convergence =
calculatePatternConvergence(methodologyPaterns[i], scenarioPatters[j])
            intersectionPower =
calculateFeatureIntersectionPower(methodologyClassification[i],
scenarioClassification[j])
            rateRow.append(calculateRate(convergence, intersectionPower))

        rateMap[methodologyNames[i][0]] = rateRow

    return rateMap

def printTable(dataMap):
    keys = list(dataMap.keys())
    for k in keys:
        print(k + " " + str(dataMap[k]))
    print()
    for k in keys:
        lst = dataMap[k]
        for elm in lst:
            print(elm, end=" ")

        print()

def processFile(path):
    fileText = readAllFile(path)
    blocks = formatDataFromText(fileText)
    rateMap = prepareDataForTable(blocks)
    printTable(rateMap)

if __name__ == '__main__':
    processFile("C:\\Users\\vadim\\Desktop\\patterns.txt")

```

2) Тестовий приклад 4.

```

def printTable(dataMap):
    keys = list(dataMap.keys())
    for k in keys:
        print(k + " " + str(dataMap[k]))
    print()
    for k in keys:
        lst = dataMap[k]
        for elm in lst:
            print(elm, end=" ")

```

```

        print()

def formatDataFromText(fileText):
    blocks = {}
    partStrList = fileText.split("&")
    blockNames = []
    blocksStrList = []
    for part in partStrList:
        p = part.strip()
        if p[0] != "$":
            blocksStrList.append(p)
        else:
            blockNames.append(p[1:])

    for i in range(len(blockNames)):
        blockStr = blocksStrList[i]
        blockList = blockStr.split(";")
        for j in range(len(blockList)):
            blockList[j] = re.sub(r"[\s]", "", blockList[j]).split("|")
        blocks[blockNames[i]] = blockList

    return blocks

```

3) Тестовий приклад 5.

```

import java.io.BufferedReader;
import java.io.FileReader;
import weka.classifiers.Evaluation;
import weka.classifiers.functions.GaussianProcesses;
import weka.core.Instances;

void wekaAPIRealization(String source){
    try {
        BufferedReader bufferedReader = new BufferedReader(new
FileReader(source));
        Instances datasetInstances = new Instances(bufferedReader);
        datasetInstances.randomize(new java.util.Random(0));
        int trainingDataSize = (int)
Math.round(datasetInstances.numInstances() * 0.66);
        int testDataSize = datasetInstances.numInstances() -
trainingDataSize;
        Instances trainingInstances = new
Instances(datasetInstances,0,trainingDataSize);
        Instances testInstances = new Instances(datasetInstances,
trainingDataSize,testDataSize);

```

```

trainingInstances.setClassIndex(trainingInstances.numAttributes()-
1);
    testInstances.setClassIndex(testInstances.numAttributes()-1);
    bufferedReader.close();
    var gaussianProcesses = new GaussianProcesses();
    gaussianProcesses.buildClassifier(trainingInstances);
    Evaluation trainEvaluation = new Evaluation(trainingInstances);

trainEvaluation.evaluateModel(gaussianProcesses,trainingInstances);
    Evaluation testEvaluation = new Evaluation(testInstances);

    testEvaluation.evaluateModel(gaussianProcesses,testInstances);
    System.out.println("Weka regression model.");
    System.out.println(trainEvaluation.toSummaryString(
        "\nResults on train data:",false));
    System.out.println(testEvaluation.toSummaryString(
        "\nResults on test data:",false));
}
catch (Exception e) {
    System.out.println("Error Occurred!!!! \n" + e.getMessage());
}
}

```

4) Тестовий приклад 6.

```

TEST(createPalindrome, createPalindrome_Test1) {
    EXPECT_EQ(createPalindrome(5, "abc"), "abcba");
    EXPECT_EQ(createPalindrome(8, "rTy"), "rTyrryTr");
    EXPECT_EQ(createPalindrome(3, "jj"), "jjj");
    EXPECT_EQ(createPalindrome(1, "125"), "1");
    EXPECT_EQ(createPalindrome(4, "q"), "qqqq");
    EXPECT_EQ(createPalindrome(5, ""), "");
    EXPECT_EQ(createPalindrome(2, " "), " ");
    EXPECT_EQ(createPalindrome(0, ""), "");
    EXPECT_EQ(createPalindrome(0, "jd"), "");
    EXPECT_EQ(createPalindrome(-3, "vkx"), "");
}

```

5) Тестовий приклад 7.

```

#include <string>
#include <vector>
#include <fstream>
#include <mutex>
#include <iostream>

```

```

#include <sstream>
#include <array>
#include <windows.h>

struct HashItem {
    std::wstring key;
    std::vector<std::wstring> value;
    long long byteSize = 0;
    long long next = -1;
};

class HashTable {
public:
    const static std::wstring delimiter;
    const static size_t cellSize;

    HashTable(long long size);
    HashTable(const HashTable& other);
    HashTable& operator=(const HashTable& other);
    int add(std::wstring key, std::vector<std::wstring> value);
    std::vector<std::wstring> get(std::wstring key);
    int increaseTable();
    void setBusyThreadCount(int busyThreadCount);
    void saveTable(std::wstring path);
    static long long hashFunction(std::wstring key, long long size);
    long long getCapacity();
    long long getCount();
    long long getSize();
    double getPercent();

private:
    const double percent = 0.75;
    const long long maxCapacity = 2000000000;
    const long long itemsSizeOnItemsMutexRatio = 32;
    std::atomic<int> busyThreadCount = 0;
    std::atomic<long long> size;
    std::atomic<long long> count = 0;
    std::atomic<long long> currentCapacity = 0;
    std::atomic<long long> itemsMutexesSize = 0;
    std::atomic<bool> increaseInProgress = false;
    std::atomic<unsigned int> currentStopedThreadCount = 0;
    std::vector<HashItem> items;
    std::vector<HashItem> colisions;
    std::vector<std::mutex> itemsMutexes;
    std::mutex colisionsMutex;

```

```

std::mutex sizeMutex;
std::mutex countMutex;
std::mutex currentCapacityMutex;
std::mutex increaseInProgressMutex;

std::vector<HashItem> createNewVector(long long s);
void handleCollision(HashItem item, long long colisionIndex);
long long getItemCapacity(HashItem item);
std::array<long long, 4> itemToArray(HashItem item, size_t cellSize,
long long valueAddress);
std::wstring writeVectorToString(std::vector<std::wstring> vector);
long long getMutexIndex(long long itemIndex);
void checkIncreaseProcess();
};

```

6) Тестовий приклад 8.

```

#include <vector>
#include <thread>
#include <queue>
#include <mutex>
#include <condition_variable>
#include <functional>
#include <future>
#include "Windows.h"

template <typename T>
class ThreadPool {
public:
    std::mutex mfinish;
    std::atomic<bool> finish = false;

    ThreadPool<T>() {}
    ThreadPool<T>(std::function<int(T)> _taskHandler, int threadCount);
    ThreadPool<T>(const ThreadPool<T>& other);
    ThreadPool<T>& operator=(const ThreadPool<T>& other);
    void addTask(T task);
    T removeTask();
    void shutdown(bool evenNonEmptyQueue, long waitDuration);
    void initThreads(int newThreadCount);
    void setTaskHandler(std::function<int(T)> _taskHandler);
    bool joinAll();
    int getBusyThreadCount();

private:

```

```

int threadCount;
std::atomic<int> busyThreadCount = 0;
std::vector<std::thread> threads;
std::queue<T> tasks;
std::mutex taskQueueMutex;
std::mutex lkm;
std::condition_variable task_added;
std::mutex mtskHndl;
std::mutex busyThreadCountMutex;
std::function<int(T)> taskHandler;

void workFunction(std::function<int(T)>& taskHandler);
};

```

7) Тестовий приклад 9.

```

import org.ejml.simple.SimpleMatrix;
import java.util.ArrayList;

class HopfieldNetwork {
    private static final int MAX_ITERATIONS = 50;
    private SimpleMatrix W = null;
    private ArrayList<ArrayList<Integer>> learned;

    HopfieldNetwork(){learned = new ArrayList<>();}

    public void train(ArrayList<Integer> set){
        learned.add(set);
        double[] d = set.stream().mapToDouble(i -> (double) i).toArray();
        SimpleMatrix m = new SimpleMatrix(1,set.size(),true,d);
        if (learned.size() == 1) W = m.transpose().mult(m);
        else W = W.plus(m.transpose().mult(m));
        for(int i = 0; i < W.numRows(); i++)
            W.set(i, i, 0);
    }

    public ArrayList<Integer> test(ArrayList<Integer> set){
        double[] d = set.stream().mapToDouble(i -> (double) i).toArray();
        SimpleMatrix input = new SimpleMatrix(set.size(),1,true,d);
        SimpleMatrix Y = normalize(W.mult(input));
        int i = 0;
        while (i < MAX_ITERATIONS){
            Y = normalize(W.mult(Y));i++;
        }
    }
}

```

```

        ArrayList<Integer> result = new ArrayList<>();
        for(i = 0; i < Y.numRows(); i++)
            for(int j = 0; j < Y.numCols(); j++)
                result.add((int)Y.get(i, j));
        return result;
    }

    private SimpleMatrix normalize(SimpleMatrix m){
        double temp;
        for(int i = 0; i < m.numRows(); i++)
            for(int j = 0; j < m.numCols(); j++){
                temp = m.get(i,j);
                if (temp >= 0)
                    m.set(i, j, 1);
                else
                    m.set(i, j, -1);
            }
        return m;
    }

    public ArrayList<Integer> getInstance(int index){
        return learned.get(index);
    }

    public int sampleSize(){
        return learned.size();
    }
}

```


3. Додаток В. rateTable.xlsx

	A	B	C	D	E	F	G	H	I	J	M
1		Методологія \ Сценарій	1	2	3	4	5	6	7	8	Всього
2	18	ADD									30
3	12	Type DD									28
4	10	Lean									28
5	1	Waterfall									27
6	14	DDD									26
7	15	FDD									26
8	2	V-Model									24
9	6	RUP									24
10	13	BDD									23
11	4	Spiral									23
12	11	Test DD									23
13	5	DSDM									23
14	16	MDD									22
15	3	Iterative та Incremental									22
16	7	Agile									20
17	9	Kanban									20
18	8	Scrum									20
19	17	PDD									17

Рис В.1. Вміст файлу rateTable.xlsx.

4. Додаток Г. Результати перевірки фреймворків.

ChatGPT v3.0

1. **Користувач:** Створити рекурсивний алгоритм для пошуку в ширину у дереві на Python та C++.

ChatGPT: Ось приклад рекурсивного алгоритму для пошуку в ширину (BFS) у дереві на Python:

```
python
class Node:
    def __init__(self, value):
        self.value = value
        self.children = []
```

```

def breadth_first_search(root):
    queue = []
    queue.append(root)
    while queue:
        node = queue.pop(0)
        print(node.value)
        queue.extend(node.children)

    for child in node.children:
        breadth_first_search(child)
    ...

```

У цьому прикладі `Node` представляє вузол у дереві, а `breadth_first_search` є рекурсивною функцією для пошуку в ширину. Вона починає з кореневого вузла і додає його до черги. Потім вона видаляє перший елемент з черги, виводить його значення і додає всіх його дітей до черги. На кожній ітерації функція викликає саму себе для кожного дитя вузла, щоб рекурсивно обробити всі вузли у дереві.

Ось приклад рекурсивного алгоритму для пошуку в ширину у дереві на C++:

```

```cpp
#include <iostream>
#include <queue>
using namespace std;

struct Node {
 int value;
 vector<Node*> children;
};

void breadthFirstSearch(Node* root) {
 queue<Node*> q;
 q.push(root);
 while (!q.empty()) {
 Node* node = q.front();
 q.pop();
 cout << node->value << endl;
 for (Node* child : node->children) {
 q.push(child);
 }
 for (Node* child : node->children) {
 breadthFirstSearch(child);
 }
 }
}
...

```

У цьому прикладі `Node` представляє вузол у дереві, а `breadthFirstSearch` є рекурсивною функцією для пошуку в ширину. Вона починає з кореневого вузла і додає його до черги. Потім вона видаляє перший елемент з черги, виводить його значення і додає всіх його дітей до черги. На кожній ітерації функція викликає саму себе для кожного дитя вузла, щоб рекурсивно обробити всі вузли у дереві.

2. **Користувач:** Створити вебсайт для пошуку оптимальних ходів у грі шахи. Користувач розташовує фігури на дошці, а програма має врахувати рейтинг оптимальних ходів і показати критерії їхнього підбору у вигляді однієї таблиці, що розташовується нижче дошки.

**ChatGPT:** Створення вебсайту для пошуку оптимальних ходів у грі шахи може бути цікавим проектом! Для реалізації цієї ідеї вам знадобиться знання веб-розробки, включаючи мови програмування для вебу, такі як HTML, CSS і JavaScript, а також навички обробки шахових логічних правил і алгоритмів пошуку оптимальних ходів.

Ось загальний опис кроків, які можна виконати для створення такого вебсайту:

1. **Проектування інтерфейсу:** Спочатку вам потрібно спроектувати інтерфейс веб-сторінки для гри в шахи. Ви можете використовувати HTML і CSS для створення дошки і фігур на ній. Додайте можливість користувачеві розташовувати фігури на дошці за допомогою перетягування або кліків.

2. **Логіка гри:** Реалізуйте логіку гри шахи на веб-сторінці за допомогою JavaScript. Створіть об'єкти для кожної фігури, включаючи їхні можливі ходи і логічні правила. Забезпечте можливість користувачеві перетягувати фігури і оновлювати стан дошки.

3. **Алгоритм пошуку оптимальних ходів:** Реалізуйте алгоритм пошуку оптимальних ходів, який оцінює поточний стан дошки і розраховує рейтинг для кожного можливого ходу. Існує багато алгоритмів, які можна використовувати для цієї мети, наприклад, алгоритм Мінімакс або алгоритм Alpha-Beta pruning. Виберіть підходящий алгоритм і реалізуйте його в JavaScript.

4. **Відображення результатів:** Після обчислення рейтингів для можливих ходів, відобразіть їх у вигляді таблиці нижче дошки.

**Користувач:** Спроектуй інтерфейс веб-сторінки для гри в шахи. Ви можете використовувати HTML і CSS для створення дошки і фігур на ній. Додайте можливість користувачеві розташовувати фігури на дошці за допомогою перетягування або кліків.

**ChatGPT:** Звичайно! Ось приклад коду HTML і CSS для створення базового інтерфейсу веб-сторінки для гри в шахи з можливістю перетягування фігур на дошці:

```
```html
<!DOCTYPE html>
<html>
<head>
  <title>Гра в шахи</title>
  <style>
    .chessboard {
      width: 400px;
```

```

    height: 400px;
    border: 1px solid #000;
    display: flex;
    flex-wrap: wrap;
  }

  .square {
    width: 50px;
    height: 50px;
    box-sizing: border-box;
    border: 1px solid #000;
  }

  .piece {
    width: 100%;
    height: 100%;
    background-size: contain;
    background-repeat: no-repeat;
    background-position: center;
  }
</style>
</head>
<body>
  <div class="chessboard">
    <!-- Рядки і стовпці дошки -->
    <!-- Кожен квадрат представляє собою клітинку на дошці -->
    <!-- Додайте класи square і відповідний ID для кожної клітинки -->
    <!-- Наприклад, клітинка a1 має ID "a1" -->
    <div class="square" id="a1"></div>
    <div class="square" id="b1"></div>
    <div class="square" id="c1"></div>
    <div class="square" id="d1"></div>
    <div class="square" id="e1"></div>
    <div class="square" id="f1"></div>
    <div class="square" id="g1"></div>
    <div class="square" id="h1"></div>

    <!-- Додайте інші рядки та стовпці за аналогією -->

  </div>

  <script>
    // Додайте JavaScript-код для обробки перетягування фігур та оновлення стану
    дошки
  </script>
</body>
</html>

```

...

У цьому прикладі створюється дошка шахів розміром 400x400 пікселів з використанням елементів ``<div>``. Кожен квадрат представляє клітинку на дошці і має свій унікальний ідентифікатор (`id`). Ви можете додати решту рядків і стовпців дошки, повторюючи блок ``<div class="square">``.

Тепер вам потрібно додати JavaScript-код для обробки перетягування фігур та оновлення стану дошки. Ви можете скористатися різними бібліотеками, таким

Користувач: Додайте можливість користувачеві розташовувати фігури на дошці за допомогою перетягування або кліків.

ChatGPT: Для додавання можливості перетягування фігур на дошці ви можете використовувати бібліотеку Drag and Drop HTML5 або використовувати сторонні бібліотеки, такі як jQuery UI Draggable.

Ось приклад використання бібліотеки jQuery UI Draggable для додавання функціональності перетягування фігур на дошці:

```
```html
<!DOCTYPE html>
<html>
<head>
 <title>Гра в шахи</title>
 <style>
 .chessboard {
 width: 400px;
 height: 400px;
 border: 1px solid #000;
 display: flex;
 flex-wrap: wrap;
 }

 .square {
 width: 50px;
 height: 50px;
 box-sizing: border-box;
 border: 1px solid #000;
 }

 .piece {
 width: 100%;
 height: 100%;
 background-size: contain;
 background-repeat: no-repeat;
 background-position: center;
 }
 </style>
</head>
</html>
```
```

```

    }
  </style>
  <link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
</head>
<body>
  <div class="chessboard">
    <!-- Рядки і стовпці дошки -->
    <!-- Кожен квадрат представляє собою клітинку на дошці -->
    <!-- Додайте класи square і відповідний ID для кожної клітинки -->
    <!-- Наприклад, клітинка a1 має ID "a1" -->
    <div class="square" id="a1"></div>
    <div class="square" id="b1"></div>
    <div class="square" id="c1"></div>
    <div class="square" id="d1"></div>
    <div class="square" id="e1"></div>
    <div class="square" id="f1"></div>
    <div class="square" id="g1"></div>
    <div class="square" id="h1"></div>

    <!-- Додайте інші рядки та стовпці за аналогією -->

  </div>

  <script>
    $(document).ready(function() {
      $(".piece").draggable(); // Зробити фігури перетягуваними
    });
  </script>
</body>
</html>
```

```

У цьому прикладі підключена бібліотека jQuery та jQuery UI. Фігури з класом `piece` зроблені перетягуваними за допомогою функції `draggable()`.

Ви можете розмістити ф

**Користувач:** Реалізуйте логіку гри шахи на веб-сторінці за допомогою JavaScript. Створіть об'єкти для кожної фігури, включаючи їхні можливі ходи і логічні правила. Забезпечте можливість користувачеві перетягувати фігури і оновлювати стан дошки.

**ChatGPT:** Оскільки шахи мають багато правил і можливостей, реалізація повноцінної логіки гри на веб-сторінці може бути складною задачею. Однак, я можу показати приклад коду JavaScript, який дозволяє користувачеві перетягувати фігури та оновлювати стан дошки.

```

```html
<!DOCTYPE html>
<html>
<head>
  <title>Гра в шахи</title>
  <style>
    .chessboard {
      width: 400px;
      height: 400px;
      border: 1px solid #000;
      display: flex;
      flex-wrap: wrap;
    }

    .square {
      width: 50px;
      height: 50px;
      box-sizing: border-box;
      border: 1px solid #000;
    }

    .piece {
      width: 100%;
      height: 100%;
      background-size: contain;
      background-repeat: no-repeat;
      background-position: center;
    }
  </style>
  <link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
</head>
<body>
  <div class="chessboard">
    <!-- Рядки і стовпці дошки -->
    <!-- Кожен квадрат представляє собою клітинку на дошці -->
    <!-- Додайте класи square і відповідний ID для кожної клітинки -->
    <!-- Наприклад, клітинка a1 має ID "a1" -->
    <div class="square" id="a1"></div>
    <div class="square" id="b1"></div>
    <div class="square" id="c1"></div>
    <div class="square" id="d1"></div>
    <div class="square" id="e1"></div>
    <div class="square" id="f1"></div>
    <div class="square" id="g1"></div>
    <div class="square" id="h1"></div>
  </div>
</body>
</html>

```

```

<!-- Додайте інші рядки та стовпці за аналогією -->

</div>

<script>
$(document).ready(function() {
    $(".piece").draggable(); // Зробити фігури перетягуваними

    $(".square").droppable({
        drop: function(event, ui) {
            var piece = ui.draggable; // Перетягувана фігура
            var square = $(this); // Клітинка, на яку фігура була перетягнута

            // Оновлення стану дошки
            var previousSquare = piece.parent();
            previousSquare.empty(); // Видалити п

```

Користувач: Алгоритм пошуку оптимальних ходів: Реалізуйте алгоритм пошуку оптимальних ходів, який оцінює поточний стан дошки і розраховує рейтинг для кожного можливого ходу. Існує багато алгоритмів, які можна використовувати для цієї мети, наприклад, алгоритм Мінімакс або алгоритм Alpha-Beta pruning. Виберіть підходящий алгоритм і реалізуйте його в JavaScript.

ChatGPT: Реалізація повноцінного алгоритму пошуку оптимальних ходів у шахах, таких як Мінімакс або Alpha-Beta pruning, може бути складною задачею, оскільки вона включає оцінку поточного стану дошки і розрахунок рейтингу для кожного можливого ходу.

Нижче наведено приклад коду, який демонструє просту реалізацію алгоритму Мінімакс для пошуку оптимального ходу в шаховій грі. Зауважте, що цей код просто ілюструє основну ідею алгоритму і не включає всі деталі та оптимізації, які можуть бути важливими для повноцінного рішення.

```

```javascript
// Оцінка поточного стану дошки
function evaluateBoard(board) {
 // Реалізуйте логіку оцінки стану дошки і поверніть рейтинг
 // Чим вище рейтинг, тим кращий стан дошки для гравця
}

// Функція Мінімакс для пошуку оптимального ходу
function minimax(board, depth, maximizingPlayer) {
 if (depth === 0) {
 return evaluateBoard(board);
 }

 if (maximizingPlayer) {

```



```

var maxEval = -Infinity;

// Перебір всіх можливих ходів
var possibleMoves = generateMoves(board);
for (var i = 0; i < possibleMoves.length; i++) {
 var move = possibleMoves[i];
 var newBoard = makeMove(board, move);
 var eval = minimax(newBoard, depth - 1, false);
 maxEval = Math.max(maxEval, eval);
}

return maxEval;
} else {
 var minEval = Infinity;

 // Перебір всіх можливих ходів
 var possibleMoves = generateMoves(board);
 for (var i = 0; i < possibleMoves.length; i++) {
 var move = possibleMoves[i];
 var newBoard = makeMove(board, move);
 var eval = minimax(newBoard, depth - 1, true);
 minEval = Math.min(minEval, eval);
 }

 return minEval;
}
}

// Функція для знаходження оптимального ходу
function findBestMove(board, depth) {
 var bestEval = -Infinity;
 var bestMove = null;

 // Перебір всіх можливих ходів
 var possibleMoves = generateMoves(board);
 for (var i = 0; i < possibleMoves.length; i++) {
 var move = possibleMoves[i];

 var newBoard = makeMove(board, move);
 var eval = minimax(newBoard, depth - 1, false);

 if (eval > bestEval) {
 bestEval = eval;
 bestMove = move;
 }
 }
}

```

```

 return bestMove;
}

// Приклад використання
var board = initializeBoard();
var bestMove = findBestMove(board, 3);
console.log("Optimal Move:", bestMove);
```

```

Цей код включає функцію `evaluateBoard`, яка оцінює поточний стан дошки, функцію `minimax`, яка рекурсивно викликає себе для пошуку оптимального ходу, і функцію `findBestMove`, яка викликає `minimax` для всіх можливих ходів і повертає найкращий з них.

Ви можете використати цей код як основу і додати його до свого веб-сайту для реалізації алгоритму пошуку оптимальних ходів у шахах. Проте, слід зазначити, що цей код не включає логіку фігур, правила гри та реалізацію користувацького інтерфейсу, які також потрібні для повноцінної гри в шахи.

3. **Користувач:** Створити вебсайт для компанії, яка продає дрони інших виробників. Сайт повинен мати таку структуру та вміст на одну сторінку:

- Header сайту, де є логотип компанії, меню для вибору розділів: «про нас», «новини», «наші партнери», «контакти»;
- Реклама у вигляді зображення із текстом по центру;
- Розділи, що показують лінійку товарів та сервісних послуг;
- Розділи що описують компанію, її досягнення;
- Footer сайту із контактами, логотипом, сервісними розділами.

ChatGPT: Щоб створити вебсайт для компанії, яка продає дрони інших виробників і має вказану структуру та вміст на одній сторінці, ви можете скористатись HTML та CSS для розмітки і стилізації сайту. Ось приклад коду, який можна використати для створення такого сайту:

```

```html
<!DOCTYPE html>
<html>
<head>
 <title>Компанія дронів</title>
 <style>
 /* CSS стилі для сайту */
 body {
 font-family: Arial, sans-serif;
 margin: 0;
 padding: 0;
 }
 </style>
</head>
<body>
 <div class="header">

 Про нас
 Новини
 Наші партнери
 Контакти

 </div>
 <div class="ad">

 </div>
 <div class="products">

 Товар 1
 Товар 2
 Товар 3

 </div>
 <div class="about">
 <h3>Про компанію</h3>
 <p>Наші досягнення</p>
 </div>
 <div class="footer">

 Контакти
 Сервіс

 </div>
</body>
</html>
```

```

```

header {
  background: #f8f8f8;
  padding: 20px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

nav ul {
  list-style: none;
  display: flex;
  margin: 0;
  padding: 0;
}

nav ul li {
  margin-left: 20px;
}

section {
  padding: 40px;
}

footer {
  background: #333;
  color: #fff;
  padding: 20px;
  text-align: center;
}
</style>
</head>
<body>
<header>
  <div>
    
  </div>
  <nav>
    <ul>
      <li><a href="#about">Про нас</a></li>
      <li><a href="#news">Новини</a></li>
      <li><a href="#partners">Наші партнери</a></li>
      <li><a href="#contact">Контакти</a></li>
    </ul>
  </nav>
</header>

<section>

```

```

    
</section>

<section id="products">
  <h2>Наша лінійка товарів</h2>
  <!-- Додайте тут розділи з дронами та їх характеристиками -->
</section>

<section id="services">
  <h2>Наші сервісні послуги</h2>
  <!-- Додайте тут розділи з описом сервісів -->
</section>

<section id="about">
  <h2>Про нас</h2>
  <!-- Додайте тут розділ з описом компанії та її досягнень -->
</section>

<footer>
  <div>
    
  </div>
  <div>
    <ul>
      <li><a href="#services">Сервіси</a></li>
      <li><a href="#news">Новини</a></li>
      <li><a href="#contact">Конт

```

Користувач: Створи для цього сайту наповнення, поведінку, щоб у кожній секції була анімація

ChatGPT: Зважаючи на те, що потрібно багато додаткового коду та ресурсів для створення повноцінної анімації, я надам вам базовий приклад анімації для кожної секції. Ви можете вдосконалити цей код та додати додаткові анімаційні ефекти за допомогою CSS або JavaScript. Ось приклад, як ви можете додати анімацію до кожної секції:

```

```html
<!DOCTYPE html>
<html>
<head>
 <title>Компанія дронів</title>
 <style>
 /* CSS стилі для сайту */
 body {
 font-family: Arial, sans-serif;
 margin: 0;
 padding: 0;

```

```

}

header {
 background: #f8f8f8;
 padding: 20px;
 display: flex;
 justify-content: space-between;
 align-items: center;
}

nav ul {
 list-style: none;
 display: flex;
 margin: 0;
 padding: 0;
}

nav ul li {
 margin-left: 20px;
}

section {
 padding: 40px;
 opacity: 0; /* Початково секція невидима */
 transform: translateY(20px); /* Початкове зсування секції вниз */
 transition: opacity 0.5s ease, transform 0.5s ease; /* Плавний перехід для анімації */
}

section.visible {
 opacity: 1; /* Показувати секцію */
 transform: translateY(0); /* Відмінити зсування секції */
}

footer {
 background: #333;
 color: #fff;
 padding: 20px;
 text-align: center;
}
</style>

<script>
// JavaScript код для анімації
window.addEventListener('scroll', animateSections);

function animateSections() {
 var sections = document.querySelectorAll('section');

```

```

for (var i = 0; i < sections.length; i++) {
 var section = sections[i];
 var sectionTop = section.getBoundingClientRect().top;
 var windowHeight = window.innerHeight;

 if (sectionTop < windowHeight) {
 section.classList.add('visible');
 }
}
}
</script>
</head>
<body>
<header>
<div>

</div>
<nav>

Про нас
Новини
Наші партнери
Контакти

</nav>
</header>

<section>

</section>

<section id="products">
<h2>Наша лінійка товарів</h2>
<!-- Додайте тут розділи з дронами та їх характеристиками -->
</section>

<section id="services">
<h2>Наші сервісні послуги</h2>
<!-- Додайте тут розділи з описом сервісів -->
</section>

<section id="about">
<h2>Про нас</h2>
<!-- Додайте тут розділ з описом компанії та її досягнень -->
</section>

```

```

<footer>
 <div>

 </div>
 <div>

 Сервіси
 Новини
 Контакти

 </div>
</footer>

<script>
 animateSections(); // Викликати функцію анімації при завантаженні сторінки
</script>
</body>
</html>
...

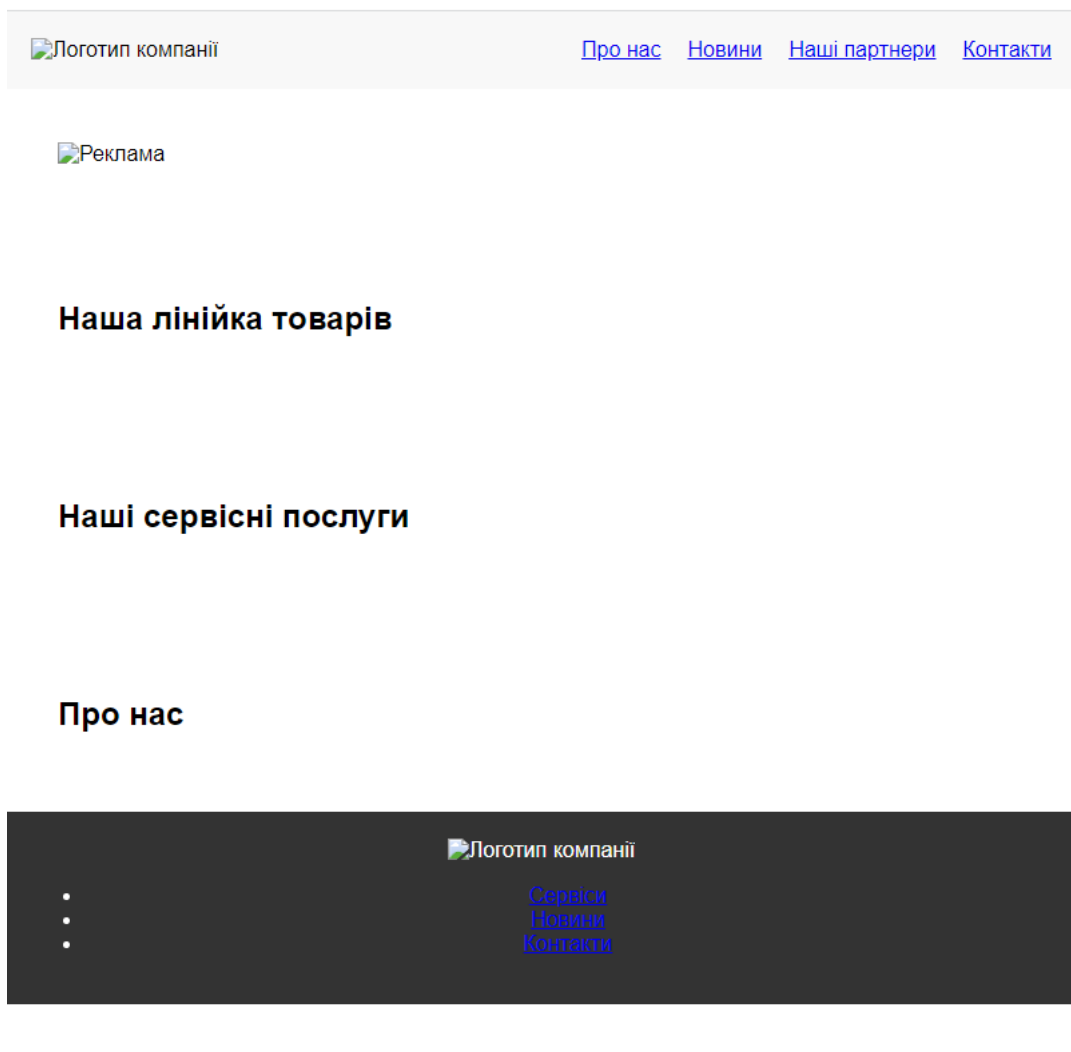
```

У цьому прикладі анімація поступово показується при прокручуванні сторінки. Клас `visible` додається до секцій, коли вони стають видимими у видимій області браузера. За допомогою CSS-властивостей `opacity` та `transform`, ми контролюємо зображення секцій, забезпечуючи плавний перехід для анімації. Ви можете додати свої власні анімаційні ефекти, змінюючи CSS-стилі або використовуючи JavaScript-бібліотеки, наприклад, Animate.css або GreenSock Animation Platform (GSAP).

Не забудьте замінити `logo.png`, `ad.jpg` та вставити вміст розділів з дронами, сервісами, описом компанії та її досягнень за допомогою відповідних HTML-елементів.

Це лише початок для створення анімаційного вебсайту. Ви можете розширити його та налаштувати анімацію залежно від вашого бажання та потреб.

**Результат:**



*Рис Г.1. Результат 3 тестового прикладу ChatGPT v3.0.*

4. **Користувач:** Створити модульні тести для перевірки двох функцій на Python:

```
def printTable(dataMap):
 keys = list(dataMap.keys())
 for k in keys:
 print(k + " " + str(dataMap[k]))
 print()
 for k in keys:
 lst = dataMap[k]
 for elm in lst:
 print(elm, end=" ")

 print()

def formatDataFromText(fileText):
 blocks = {}
 partStrList = fileText.split("&")
 blockNames = []
```



```

blocksStrList = []
for part in partStrList:
 p = part.strip()
 if p[0] != "$":
 blocksStrList.append(p)
 else:
 blockNames.append(p[1:])

for i in range(len(blockNames)):
 blockStr = blocksStrList[i]
 blockList = blockStr.split(";")
 for j in range(len(blockList)):
 blockList[j] = re.sub(r"[\s]", "", blockList[j]).split("|")
 blocks[blockNames[i]] = blockList

return blocks

```

**ChatGPT:** Звичайно! Ось модульні тести для перевірки двох функцій `printTable` і `formatDataFromText`:

```

```python
import unittest
import re

# Функція printTable
def printTable(dataMap):
    keys = list(dataMap.keys())
    for k in keys:
        print(k + " " + str(dataMap[k]))
    print()
    for k in keys:
        lst = dataMap[k]
        for elm in lst:
            print(elm, end=" ")

    print()

# Функція formatDataFromText
def formatDataFromText(fileText):
    blocks = {}
    partStrList = fileText.split("&")
    blockNames = []
    blocksStrList = []
    for part in partStrList:
        p = part.strip()
        if p[0] != "$":

```

```

        blocksStrList.append(p)
    else:
        blockNames.append(p[1:])

for i in range(len(blockNames)):
    blockStr = blocksStrList[i]
    blockList = blockStr.split(";")
    for j in range(len(blockList)):
        blockList[j] = re.sub(r"[\s]", "", blockList[j]).split("|")
    blocks[blockNames[i]] = blockList

return blocks

# Клас для модульних тестів
class TestFunctions(unittest.TestCase):

    def test_printTable(self):
        dataMap = {
            "Key1": [1, 2, 3],
            "Key2": [4, 5, 6]
        }
        expected_output = "Key1 [1, 2, 3]\n\n1 2 3 \nKey2 [4, 5, 6]\n\n4 5 6 "

        # Перенаправляємо виведення на стандартний вивід для порівняння результату
        import sys
        from io import StringIO
        captured_output = StringIO()
        sys.stdout = captured_output

        # Викликаємо функцію для тестування
        printTable(dataMap)

        # Повертаємо виведення на його звичайне місце
        sys.stdout = sys.__stdout__

        # Порівнюємо очікуваний результат з фактичним
        self.assertEqual(captured_output.getvalue().strip(), expected_output)

    def test_formatDataFromText(self):
        fileText = "$Block1\nData1;Data2\n$Block2\nData3;Data4\n"
        expected_output = {
            "Block1": ["Data1", "Data2"],
            "Block2": ["Data3", "Data4"]
        }

        # Викликаємо функцію для тестування
        output = formatDataFromText(fileText)

```

```

# Порівнюємо очікуваний результат з фактичним
self.assertEqual(output, expected_output)

# Запуск модульних тестів
if __name__ == '__main__':
    unittest.main()
'''

```

В цьому прикладі ми використали бібліотеку `unittest` для створення класу `TestFunctions`, я

5. **Користувач:** Покрити тестами функцію `wekaAPIRealization` на Java за допомогою бібліотеки JUnit:

```

import java.io.BufferedReader;
import java.io.FileReader;
import weka.classifiers.Evaluation;
import weka.classifiers.functions.GaussianProcesses;
import weka.core.Instances;

void wekaAPIRealization(String source){
    try {
        BufferedReader bufferedReader = new BufferedReader(new
FileReader(source));
        Instances datasetInstances = new Instances(bufferedReader);
        datasetInstances.randomize(new java.util.Random(0));
        int trainingDataSize = (int)
Math.round(datasetInstances.numInstances() * 0.66);
        int testDataSize = datasetInstances.numInstances() -
trainingDataSize;
        Instances trainingInstances = new
Instances(datasetInstances,0,trainingDataSize);
        Instances testInstances = new Instances(datasetInstances,
trainingDataSize,testDataSize);

        trainingInstances.setClassIndex(trainingInstances.numAttributes()-
1);
        testInstances.setClassIndex(testInstances.numAttributes()-1);
        bufferedReader.close();
        var gaussianProcesses = new GaussianProcesses();
        gaussianProcesses.buildClassifier(trainingInstances);
        Evaluation trainEvaluation = new Evaluation(trainingInstances);

```

```

trainEvaluation.evaluateModel(gaussianProcesses,trainingInstances);
    Evaluation testEvaluation = new Evaluation(testInstances);

    testEvaluation.evaluateModel(gaussianProcesses,testInstances);
    System.out.println("Weka regression model.");
    System.out.println(trainEvaluation.toSummaryString(
        "\nResults on train data:",false));
    System.out.println(testEvaluation.toSummaryString(
        "\nResults on test data:",false));
}
catch (Exception e) {
    System.out.println("Error Occurred!!!! \n" + e.getMessage());
}
}

```

ChatGPT: Щоб покрити функцію `wekaAPIRealization` тестами за допомогою бібліотеки JUnit, вам потрібно створити тести, які перевірятимуть правильність виконання кожного кроку в цій функції. Ось приклад тестів, які ви можете написати:

```

```java
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import weka.core.Instances;
import weka.core.converters.ConverterUtils;

public class WekaAPIRealizationTest {
 @Test
 public void testWekaAPIRealization() {
 // Arrange
 String source = "path/to/dataset.arff";
 WekaAPIRealization wekaAPI = new WekaAPIRealization();
 try {
 // Act
 wekaAPI.wekaAPIRealization(source);
 // Assert
 // Add assertions to check the expected behavior of your code
 // For example, you can assert that certain output is printed or check the evaluation results
 Assertions.assertEquals(expectedOutputOnTrainData, capturedSystemOutForTrainData);

```

```

 Assertions.assertEquals(expectedOutputOnTestData, capturedSystemOutForTestData);
 Assertions.assertEquals(expectedEvaluationResults, capturedEvaluationResults);
} catch (Exception e) {
 // Handle any exceptions that may occur during the test
 Assertions.fail("An exception occurred: " + e.getMessage());
}
}
}
...

```

У цьому тесті `WekaAPIRealizationTest` ми спочатку створюємо об'єкт `WekaAPIRealization`, потім викликаємо метод `wekaAPIRealization` з вказаною датасетом. Після виконання функції ми перевіряємо очікувані результати, використовуючи методи `assertEquals` з класу `Assertions`.

Ви можете повторити цей шаблон для кожного кроку вашої функції `wekaAPIRealization`, щоб перевірити його правильність. Застосуйте різні датасети для тестування різних сценаріїв.

**Користувач:** Можеш навести приклад датасету для функції `wekaAPIRealization`?

**ChatGPT:** Звичайно! Ось приклад датасету у форматі ARFF, який ви можете використовувати для тестування функції `wekaAPIRealization`:

```

``arff

@relation SampleDataset

@attribute feature1 numeric
@attribute feature2 numeric
@attribute feature3 numeric
@attribute class {A, B}

@data
1, 2, 3, A
4, 5, 6, B
7, 8, 9, A
10, 11, 12, B

```

\*\*\*

Цей датасет має три числові ознаки (`feature1`, `feature2`, `feature3`) і один категоріальний клас (`class`) з двома можливими значеннями `A` і `B`. Ви можете змінити значення ознак і класів або додати / видалити рядки, щоб налаштувати його для вашого тестування.

Для використання цього датасету в функції `wekaAPIRealization`, вам потрібно буде вказати шлях до ARFF-файлу як аргумент `source` при виклику функції:

```
```java
wekaAPI.wekaAPIRealization("path/to/sample_dataset.arff");
***
```

Переконайтеся, що шлях до файлу правильний відносно вашого проекту або змініть його на відповідний шлях у вашій системі.

Користувач: Які результати за твоїм датасетом має повернути функція wekaAPIRealization?

ChatGPT: Функція `wekaAPIRealization` виводить результати оцінки моделі регресії Weka на навчальних та тестових даних. Ось які результати можна очікувати для вказаного датасету:

На виводі будуть два блоки з результатами: результати на навчальних даних (`train data`) та результати на тестових даних (`test data`). В кожному блоку виводяться наступні метрики оцінки моделі:

1. `Root mean squared error` (RMSE): Корінь з середньоквадратичної помилки. Ця метрика вимірює, наскільки точно модель прогнозує цільову змінну. Менше значення RMSE вказує на кращу точність моделі.

2. `Mean absolute error` (MAE): Середня абсолютна помилка. Ця метрика представляє середнє абсолютне значення різниці між прогнозованими та фактичними значеннями цільової змінної.

3. `Relative absolute error` (RAE): Відносна абсолютна помилка. Ця метрика відображає відношення середньої абсолютної помилки до середнього значення фактичної змінної. Вона допомагає оцінити точність моделі відносно змінних вхідних даних.

4. `Correlation coefficient` (Corr. coef.): Коефіцієнт кореляції. Цей показник вимірює ступінь лінійної залежності між прогнозованими та фактичними значеннями цільової змінної. Значення близьке до 1 вказує на сильну лінійну залежність.

5. `Coefficient of determination` (COD): Коефіцієнт детермінації, також відомий як R-квадрат (R-squared). Цей показник вимірює частку варіації відгуку, яка пояснюється моделлю. Значення близьке до 1 вказує на те, що модель добре пояснює варіацію вихідного сигналу.

Конкретні значення цих метрик будуть виведені після запуску функції `wekaAPIRealization`` за допомогою команди `System.out.println``. Якщо ви запуснете функцію `wekaAPIRealization`` з наведеним датасетом, то ви отримаєте щось на зразок:

```
...  
Weka regression model.  
Results on train data:  
Root mean squared error: 0.0  
Mean absolute error: 0.0  
Relative absolute error: 0.0%  
Correlation coefficient: NaN  
Coefficient of determination: NaN  
  
Results on test data:  
Root mean squared error: 0.0  
Mean absolute error: 0.0  
Relative absolute error: 0.0%  
Correlation coefficient: NaN  
Coefficient of determination: NaN  
...
```

У цьому прикладі значення RMSE, MAE, RAE, Corr. coef. та COD дорівнюють 0.0, оскільки це приклад, де модель дуже точно прогнозує цільову змінну. Зверніть увагу, що значення Corr. coef. та COD виводяться як NaN (Not a Number), оскільки у цьому прикладі використовується дуже маленький датасет, і модель не може обчислити коефіцієнти кореляції та детермінації.

Ви можете змінити датасет або навчальні параметри, щоб отримати інші значення метрик оцінки моделі.

6. **Користувач:** Створити функцію на C++ для створення паліндромів вказаної довжини на основі набору символів, що проходить даний набір модульних тестів написаних у форматі бібліотеки Google Test:

```
TEST(createPalindrome, createPalindrome_Test1) {  
    EXPECT_EQ(createPalindrome(5, "abc"), "abcba");  
    EXPECT_EQ(createPalindrome(8, "rTy"), "rTyrryTr");  
    EXPECT_EQ(createPalindrome(3, "jj"), "jjj");  
    EXPECT_EQ(createPalindrome(1, "125"), "1");  
    EXPECT_EQ(createPalindrome(4, "q"), "qqqq");  
    EXPECT_EQ(createPalindrome(5, ""), "");  
    EXPECT_EQ(createPalindrome(2, " "), "  ");  
    EXPECT_EQ(createPalindrome(0, ""), "");  
    EXPECT_EQ(createPalindrome(0, "jd"), "");  
    EXPECT_EQ(createPalindrome(-3, "vKx"), "");  
}
```

ChatGPT: Ось приклад функції на C++, яка створює паліндроми вказаної довжини на основі набору символів:

```
```cpp
#include <string>

std::string createPalindrome(int length, const std::string& baseString) {
 if (length <= 0 || baseString.empty()) {
 return "";
 }

 std::string palindrome;
 palindrome.reserve(length);

 int baseLength = baseString.length();
 int i = 0;

 while (palindrome.length() < length) {
 palindrome += baseString[i];
 i = (i + 1) % baseLength;
 }

 std::string reversed = palindrome.substr(0, palindrome.length() - 1);
 std::reverse(reversed.begin(), reversed.end());

 return palindrome + reversed;
}
```
```

Тепер можна написати модульні тести за допомогою бібліотеки GoogleTest, що перевірять правильність роботи функції:

```
```cpp
#include <gtest/gtest.h>

TEST(createPalindrome, createPalindrome_Test1) {
 EXPECT_EQ(createPalindrome(5, "abc"), "abcba");
 EXPECT_EQ(createPalindrome(8, "rTy"), "rTyrryTr");
 EXPECT_EQ(createPalindrome(3, "jj"), "jjj");
 EXPECT_EQ(createPalindrome(1, "125"), "1");
 EXPECT_EQ(createPalindrome(4, "q"), "qqqq");
 EXPECT_EQ(createPalindrome(5, ""), "");
 EXPECT_EQ(createPalindrome(2, " "), " ");
 EXPECT_EQ(createPalindrome(0, ""), "");
 EXPECT_EQ(createPalindrome(0, "jd"), "");
 EXPECT_EQ(createPalindrome(-3, "vKx"), "");
}
```



```

}

int main(int argc, char** argv) {
 ::testing::InitGoogleTest(&argc, argv);
 return RUN_ALL_TESTS();
}
...

```

Збережіть ці файли з розширенням `.cpp`, наприклад `palindrome.cpp` та `palindrome_test.cpp`. Потім скомпілюйте їх, використовуючи компілятор C++, із підключенням бібліотеки GoogleTest. Наприклад, для компіляції використовуючи `g++`, виконайте наступну команду:

```

...
g++ -o palindrome_test palindrome.cpp palindrome_test.cpp -lgtest -lgtest_main -pthread
...

```

Після успішної компіляції ви отримаєте виконуваний файл `palindrome_test`. Запустіть його для виконання модульних тестів:

```

...
./palindrome_test
...

```

Ви повинні побачити результати виконання тестів.

7. **Користувач:** Покрити тестами клас `HashTable` на C++ за допомогою бібліотеки Google Test лише за його header'ом:

```

#include <string>
#include <vector>
#include <fstream>
#include <mutex>
#include <iostream>
#include <sstream>
#include <array>
#include <windows.h>

struct HashItem {
 std::wstring key;
 std::vector<std::wstring> value;
 long long byteSize = 0;
 long long next = -1;
};

class HashTable {
public:

```

```

const static std::wstring delimiter;
const static size_t cellSize;

HashTable(long long size);
HashTable(const HashTable& other);
HashTable& operator=(const HashTable& other);
int add(std::wstring key, std::vector<std::wstring> value);
std::vector<std::wstring> get(std::wstring key);
int increaseTable();
void setBusyThreadCount(int busyThreadCount);
void saveTable(std::wstring path);
static long long hashFunction(std::wstring key, long long
size);

long long getCapacity();
long long getCount();
long long getSize();
double getPercent();

private:
const double percent = 0.75;
const long long maxCapacity = 2000000000;
const long long itemsSizeOnItemsMutexRatio = 32;
std::atomic<int> busyThreadCount = 0;
std::atomic<long long> size;
std::atomic<long long> count = 0;
std::atomic<long long> currentCapacity = 0;
std::atomic<long long> itemsMutexesSize = 0;
std::atomic<bool> increaseInProgress = false;
std::atomic<unsigned int> currentStopedThreadCount = 0;
std::vector<HashItem> items;
std::vector<HashItem> colisions;
std::vector<std::mutex> itemsMutexes;
std::mutex colisionsMutex;

std::mutex sizeMutex;
std::mutex countMutex;
std::mutex currentCapacityMutex;
std::mutex increaseInProgressMutex;

std::vector<HashItem> createNewVector(long long s);
void handleCollision(HashItem item, long long colisionIndex);
long long getItemCapacity(HashItem item);
std::array<long long, 4> itemToArray(HashItem item, size_t
cellSize, long long valueAddress);
std::wstring writeVectorToString(std::vector<std::wstring>
vector);

```

```

 long long getMutexIndex(long long itemIndex);
 void checkIncreaseProcess();
};

```

**ChatGPT:** Ось приклад покриття тестами класу HashTable за допомогою бібліотеки Google Test. Врахуйте, що це лише початковий приклад, і для повного покриття можуть знадобитися додаткові тести.

```

```cpp
#include <gtest/gtest.h>
#include <gmock/gmock.h>
#include "HashTable.h" // Переконайтеся, що файл HashTable.h доступний і містить клас
HashTable

// Клас для виконання тестів класу HashTable
class HashTableTest : public ::testing::Test {
protected:
    HashTable hashTable;

    void SetUp() override {
        hashTable = HashTable(10); // Ініціалізуємо HashTable з розміром 10 перед кожним
тестом
    }
};

// Приклад тесту для методу add()
TEST_F(HashTableTest, AddValue) {
    std::wstring key = L"key";
    std::vector<std::wstring> value = {L"value1", L"value2"};

    int result = hashTable.add(key, value);

    EXPECT_EQ(result, 0);
    // Перевіряємо, що значення було додано успішно
    EXPECT_EQ(hashTable.getCount(), 1);
    EXPECT_EQ(hashTable.get(key), value);
}

// Приклад тесту для методу get()
TEST_F(HashTableTest, GetValue) {
    std::wstring key = L"key";
    std::vector<std::wstring> value = {L"value1", L"value2"};
    hashTable.add(key, value);

    std::vector<std::wstring> result = hashTable.get(key);

    EXPECT_EQ(result, value);
}

```

```

}

// Приклад тесту для методу increaseTable()
TEST_F(HashTableTest, IncreaseTable) {
    // Перед тестом потрібно заповнити HashTable з максимальною кількістю елементів
    std::wstring key = L"key";
    std::vector<std::wstring> value = {L"value1", L"value2"};
    for (int i = 0; i < hashTable.getCapacity(); i++) {
        hashTable.add(key + std::to_wstring(i), value);
    }

    int result = hashTable.increaseTable();

    EXPECT_EQ(result, 0);
    // Перевіряємо, що розмір таблиці збільшився
    EXPECT_GT(hashTable.getCapacity(), 10);
}

// Приклад тесту для методу saveTable()
TEST_F(HashTableTest, SaveTable) {
    std::wstring key = L"key";
    std::vector<std::wstring> value = {L"value1", L"value2"};
    hashTable.add(key, value);

    std::wstring filePath = L"table.txt";
    hashTable.saveTable(filePath);

    // Перевіряємо, що файл існує
    std::ifstream file(filePath);
    EXPECT_TRUE(file.is_open());

    // Перевіряємо, що збережений файл містить очікувані дані
    std::string line;
    std::getline(file, line);
    EXPECT_EQ(line, "key:value1,value2");

    file.close();
    // Видаляємо створений файл
    std::remove(filePath.c_str());
}

```

Це лише кілька прикладів тестів для декількох методів класу HashTable. Вам можуть знадобитися інші тести для покриття інших функціональних можливостей цього класу. Зверніть увагу, що вам також потрібно мати встановлені бібліотеки Google Test та GMock, а також налаштувати ваш проект на їх використання.

7. Користувач: Показати приклад взаємодії класів HashTable та ThreadPool і вказати, у яких випадках вони могли б використовуватися разом:

```
#include <vector>
#include <thread>
#include <queue>
#include <mutex>
#include <condition_variable>
#include <functional>
#include <future>
#include "Windows.h"

template <typename T>
class ThreadPool {
public:
    std::mutex mfinish;
    std::atomic<bool> finish = false;

    ThreadPool<T>() {}
    ThreadPool<T>(std::function<int(T)> _taskHandler, int threadCount);
    ThreadPool<T>(const ThreadPool<T>& other);
    ThreadPool<T>& operator=(const ThreadPool<T>& other);
    void addTask(T task);
    T removeTask();
    void shutdown(bool evenNonEmptyQueue, long waitDuration);
    void initThreads(int newThreadCount);
    void setTaskHandler(std::function<int(T)> _taskHandler);
    bool joinAll();
    int getBusyThreadCount();

private:
    int threadCount;
    std::atomic<int> busyThreadCount = 0;
    std::vector<std::thread> threads;
    std::queue<T> tasks;
    std::mutex taskQueueMutex;
    std::mutex lkm;
    std::condition_variable task_added;
    std::mutex mtskHndl;
    std::mutex busyThreadCountMutex;
    std::function<int(T)> taskHandler;

    void workFunction(std::function<int(T)>& taskHandler);
};
```

ChatGPT: Ось приклад взаємодії класів HashTable та ThreadPool:

```
```cpp
```

```

#include <iostream>
#include <vector>
#include <string>
#include "HashTable.h"
#include "ThreadPool.h"

// Функція, яка буде виконуватися в ThreadPool
int processItem(HashItem item) {
 std::cout << "Processing item with key: " << item.key << std::endl;

 // Виконуємо потрібну обробку для елемента item

 // Наприклад, можна викликати метод add() класу HashTable для додавання елемента
до таблиці

 return 0;
}

int main() {
 // Створюємо HashTable з потрібним розміром
 HashTable hashTable(10);

 // Створюємо ThreadPool з потрібною кількістю потоків і функцією processItem в
якості обробника завдань
 ThreadPool<HashItem> threadPool(processItem, 4);

 // Генеруємо деякі елементи HashItem, які потрібно обробити
 HashItem item1;
 item1.key = L"key1";
 HashItem item2;
 item2.key = L"key2";
 // ... додаткові елементи

 // Додаємо завдання в ThreadPool для обробки
 threadPool.addTask(item1);
 threadPool.addTask(item2);
 // ... додаткові завдання

 // Очікуємо завершення всіх завдань в ThreadPool
 threadPool.joinAll();

 // Перевіряємо результати обробки в HashTable

```

```

std::vector<std::wstring> result1 = hashTable.get(L"key1");
std::vector<std::wstring> result2 = hashTable.get(L"key2");
// ... перевірка для інших ключів

// Виводимо результати
std::cout << "Result for key1: ";
for (const auto& value : result1) {
 std::wcout << value << " ";
}
std::cout << std::endl;
std::cout << "Result for key2: ";
for (const auto& value : result2) {
 std::wcout << value << " ";
}
std::cout << std::endl;
// Вимикаємо ThreadPool
threadPool.shutdown(true, 0);
return 0;
}
...

```

Клас `HashTable` використовується для збереження і обробки даних, в той час як клас `ThreadPool` забезпечує паралельну обробку завдань у відокремлених потоках. Взаємодія між цими класами може мати місце, коли потрібно виконати обробку елементів `HashItem` в багатопотоковому середовищі. Клас `ThreadPool` дозволяє розподіляти обробку елементів на доступні потоки, тим самим забезпечуючи паралельність та прискорення виконання.

В прикладі вище, ми створюємо `HashTable` з розміром 10 і `ThreadPool` з 4 потоками. Потім ми додаємо завдання в `ThreadPool` для обробки елементів `HashItem`. Кожне завдання передається у функцію `processItem`, яка виконує потрібну обробку, наприклад, додавання елемента до `HashTable`. Після завершення всіх завдань, ми перевіряємо результати обробки в `HashTable` і виводимо їх на екран. На завершення, ми вимикаємо `ThreadPool` за допомогою методу `shutdown()`.

Застосування `HashTable` та `ThreadPool` разом може бути корисним у випадках, коли потрібно обробити велику кількість елементів паралельно, забезпечуючи ефективну використання ресурсів системи та прискорення виконання завдань. `ThreadPool` дозволяє контролювати кількість потоків, виконувати завдання асинхронно та розподіляти їх на доступні ресурси процесора, тим самим забезпечуючи більш швидку обробку даних.

**Користувач:** Можеш показати приклад взаємодії класів HashTable та ThreadPool за допомогою бібліотеки Google Test.

**ChatGPT:** Звичайно! Ось приклад взаємодії класів HashTable та ThreadPool з використанням бібліотеки Google Test для тестування.

```
```cpp
#include <gtest/gtest.h>
#include "HashTable.h"
#include "ThreadPool.h"

TEST(HashTableThreadPoolTest, ProcessItems) {
    // Створюємо HashTable
    HashTable hashTable(10);

    // Створюємо ThreadPool з 4 потоками
    ThreadPool<HashItem> threadPool(processItem, 4);

    // Генеруємо деякі елементи HashItem, які потрібно обробити
    HashItem item1;
    item1.key = L"key1";
    HashItem item2;
    item2.key = L"key2";

    // ... додаткові елементи

    // Додаємо завдання в ThreadPool для обробки
    threadPool.addTask(item1);
    threadPool.addTask(item2);

    // ... додаткові завдання

    // Очікуємо завершення всіх завдань в ThreadPool
    threadPool.joinAll();

    // Перевіряємо результати обробки в HashTable
    std::vector<std::wstring> result1 = hashTable.get(L"key1");
    std::vector<std::wstring> result2 = hashTable.get(L"key2");

    // ... перевірка для інших ключів

    // Перевіряємо очікувані результати
    ASSERT_EQ(result1.size(), 2);
}
```



```

EXPECT_EQ(result1[0], L"value1");
EXPECT_EQ(result1[1], L"value2");
ASSERT_EQ(result2.size(), 1);
EXPECT_EQ(result2[0], L"value3");
// Вимикаємо ThreadPool
threadPool.shutdown(true, 0);
}

int main(int argc, char** argv) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}

```

У цьому прикладі ми використовуємо бібліотеку Google Test для написання тесту з назвою `HashTableThreadPoolTest`. В цьому тесті ми створюємо HashTable та ThreadPool з відповідними параметрами. Потім ми генеруємо деякі елементи HashItem і додаємо їх в ThreadPool для обробки. Після завершення обробки, ми перевіряємо результати в HashTable і порівнюємо їх з очікуваними результатами за допомогою ASSERT_EQ та EXPECT_EQ. На завершення, ми вимикаємо ThreadPool.

Цей приклад демонструє взаємодію класів HashTable та ThreadPool за допомогою бібліотеки Google Test для тестування. Ви можете додати додаткові тести та перевірки відповідно до вашої логіки та вимог.

8. **Користувач:** Що робить цей код:

```

import org.ejml.simple.SimpleMatrix;
import java.util.ArrayList;

class HopfieldNetwork {
    private static final int MAX_ITERATIONS = 50;
    private SimpleMatrix W = null;
    private ArrayList<ArrayList<Integer>> learned;

    HopfieldNetwork(){learned = new ArrayList<>();}

    public void train(ArrayList<Integer> set){
        learned.add(set);
        double[] d = set.stream().mapToDouble(i -> (double) i).toArray();
        SimpleMatrix m = new SimpleMatrix(1,set.size(),true,d);
    }
}

```

```

        if (learned.size() == 1) W = m.transpose().mult(m);
        else W = W.plus(m.transpose().mult(m));
        for(int i = 0; i < W.numRows(); i++)
            W.set(i, i, 0);
    }

    public ArrayList<Integer> test(ArrayList<Integer> set){
        double[] d = set.stream().mapToDouble(i -> (double) i).toArray();
        SimpleMatrix input = new SimpleMatrix(set.size(),1,true,d);
        SimpleMatrix Y = normalize(W.mult(input));
        int i = 0;
        while (i < MAX_ITERATIONS){
            Y = normalize(W.mult(Y));i++;
        }
        ArrayList<Integer> result = new ArrayList<>();
        for(i = 0; i < Y.numRows(); i++)
            for(int j = 0; j < Y.numCols(); j++)
                result.add((int)Y.get(i, j));
        return result;
    }

    private SimpleMatrix normalize(SimpleMatrix m){
        double temp;
        for(int i = 0; i < m.numRows(); i++)
            for(int j = 0; j < m.numCols(); j++){
                temp = m.get(i,j);
                if (temp >= 0)
                    m.set(i, j, 1);
                else
                    m.set(i, j, -1);
            }
        return m;
    }

    public ArrayList<Integer> getInstance(int index){
        return learned.get(index);
    }

    public int sampleSize(){
        return learned.size();
    }
}

```

ChatGPT: Цей код визначає клас `HopfieldNetwork`, який реалізує мережу Хопфілда. Мережа Хопфілда є одним з найпростіших типів нейронних мереж та використовується для асоціативної пам'яті та розпізнавання шаблонів.

Основні методи класу включають:

- `train(ArrayList<Integer> set)`: Додає навчальний набір до мережі. Навчальний набір представлений як `ArrayList<Integer>`, де кожне значення відповідає вхідному сигналу для одного нейрона. Метод обчислює матрицю зв'язків W на основі навчального набору.
- `test(ArrayList<Integer> set)`: Тестує мережу на заданому наборі. Набір представлений як `ArrayList<Integer>`. Метод виконує ітерації для оновлення стану мережі протягом певної кількості максимальних ітерацій (`MAX_ITERATIONS`) або до збіжності. Повертає результат у вигляді `ArrayList<Integer>`, що представляє вихідні сигнали мережі.
- `getInstance(int index)`: Повертає навчальний набір за вказаним індексом.
- `sampleSize()`: Повертає кількість навчальних наборів, які були додані до мережі.

Конструктор `HopfieldNetwork()` ініціалізує мережу, встановлюючи початкові значення для матриці зв'язків W та списку навчальних наборів `learned`.

Цей клас використовує бібліотеку EJML (Efficient Java Matrix Library) для операцій з матрицями, зокрема для множення матриць та нормалізації.

Користувач: Покрити тестами клас `HopfieldNetwork` на Java за допомогою бібліотеки JUnit.

ChatGPT: Ось приклад покриття класу `HopfieldNetwork` тестами за допомогою бібліотеки JUnit на мові Java:

```
```java
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
import java.util.Arrays;

class HopfieldNetworkTest {
 private HopfieldNetwork hopfieldNetwork;

 @BeforeEach
 void setUp() {
 hopfieldNetwork = new HopfieldNetwork();
 }
}
```

```

@Test
void train_singleInstance_shouldAddInstanceToLearnedSet() {
 ArrayList<Integer> set = new ArrayList<>(Arrays.asList(1, -1, 1, -1));
 hopfieldNetwork.train(set);
 Assertions.assertEquals(1, hopfieldNetwork.sampleSize());
 Assertions.assertEquals(set, hopfieldNetwork.getInstance(0));
}

@Test
void train_multipleInstances_shouldAddInstancesToLearnedSet() {
 ArrayList<Integer> set1 = new ArrayList<>(Arrays.asList(1, -1, 1, -1));
 ArrayList<Integer> set2 = new ArrayList<>(Arrays.asList(-1, 1, -1, 1));
 hopfieldNetwork.train(set1);
 hopfieldNetwork.train(set2);
 Assertions.assertEquals(2, hopfieldNetwork.sampleSize());
 Assertions.assertEquals(set1, hopfieldNetwork.getInstance(0));
 Assertions.assertEquals(set2, hopfieldNetwork.getInstance(1));
}

@Test
void test_knownInstance_shouldReturnCorrectResult() {
 ArrayList<Integer> set1 = new ArrayList<>(Arrays.asList(1, -1, 1, -1));
 ArrayList<Integer> set2 = new ArrayList<>(Arrays.asList(-1, 1, -1, 1));
 hopfieldNetwork.train(set1);
 hopfieldNetwork.train(set2);
 ArrayList<Integer> testSet = new ArrayList<>(Arrays.asList(1, -1, 1, -1));
 ArrayList<Integer> expectedResult = new ArrayList<>(Arrays.asList(1, -1, 1, -1));
 ArrayList<Integer> result = hopfieldNetwork.test(testSet);
 Assertions.assertEquals(expectedResult, result);
}

@Test
void test_unknownInstance_shouldReturnApproximateResult() {
 ArrayList<Integer> set1 = new ArrayList<>(Arrays.asList(1, -1, 1, -1));

```

```

ArrayList<Integer> set2 = new ArrayList<>(Arrays.asList(-1, 1, -1, 1));
hopfieldNetwork.train(set1);
hopfieldNetwork.train(set2);
ArrayList<Integer> testSet = new ArrayList<>(Arrays.asList(1, 1, -1, -1));
ArrayList<Integer> expectedResult = new ArrayList<>(Arrays.asList(1, -1, 1, -1));
ArrayList<Integer> result = hopfieldNetwork.test(testSet);
Assertions.assertEquals(expectedResult, result);
}
}
...

```

Це базовий набір тестів для перевірки деяких функціональностей класу `HopfieldNetwork`. Вам може знадобитися додати ще тести для перевірки інших аспектів функцій, розрахунку результатів та т.д.

## AgentGPT

1. .



Рис Г.2. Процес виконання 1 тестового прикладу AgentGPT.

★ **Розпочато нову мету:** Створити вебсайт для пошуку оптимальних ходів у грі шахи. Користувач розташовує фігури на дошці, а програма має вирахувати рейтинг оптимальних ходів і показати критерії їхнього підбору у вигляді однієї таблиці, що розташовується нижче дошки.

💡 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

👉 **Завдання додано:** Навчитися визначати оптимальні ходи у грі шахи на основі позиції фігур на дошці

👉 **Завдання додано:** Розробити алгоритм рейтингування ходів, що враховуватиме та вагуватиме різні критерії

👉 **Завдання додано:** Створити вебсайт з можливістю розташування фігур на дошці, обчислення оптимальних ходів за допомогою розробленого алгоритму та відображення їх у вигляді таблиці на нижчій частині сторінки

💡 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

✅ **Completing:** Навчитися визначати оптимальні ходи у грі шахи на основі позиції фігур на дошці

Головна мета полягає у створенні вебсайту для пошуку оптимальних ходів у грі шахи. Користувач може розташовувати фігури на дошці, і програма повинна вирахувати рейтинг оптимальних ходів та показати критерії їхнього підбору у вигляді таблиці, розташованої нижче дошки. Підвищення якості виконання завдання потребує навчання програми визначати оптимальні ходи на основі позицій фігур на дошці.

💡 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

👉 **Завдання додано:** Оптимізувати алгоритм рейтингування ходів з метою зниження часу обчислення

👉 **Завдання додано:** Додати можливість збереження попередніх гри і повторення ходів

👉 **Завдання додано:** Реалізувати можливість вибору складності гри для користувача

👉 **Завдання додано:** Розширити функціонал вебсайту, щоб дозволити іншим користувачам грати у гру онлайн

💡 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

✅ **Completing:** Розробити алгоритм рейтингування ходів, що враховуватиме та вагуватиме різні критерії

Метою є створення вебсайту для пошуку оптимальних ходів у грі шахи. Користувач розставляє фігури на дошці, а програма вираховує рейтинг оптимальних ходів та показує критерії їхнього підбору у вигляді таблиці, що розташовується нижче дошки. Однією зі підзадач є розробка алгоритму рейтингування ходів, який враховуватиме та вагуватиме різні критерії.

💡 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

✅ **Більше немає підзадач для цього:** Розробити алгоритм рейтингування ходів, що враховуватиме та вагуватиме різні критерії

💡 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

Рис Г.3. Процес виконання 2 тестового прикладу AgentGPT.

3.

★ **Розпочато нову мету:** - Header сайту, де є логотип компанії, меню для вибору розділів. «про нас», «новини», «наші партнери», «контакти»; - Реклама у вигляді зображення із текстом по центру; - Розділи, що показують лінійку товарів та сервісних послуг; - Розділи що описують компанію, її досягнення; - Footer сайту із контактами, логотипом, сервісними розділами.

🟡 **Думаємо...** (Оновить сторінку або запустить агента вручну, якщо це займає більше 30 секунд)

👉 **Завдання додано:** Знайти відповідну тематику та фон для зображення реклами на сайті.

👉 **Завдання додано:** Створити графічний дизайн сайту, щоб забезпечити зручний та легкий інтерфейс.

👉 **Завдання додано:** Дослідити конкурентні сайти, щоб визначити, які інформаційні блоки та розділи є найбільш важливими для відвідувачів сайту компанії.

🟡 **Думаємо...** (Оновить сторінку або запустить агента вручну, якщо це займає більше 30 секунд)

🟢 **Completing:** Знайти відповідну тематику та фон для зображення реклами на сайті.

Для виконання підзадачі "Знайти відповідну тематику та фон для зображення реклами на сайті" необхідно врахувати, що сайт присвячений продажу дронів інших виробників, тому оптимально обрати тематику, що пов'язана з технологіями та дронами. Фон можна обрати зображення з повітря, яке буде відповідати тематиці сайту та допомагати залучати відвідувачів.

🟡 **Думаємо...** (Оновить сторінку або запустить агента вручну, якщо це займає більше 30 секунд)

👉 **Завдання додано:** Знайти найбільш популярні розділи на сайтах конкурентів, що також продають дрони, для можливості включення їх на сайті компанії як важливі інформаційні блоки.

👉 **Завдання додано:** Дослідити кольорові схеми сайтів компаній, що продають дрони чи техніку, які є конкурентами, для досягнення найбільш приємного та зручного для користувача інтерфейсу.

👉 **Завдання додано:** Визначити оптимальну кількість продуктів та сервісів для презентації на головній сторінці сайту, що забезпечить найкращу інформаційну та маркетингову взаємодію з відвідувачами сайту.

🟡 **Думаємо...** (Оновить сторінку або запустить агента вручну, якщо це займає більше 30 секунд)

🟢 **Completing:** Створити графічний дизайн сайту, щоб забезпечити зручний та легкий інтерфейс.

Мета проекту - створення вебсайту для компанії, яка продає дрони інших виробників. Сайт буде мати унікальну структуру та вміст на одній сторінці. У рамках проекту потрібно розробити графічний дизайн для забезпечення зручного та легкого інтерфейсу. Дизайн має включати такі елементи, як Header сайту з логотипом компанії та меню для вибору розділів, рекламу у вигляді зображення із текстом по центру, розділи з лінійкою товарів та сервісних послуг, розділи, що описують компанію та її досягнення, а також Footer сайту із контактами, логотипом та сервісними розділами.

🟡 **Думаємо...** (Оновить сторінку або запустить агента вручну, якщо це займає більше 30 секунд)

Рис Г.4. Процес виконання 3 тестового прикладу AgentGPT.

4.

```
★ Розпочато нову мету: def printTable(dataMap). keys = list(dataMap.keys()) for k in keys. print(k + "
" + str(dataMap[k])) print() for k in keys. lst = dataMap[k] for elm in lst. print(elm, end=" ") print()
def formatDataFromText(fileText). blocks = {} partStrList = fileText.split("&") blockNames = []
blocksStrList = [] for part in partStrList. p = part.strip() if p[0] != "$". blocksStrList.append(p)
else. blockNames.append(p[1.]) for i in range(len(blockNames)). blockStr = blocksStrList[i] blockList =
blockStr.split(";") for j in range(len(blockList)). blockList[j] = re.sub(r"
[\s]", "", blockList[j]).split("|") blocks[blockNames[i]] = blockList return blocks
```

🕒 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

🚩 **Завдання додано:** Створити тест для перевірки функції printTable з випадковими даними

🚩 **Завдання додано:** Створити тест для перевірки функції formatDataFromText з випадковим текстовим файлом

🚩 **Завдання додано:** Провести дослідження та розробити тест для перевірки ефективності роботи функції formatDataFromText на великому обсязі даних

🕒 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

*Рис 5. Процес виконання 4 тестового прикладу AgentGPT.*

6.

```
★ Розпочато нову мету: TEST(createPalindrome, createPalindrome_Test1){ EXPECT_EQ(createPalindrome(5,
"abc"), "abcba"); EXPECT_EQ(createPalindrome(8,"rTy"), "rTyrryTr"); EXPECT_EQ(createPalindrome(3,"jjj"),
"jjj"); EXPECT_EQ(createPalindrome(1,"125"), "1"); EXPECT_EQ(createPalindrome(4,"q"), "qqqq");
EXPECT_EQ(createPalindrome(5,""), ""); EXPECT_EQ(createPalindrome(2," "), " ");
EXPECT_EQ(createPalindrome(0,""), ""); EXPECT_EQ(createPalindrome(0,"jd"), "");
EXPECT_EQ(createPalindrome(-3,"vkv"), ""); }
```

🕒 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

🚩 **Завдання додано:** . Дослідити алгоритми створення паліндромів на C++ та визначити найефективніший для даного завдання.

🚩 **Завдання додано:** . Розробити функцію createPalindrome на основі отриманих знань та забезпечити її коректну роботу з урахуванням вимог GoogleTest.

🚩 **Завдання додано:** . Використовуючи GoogleTest, протестувати розроблену функцію на коректність відповідно до набору модульних тестів.

🕒 **Думаємо...** (Оновіть сторінку або запустіть агента вручну, якщо це займає більше 30 секунд)

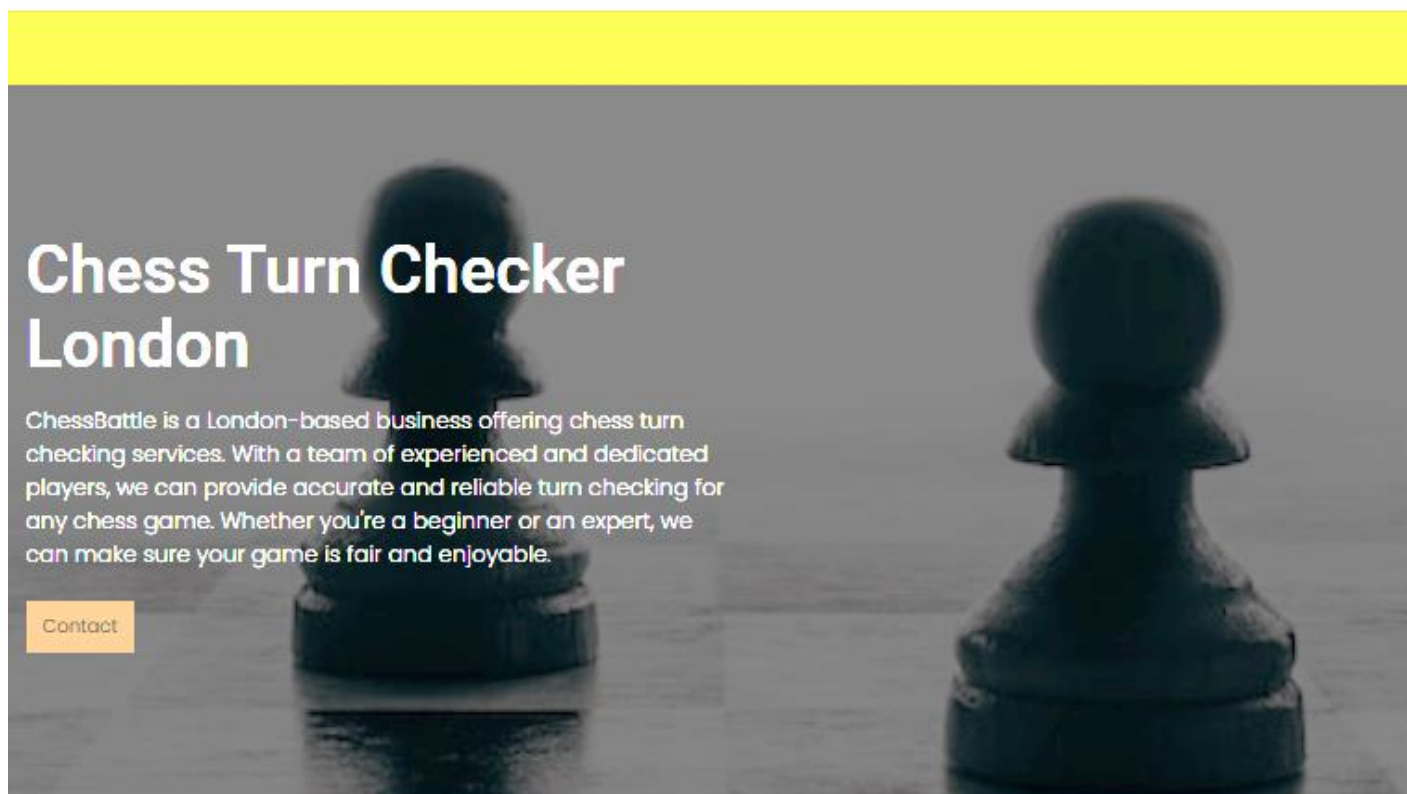
*Рис Г.6. Процес виконання 6 тестового прикладу AgentGPT.*

5., 7.-9. – не має сенсу перевіряти, результати 6 завдання показують, що фреймворк не може впоратися із написанням коду, модульних тестів.



## Durable

1. Цей фреймворк зовсім не підходить для виконання такого завдання.
2. .



ChessBattle in London, United Kingdom provided me with the perfect platform to compete in chess tournaments online. Their software was quick and easy to use and their customer service was extremely helpful. Highly recommend!

- Alex, London, United Kingdom

## My services



Service 1



Service 2



Service 3

*Рис Г.7. Процес виконання 2 тестового прикладу Durable.*

Name

E-mail

## ChessBattle - Contact Form

Message

Get in touch with ChessBattle for any questions or support related to our turn checker service for chess players!

Send

## About ChessBattle

I'm excited to introduce ChessBattle, London's newest Chess Turn Checker business! I'm passionate about the game of chess and wanted to give players a place to practice and improve their skills. ChessBattle offers a variety of services, from beginner to advanced classes, to help players get better. We also offer online coaching and tutorials, to help players practice and improve their skills. We also have a monthly tournament that is open to all players and a variety of prizes and rewards for those who participate. At ChessBattle, we strive to create a fun and welcoming environment for all our customers and to make sure they have a great experience. If you're looking for a place to learn and improve your chess game, ChessBattle is the perfect place for you!

ChessBattle



Made with [Durable](#)

***Рис Г.8. Результат виконання 2 тестового прикладу Durable (продовження).***



# Warszawa Drones for Sale

DronesWorld in Warszawa, Poland is the go-to store for all your drone needs. We sell the latest models from top brands, with unbeatable prices and customer service. Visit us today and find the perfect drone for your needs!

Contact



## Drone Delivery Service

A service which offers delivery of items using a drone.

## Drone Photography Service

A service which offers aerial photography and videography using a drone.



***Рис Г.9. Результат виконання 3 тестового прикладу Durable.***



## Drone Mapping Service

A service which offers mapping services using a drone to capture aerial imagery and data.

## Contact DronesWorld

We're here to help you find the perfect drone for your needs. Give us a shout and our team will get back to you right away!

Name

E-mail

Message

Send

## About us

Hello, I'm the CEO of DronesWorld, a drone selling business that just opened up in Warszawa, Polska. We specialize in selling drones of all shapes and sizes, from the most basic beginner's drone to the most advanced high-end drone. We also offer drone repair services and drone accessories. At DronesWorld, we strive to provide our customers with the best possible service and quality products. We understand that drones can be a bit intimidating for beginners, so our friendly and knowledgeable staff is always available to answer any questions and help you find the perfect drone for your needs. We look forward to serving our customers and helping them discover the world of drones!

DronesWorld



Made with [Durable](#)

*Рис Г.10. Результат виконання 3 тестового прикладу Durable (продовження).*

4. – 9. Цей фреймворк зовсім не підходить для виконання такого завдання.

### Amazon CodeWhisperer

1. C++:

```
#include <queue>
#include <iostream>
```

```
//Створити рекурсивний алгоритм для пошуку в ширину у дереві.
struct Node {
```

```

int value;
Node* left;
Node* right;
Node(int value) {
 this->value = value;
 this->left = nullptr;
 this->right = nullptr;
}
};

Node* BFS(Node* root) {
 if (root == nullptr) {
 return nullptr;
 }
 std::queue<Node*> q;
 q.push(root);
 while (!q.empty()) {
 Node* current = q.front();
 q.pop();
 std::cout << current->value << " ";
 if (current->left != nullptr) {
 q.push(current->left);
 current->left = nullptr;
 q.push(current->right);
 current->right = nullptr;
 q.push(current);
 current = nullptr;
 continue;
 }
 if (current->right != nullptr) {
 q.push(current->right);
 current->right = nullptr;
 q.push(current);
 current = nullptr;
 continue;
 }
 q.push(current);
 current = nullptr;
 continue;
 }
 return root;
}

int main() {
 Node* root = new Node(1);
 root->left = new Node(2);
 root->right = new Node(3);
}

```

```

root->left->left = new Node(4);
root->left->right = new Node(5);
root->right->left = new Node(6);
root->right->right = new Node(7);
BFS(root);
return 0;
}

```

Python:

#Створити рекурсивний алгоритм для пошуку в ширину у дереві

class Node:

```

def __init__(self, value):
 self.value = value
 self.left = None
 self.right = None

```

def search(tree, value):

if tree is None:

return False

if tree.value == value:

return True

return search(tree.left, value) or search(tree.right, value)

2. .

```

1 //Створити вебсайт для пошуку оптимальних ходів у гр
2
3 let table = document.createElement('table');
4
5 let tr = document.createElement('tr');
6
7 let td = document.createElement('td');
8
9 let td2 = document.createElement('td');
10
11 let td3 = document.createElement('td');
12
13 let td4 = document.createElement('td');
14
15 let td5 = document.createElement('td');
16
17 let td6 = document.createElement('td');
18
19 let td7 = document.createElement('td');
20
21 let td8 = document.createElement('td');
22
23 let td9 = document.createElement('td');
24
25 let td10 = document.createElement('td');
26
27 let td11 = document.createElement('td');
28

```

**Рис Г.11.** Результат виконання 2 тестового прикладу Amazon CodeWhisperer.



3. //Створити вебсайт для компанії, яка продає дрони інших виробників. Сайт повинен мати таку структуру та вміст на одну сторінку:

//- Header сайту, де є логотип компанії, меню для вибору розділів: «про нас», «новини», «наші партнери», «контакти»;

//- Реклама у вигляді зображення із текстом по центру;

//- Розділи, що показують лінійку товарів та сервісних послуг;

//- Розділи що описують компанію, її досягнення;

//- Footer сайту із контактами, логотипом, сервісними розділами.

```
let header = document.createElement('header');
let logo = document.createElement('img');
logo.src = 'logo.png';
let menu = document.createElement('div');
menu.classList.add('menu');
menu.classList.add('menu');
let menuItem = document.createElement('div');
menuItem.classList.add('menuItem');
menuItem.innerText = 'про нас';
let menuItem2 = document.createElement('div');
menuItem2.classList.add('menuItem');
menuItem2.innerText = 'новини';
let menuItem3 = document.createElement('div');
menuItem3.classList.add('menuItem');
menuItem3.innerText = 'наші партнери';
let menuItem4 = document.createElement('div');
menuItem4.classList.add('menuItem');
menuItem4.innerText = 'компанія';
menuItem4.innerText = 'компанії';
menu.appendChild(menuItem);
menu.appendChild(menuItem2);
menu.appendChild(menuItem3);
menu.appendChild(menuItem4);
header.appendChild(logo);
header.appendChild(menu);
document.body.appendChild(header);
document.body.appendChild(header);
let advertisement = document.createElement('img');
let footer = document.createElement('footer');
let footerText = document.createElement('div');
advertisement.src = 'advertisement.png';
document.body.appendChild(advertisement);
footerText.classList.add('footerText');
footerText.innerText = 'Компанія «Дроби»';
footerText.innerText = 'Компанія «Дрони», м. Київ, ул. Святошина, 1, тел. (044) 123-45-67';
footerText.innerText = 'Компанія «Дроби» розміщена вул. Стрілкова, 1, Київ';
```

```

footer.appendChild(footerText);
document.body.appendChild(footer);
let footerMenu = document.createElement('div');
footerMenu.classList.add('footerMenu');
let footerMenuItem = document.createElement('div');
footerMenuItem.classList.add('footerMenuItem');
footerMenuItem.innerText = 'про нас';
let footerMenuItem2 = document.createElement('div');
footerMenuItem2.classList.add('footerMenuItem');
footerMenuItem2.innerText = 'новини';
let footerMenuItem3 = document.createElement('div');
footerMenuItem3.classList.add('footerMenuItem');
footerMenuItem3.innerText = 'наші партнери';
let footerMenuItem4 = document.createElement('div');
footerMenuItem4.classList.add('footerMenuItem');
footerMenuItem4.innerText = 'компанія';
footerMenuItem4.innerText = 'компанія';
footerMenu.appendChild(footerMenuItem);
footerMenu.appendChild(footerMenuItem);
footerMenu.appendChild(footerMenuItem2);
footerMenu.appendChild(footerMenuItem);

footerMenu.appendChild(footerMenuItem3);
footerMenu.appendChild(footerMenuItem3);
footerMenu.appendChild(footerMenuItem4);
footer.appendChild(footerMenu);
footer.appendChild(footerMenu);
document.body.appendChild(footer);
document.body.appendChild(footer);

```

4. .

```

import re
import unittest

#Створити модульні тести для функції printTable та formatDataFromText.
#Запустити тести.

def test_printTable(self):
 with open("test.txt", "r") as f:
 fileText = f.read()
 dataMap = formatDataFromText(fileText)
 printTable(dataMap)
 self.assertEqual(printTable(dataMap), "name age gender\nJohn 20 male\nJane 19 female\n")

```

*Рис Г.12. Результат виконання 4 тестового прикладу Amazon CodeWhisperer.*



5.

```
/*Покрити тестами функцію wekaAPIRealization на Java за допомогою бібліотеки JUnit */
```

```
public class TestClass{
 @Test
 public static void testWekaAPIRealization() {
 String result = wekaAPIRealization("C:\\Users\\User\\Desktop\\weka-3-8-0\\data\\iris.arff");
 Assertions.assertEquals("Weka regression model.", result);
 Assertions.assertNotEquals("Weka regression model.", result);
 Assertions.assertNotNull(result);
 Assertions.assertNull(result);
 Assertions.assertTrue(result.contains(s:"Weka regression model."));
 Assertions.assertFalse(result.contains(s:"Weka regression model."));
 Assertions.assertNotEquals(0, result.length());
 Assertions.assertEquals(0, result.length());

 Assertions.assertTrue(result.contains(s:"Results on train data:"));
 Assertions.assertFalse(result.contains(s:"Results on train data:"));
 }
}
```

Рис Г.13. Результат виконання 5 тестового прикладу Amazon CodeWhisperer.

6.

```
std::string createPalindrome(int length, std::string str)
//Створити функцію для створення паліндромів вказаної довжини на основі
```

```
std::string createPalindrome(int length, std::string str){
 if (length < 0)
 return "";
 if (length == 0)
 return "";
 if (length == 1)
 return str;
 if (length == 2)
 return str + str[0];
 if (length == 3)
 return str + str[0] + str[1];
 if (length == 4)
 return str + str[0] + str[1] + str[2];
 if (length == 5)
 return str + str[0] + str[1] + str[2] + str[3];
 if (length == 6)
 return str + str[0] + str[1] + str[2] + str[3] + str[4];
}
```

Рис Г.14. Результат виконання 6 тестового прикладу Amazon CodeWhisperer.

7. //Покрити тестами клас HashTable на C++ за допомогою бібліотеки Google Test лише за його хедером

```
class HashTableTest : public ::testing::Test {
```

```

HashTable table;

protected:
 HashTableTest() {}
 virtual void SetUp() {
 table = HashTable(10);
 }

 virtual void TearDown() {
 table = HashTable(10);
 }
}

TEST_F(HashTableTest, TestAdd) {
 HashTable table(10);
 table.add(L"key", { L"value" });
 ASSERT_EQ(table.getCount(), 1);
 ASSERT_EQ(table.getSize(), 1);
 ASSERT_EQ(table.getCapacity(), 10);
 ASSERT_EQ(table.getPercent(), 0.1);
 table.add(L"key", { L"value" });
 table.add(L"key", { L"value" });
 ASSERT_EQ(table.getCount(), 1);
 ASSERT_EQ(table.getSize(), 1);
 ASSERT_EQ(table.getCapacity(), 10);
 ASSERT_EQ(table.getPercent(), 0.1);
 ASSERT_EQ(table.get(L"key").size(), 2);
}

TEST_F(HashTableTest, TestGet) {
 HashTable table(10);
 table.add(L"key", { L"value" });
 table.add(L"key", { L"value" });
 table.get(L"key");
 ASSERT_EQ(table.getCount(), 1);
 ASSERT_EQ(table.getSize(), 1);
 ASSERT_EQ(table.getCapacity(), 10);
 ASSERT_EQ(table.getPercent(), 0.1);
 ASSERT_EQ(table.get(L"key").size(), 2);
 table.get(L"key");
 table.get(L"key");
 table.get(L"key");
}

TEST_F(HashTableTest, TestGetEmpty) {

```

```

 HashTable table(10);
 table.get(L"key");
 ASSERT_EQ(table.getCount(), 0);
 ASSERT_EQ(table.getSize(), 0);
 ASSERT_EQ(table.getCapacity(), 10);
 ASSERT_EQ(table.getPercent(), 0);
 ASSERT_EQ(table.get(L"key").size(), 0);
 table.get(L"key");
 table.get(L"key");
 table.get(L"key");
}

```

```

TEST_F(HashTableTest, TestGetEmpty2) {
 HashTable table(10);
 table.get(L"key");
 table.get(L"key");
 //...
}

```

```

TEST_F(HashTableTest, TestIncreaseTable) {
 HashTable table(10);
 for(int i = 0; i < 10; i++){
 table.add(L"key", { L"value" });
 }
 table.increaseTable();
 ASSERT_EQ(table.getCount(), 10);
 ASSERT_EQ(table.getSize(), 10);
 ASSERT_EQ(table.getCapacity(), 20);
 ASSERT_EQ(table.getPercent(), 0.2);
 table.increaseTable();
 table.increaseTable();
}

```

```

TEST_F(HashTableTest, TestSaveTable) {
 HashTable table(10);
 for(int i = 0; i < 10; i++){
 table.add(L"key", { L"value" });
 }
 table.increaseTable();
 table.saveTable(L"test.txt");
 ASSERT_EQ(table.getCount(), 10);
 ASSERT_EQ(table.getSize(), 10);
 ASSERT_EQ(table.getCapacity(), 20);
 ASSERT_EQ(table.getPercent(), 0.2);
 ASSERT_EQ(table.get(L"key").size(), 2);
}

```

```

File file = openFile(L"test.txt");
ASSERT_EQ(readNextLine(), L"key");
ASSERT_EQ(readNextLine(), L"value");
file.close();
removeFile(L"test.txt");
}

```

8. //Показати приклад взаємодії класів HashTable та ThreadPool і вказати, у яких випадках вони могли б використовуватися разом

```

void testHashTableAndThreadPool(){
 HashTable table(10);
 table.add(L"key", { L"value" });
 table.add(L"key", { L"value" });
 table.add(L"key", { L"value" });

 ThreadPool<HashTable> threadPool([](HashTable table) {
 return table.increaseTable();
 }, 2);

 threadPool.addTask(table);

 threadPool.joinAll();

 ASSERT_EQ(table.getCount(), 1);
 ASSERT_EQ(table.getSize(), 1);
 ASSERT_EQ(table.getCapacity(), 10);
 ASSERT_EQ(table.getPercent(), 0.1);
 ASSERT_EQ(table.get(L"key").size(), 3);

 table.saveTable(L"table.txt");
 table = HashTable(10);
 table.loadTable(L"table.txt");

 ASSERT_EQ(table.getCount(), 1);
 ASSERT_EQ(table.getSize(), 1);
 ASSERT_EQ(table.getCapacity(), 10);
 ASSERT_EQ(table.getPercent(), 0.1);
 ASSERT_EQ(table.get(L"key").size(), 3);

 threadPool.addTask(table);
 threadPool.joinAll();

 ASSERT_EQ(threadPool.getBusyThreadCount(), 0);
 ASSERT_EQ(table.getCount(), 1);
 ASSERT_EQ(table.getSize(), 1);
}

```

```

 ASSERT_EQ(table.getCapacity(), 10);
 ASSERT_EQ(table.getPercent(), 0.1);
 ASSERT_EQ(table.get(L"key").size(), 3);

 table.saveTable(L"table.txt");
 table = HashTable(10);
}

```

9.

```

class HopfieldNetworkTest {
 @org.junit.jupiter.api.Test
 void test1() {
 ArrayList<Integer> set1 = new ArrayList<>(Array.asList(1,1,1,1));
 ArrayList<Integer> set2 = new ArrayList<>(Array.asList(1,1,1,1));
 HopfieldNetwork hn = new HopfieldNetwork();
 hn.train(set1);
 ArrayList<Integer> result = hn.test(set2);
 assert (result.equals(set2));
 assert (hn.sampleSize() == 1);
 assert (hn.getInstance(index:0).equals(set1));
 assert (hn.getInstance(index:0).equals(set2));
 assert (hn.getInstance(index:0).equals(result));
 assert (hn.getInstance(index:0).equals(hn.test(hn.getInstance(index:0))));
 }
 @org.junit.jupiter.api.Test
 void < 1/1 > Accept [Tab] Accept Word [Ctrl] + [RightArrow] ...
 ArrayList<Integer> set1 = new ArrayList<>(Array.asList(1,1,1,1));
 ArrayList<Integer> set2 = new ArrayList<>(Array.asList(1,1,1,1));
 HopfieldNetwork hn = new HopfieldNetwork();
 hn.train(set1);
 ArrayList<Integer> result = hn.test(set2);
 assert (result.equals(set2));
 assert (hn.sampleSize() == 1);
 assert (hn.getInstance(0).equals(set1));
 assert (hn.getInstance(0).equals(set2));
 assert (hn.getInstance(0).equals(result));
 assert (hn.getInstance(0).equals(hn.test(hn.getInstance(0))));
 }
}

```



**Рис Г.15. Результат виконання 9 тестового прикладу Amazon CodeWhisperer.**

5. Додаток Д. Діаграма прецедентів UML власного фреймворку.pdf

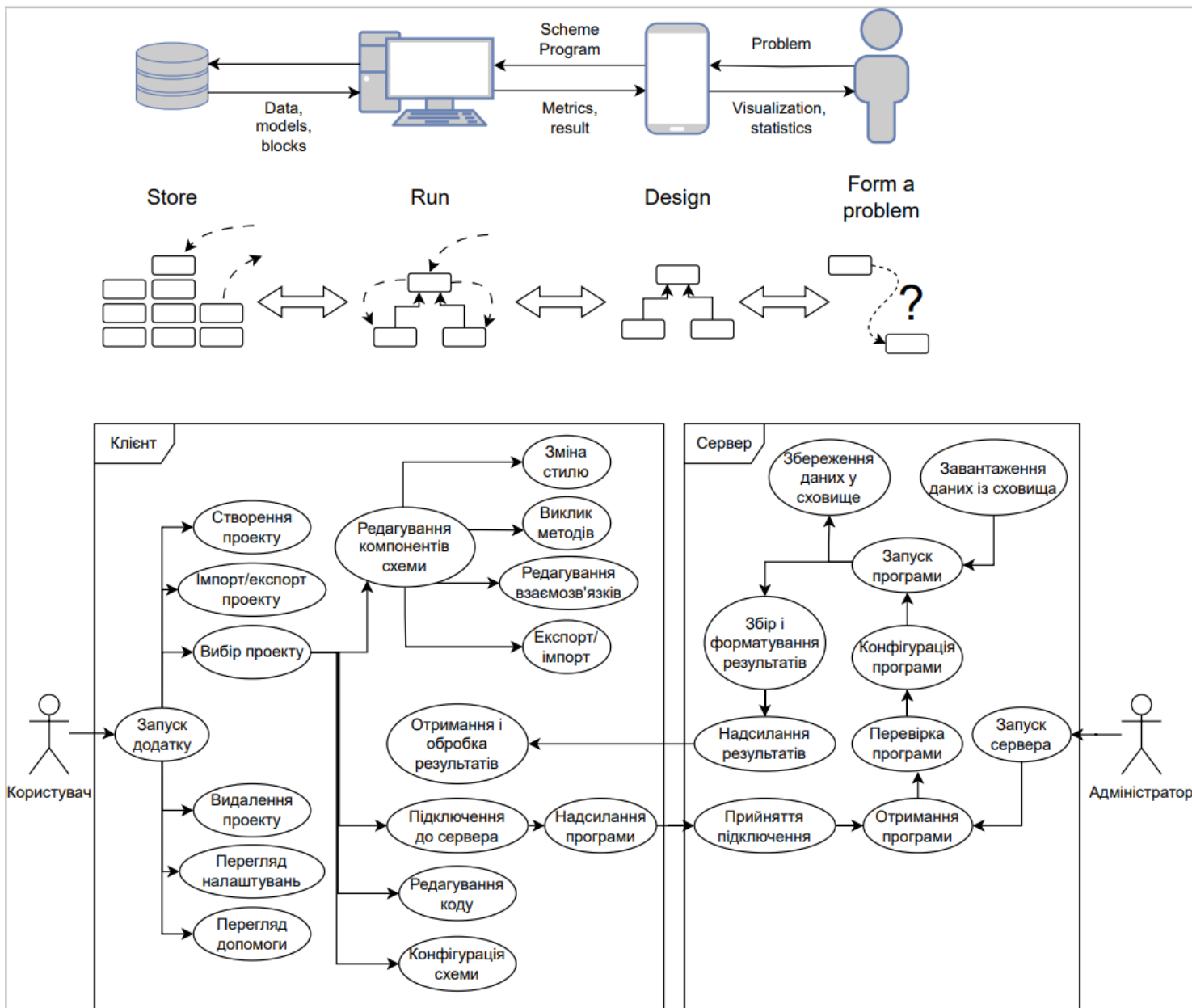


Рис. Д.1. Вміст файлу Діаграма прецедентів UML власного фреймворку.pdf.

6. Додаток Е. Діаграма класів UML власного фреймворку.pdf

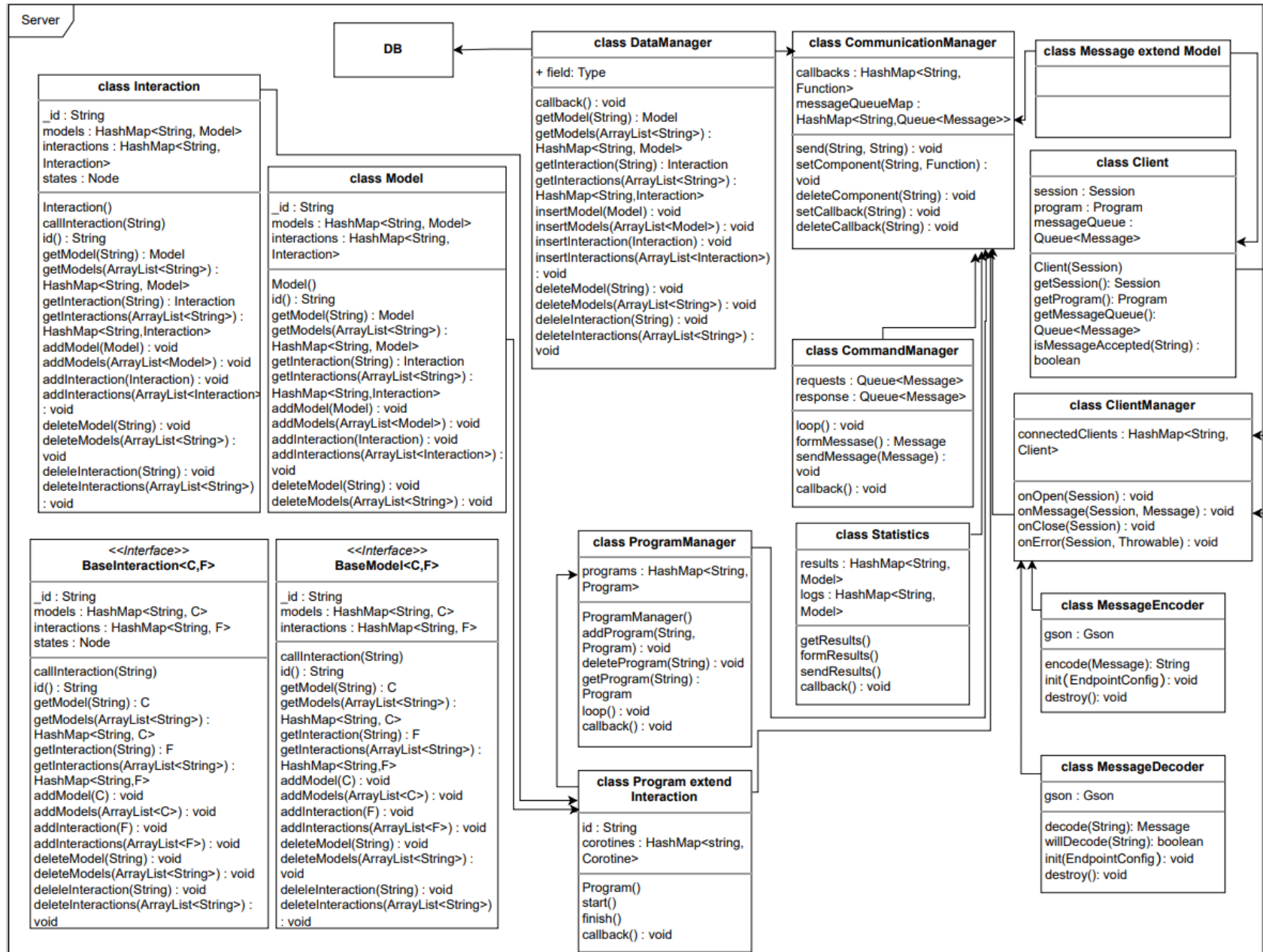


Рис. Е.1. Вміст файлу Діаграма класів UML власного фреймворку.pdf (ліва частина).



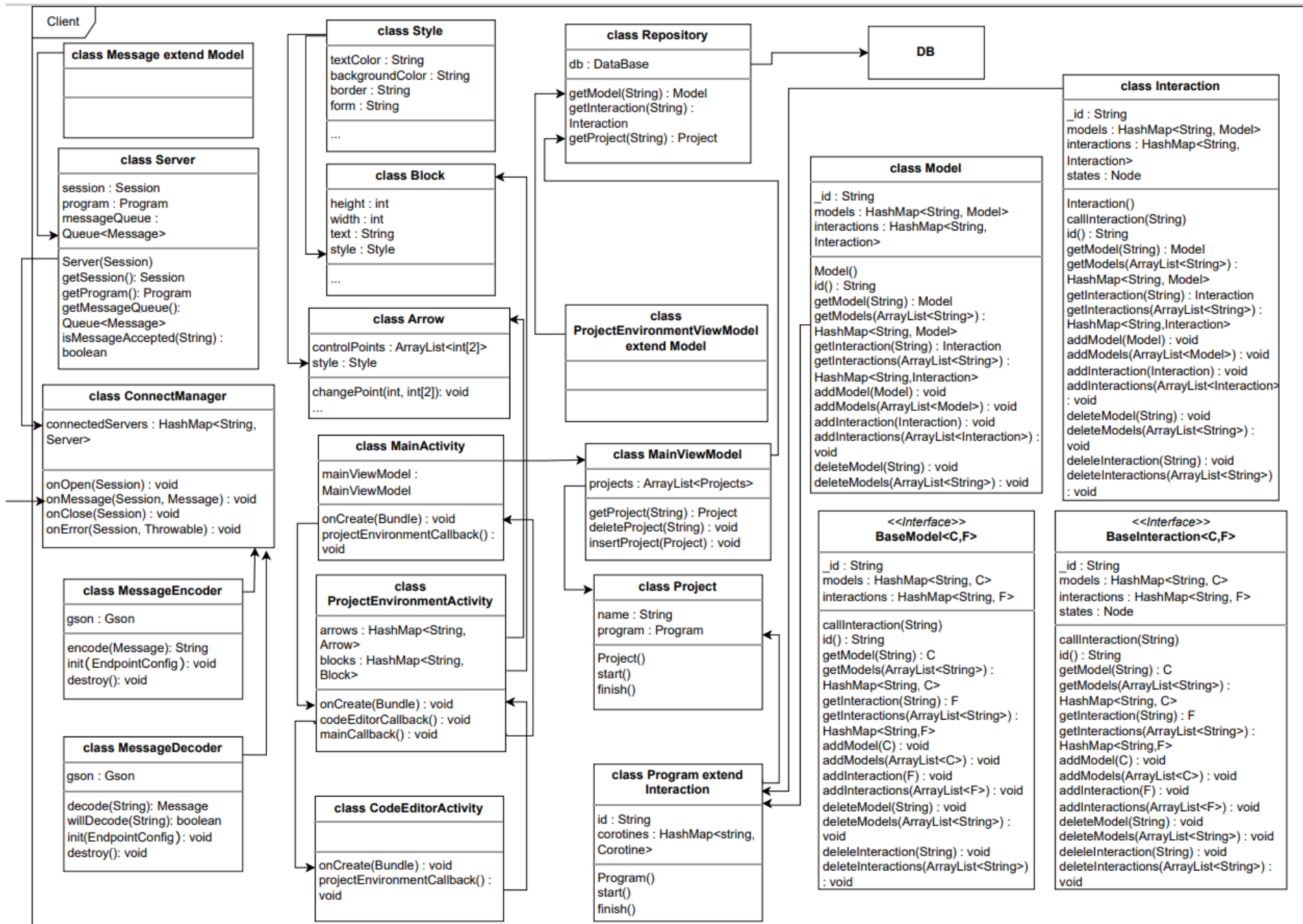


Рис. Е.2. Вміст файлу Діаграма класів UML власного фреймворку.pdf (права частина)



7. Додаток Є. Діаграма станів UML циклу життя задачі.pdf

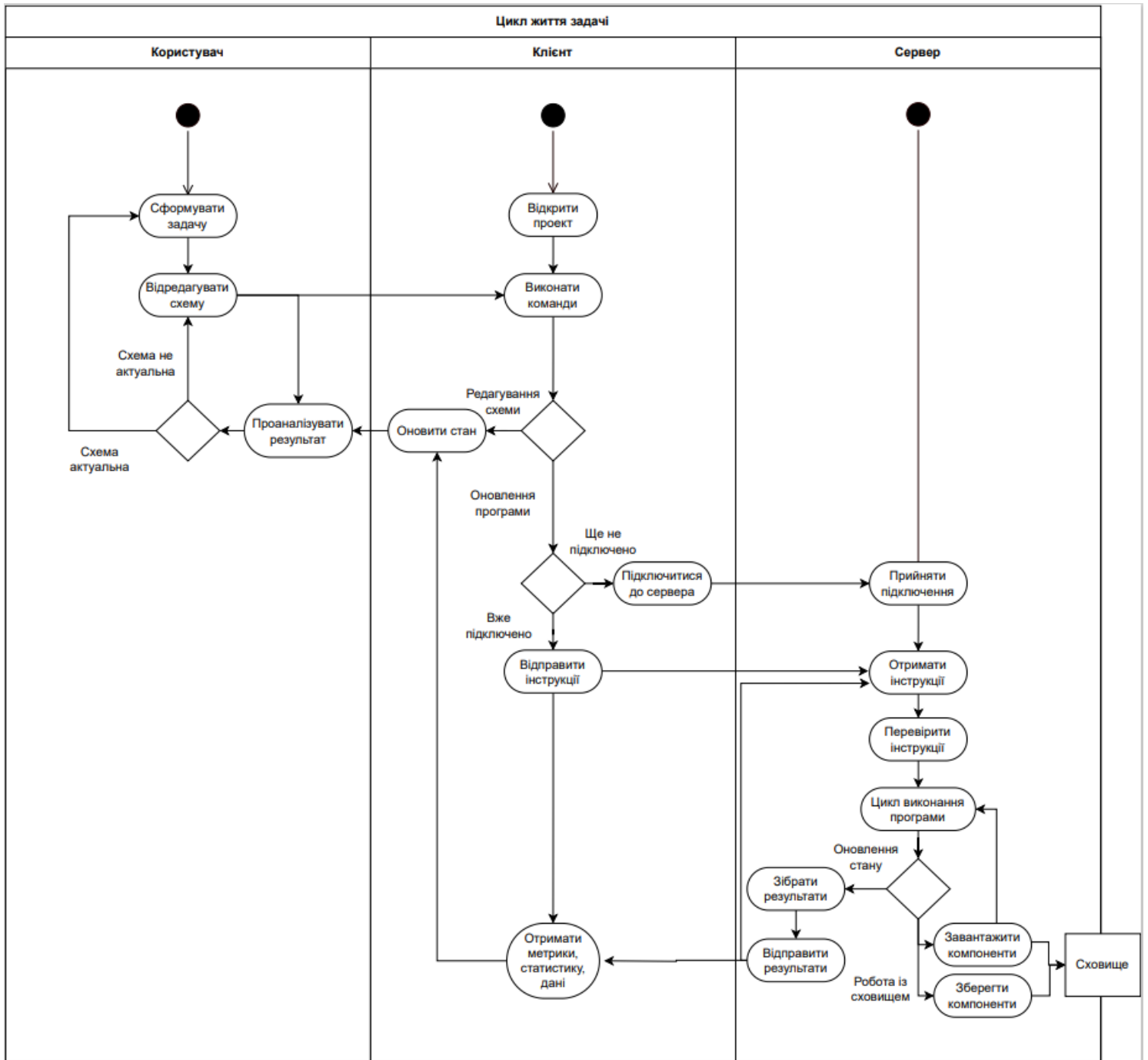
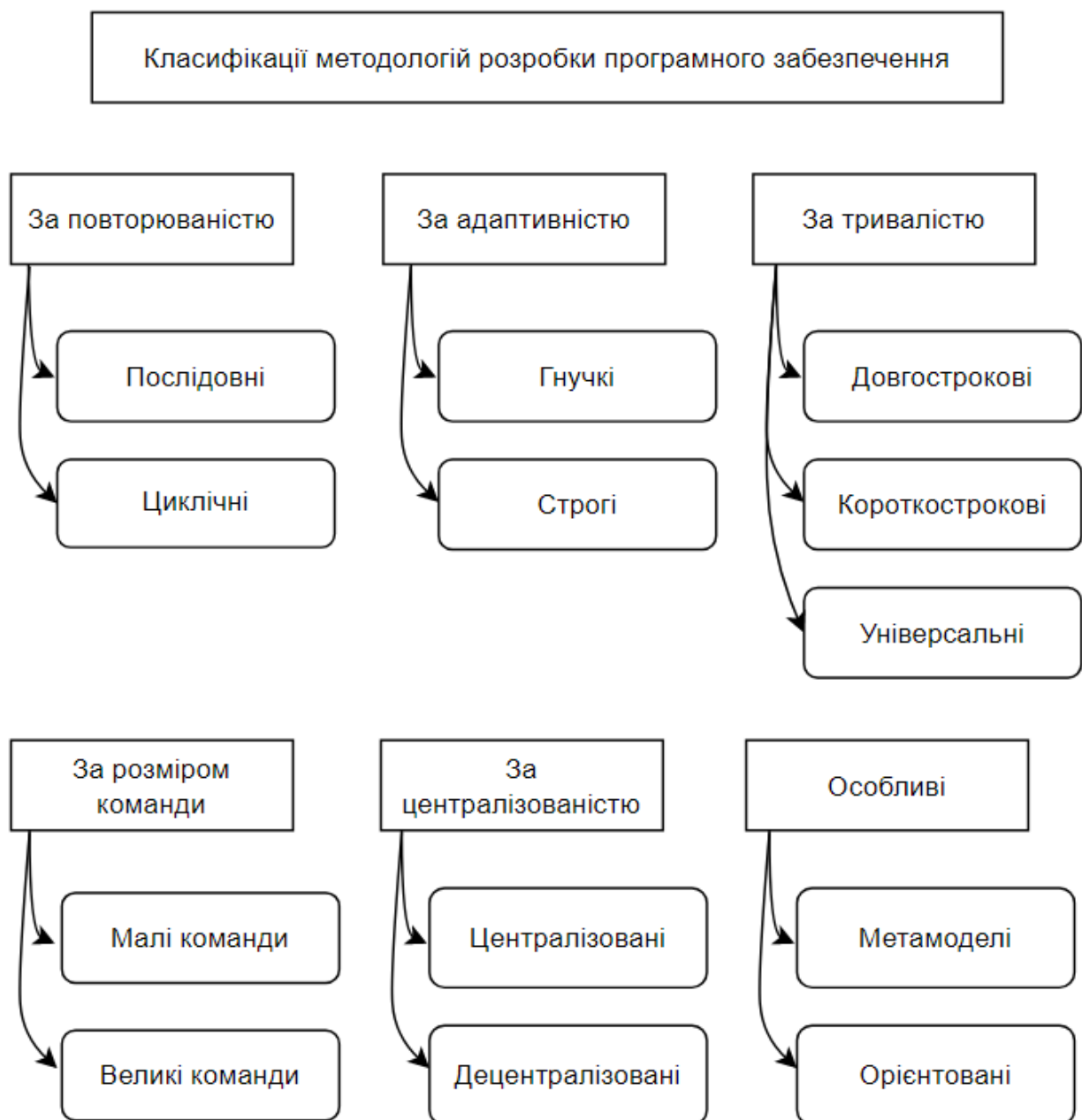


Рис. Є.1. Вміст файлу Діаграма станів UML циклу життя задачі.pdf.

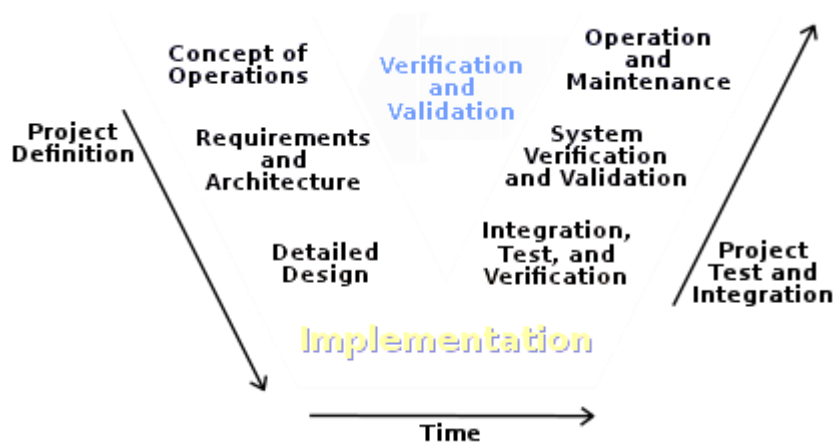
8. Додаток Ж. Малюнки, таблиці.docx



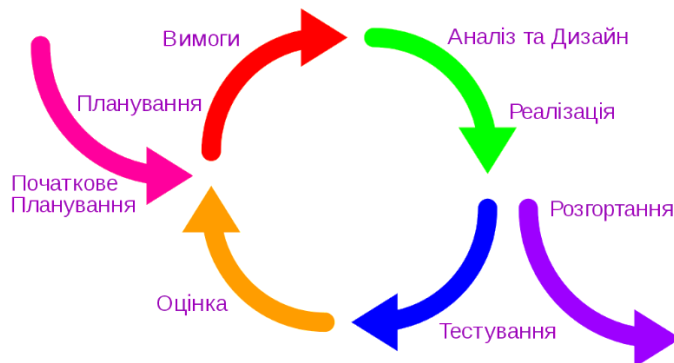
*Рис. 1.1. Власна класифікація МРПЗ.*



*Рис. 1.2. Форма ЖЦПЗ Waterfall.*



*Рис. 1.3. Форма ЖЦПЗ V-Model.*



*Рис. 1.4. Форма ЖЦПЗ Iterative/Incremental.*

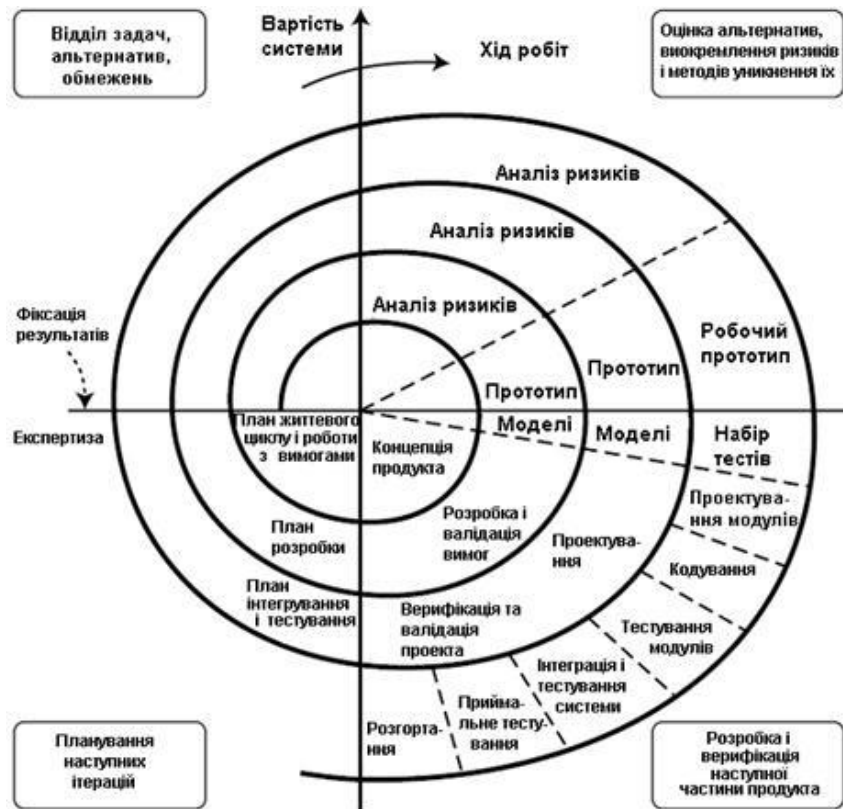


Рис. 1.5. Форма ЖЦПЗ Spiral.



Рис. 1.6. Форма життєвого циклу проекту DSDM.

### Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.

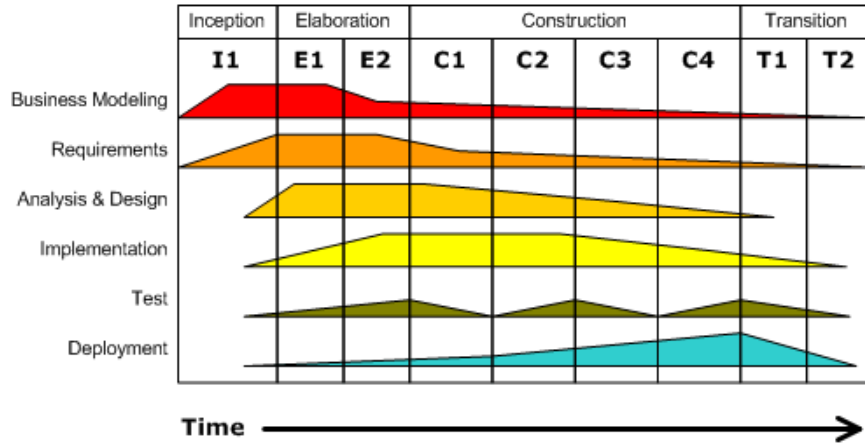


Рис. 1.7. Форма ЖЦПЗ RUP.

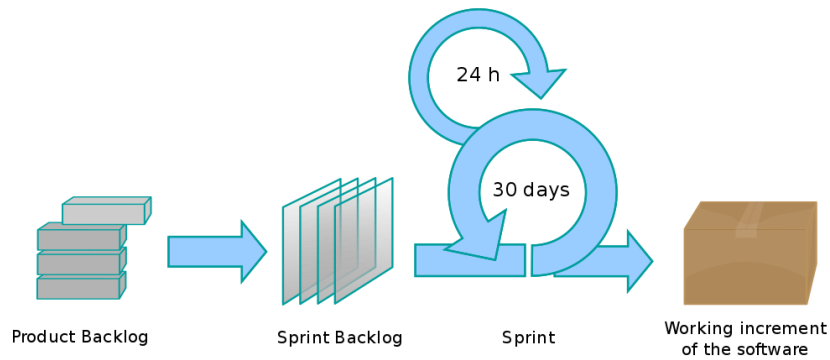


Рис. 1.8. Видял Scrum процесу.

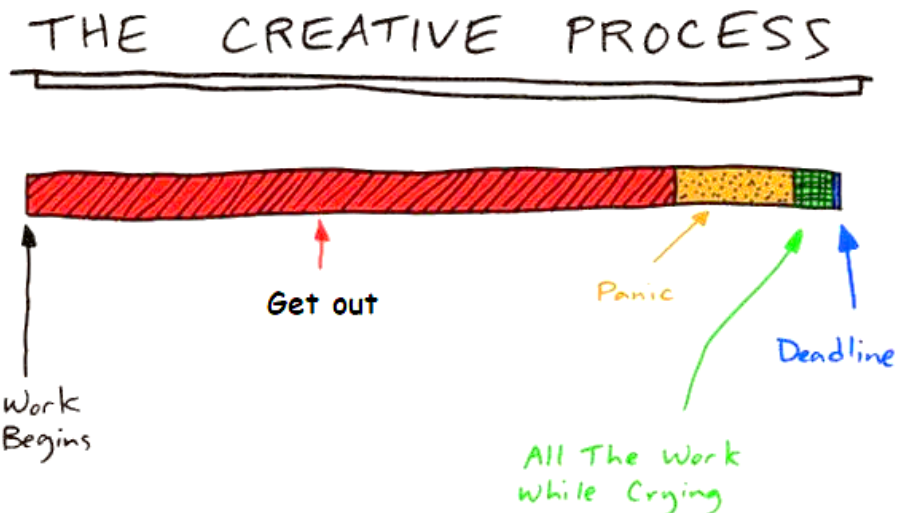


Рис. 1.9. Форма ЖЦПЗ PDD.<sup>67</sup>

<sup>67</sup> TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт. URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).

Таблиця 1.1. Розподілені методології за критеріями поділу, класифікації.

За повторюваністю	Послідовні	Waterfall, V-Model, PDD
	Циклічні	Iterative та Incremental, Spiral, DSDM, RUP, Agile, Scrum, Kanban, Lean, Test DD, BDD, FDD
За адаптивністю	Гнучкі	Iterative та Incremental, Spiral, DSDM, RUP, Agile, Scrum, Kanban, Lean, Test DD, BDD, FDD, ADD
	Строгі	Waterfall, V-Model
За тривалістю	Довгострокові	Spiral, RUP
	Короткострокові	V-Model, DSDM, Kanban.
	Універсальні	Waterfall, Iterative та Incremental, Agile, Scrum, Lean
За розміром команди	Малі команди	Agile, Scrum, Kanban
	Великі команди	V-Model, Iterative та Incremental, Spiral, DSDM, RUP, Lean
За централізованістю	Централізовані	Waterfall, V-Model, Spiral, DSDM, RUP,
	Децентралізовані	Agile, Scrum, Kanban, Lean
Особливі	Метамоделі	Spiral
	Орієнтовані	Type DD, BDD, DDD, FDD, MDD, PDD, ADD



Рис. 1.10. Шкала відносних оцінок збіжності методологій та сценаріїв використання

```

---;
MDD;
PDD;
ADD
&
$Methodology paterns
&
якість | деталізація | ризики | контрольованість | процес розробки | економія | час | команда | зворотній зв'язок | стійкість до змін;
якість | ризики | деталізація | економія | контрольованість | процес розробки | команда | час | стійкість до змін | зворотній зв'язок;
стійкість до змін | час | контрольованість | процес розробки | ризики | якість | команда | деталізація | економія | зворотній зв'язок;
ризики | процес розробки | стійкість до змін | контрольованість | деталізація | час | команда | якість | економія | зворотній зв'язок;
час | економія | зворотній зв'язок | якість | деталізація | контрольованість | процес розробки | ризики | команда | стійкість до змін;
ризики | економія | деталізація | якість | процес розробки | команда | контрольованість | час | стійкість до змін | зворотній зв'язок;
час | команда | стійкість до змін | процес розробки | зворотній зв'язок | економія | якість | деталізація | контрольованість | ризики;
процес розробки | команда | час | зворотній зв'язок | стійкість до змін | якість | деталізація | економія | ризики | контрольованість;
час | економія | команда | стійкість до змін | якість | деталізація | процес розробки | ризики | контрольованість | зворотній зв'язок;
економія | якість | ризики | час | команда | зворотній зв'язок | контрольованість | процес розробки | деталізація | стійкість до змін;
якість | час | стійкість до змін | деталізація | економія | зворотній зв'язок | ризики | контрольованість | процес розробки | команда;
якість | економія | деталізація | стійкість до змін | час | команда | процес розробки | контрольованість | ризики | зворотній зв'язок;
якість | час | стійкість до змін | деталізація | економія | зворотній зв'язок | ризики | контрольованість | процес розробки | команда;
час | економія | команда | контрольованість | стійкість до змін | процес розробки | деталізація | якість | ризики | зворотній зв'язок;
процес розробки | час | деталізація | якість | контрольованість | стійкість до змін | команда | ризики | економія | зворотній зв'язок;
час | процес розробки | економія | контрольованість | стійкість до змін | якість | команда | ризики | деталізація | зворотній зв'язок;
команда | процес розробки | зворотній зв'язок | деталізація | стійкість до змін | контрольованість | економія | час | ризики | якість;
час | якість | стійкість до змін | економія | контрольованість | ризики | зворотній зв'язок | деталізація | процес розробки | команда
&
$Methodology classification
&
Waterfall | маленька команда | велика команда | малий проект | великий проект | короткостроковий проект | довгостроковий
V-Model | велика команда | | великий проект | | короткостроковий проект
Iterative та Incremental | велика команда | | малий проект | великий проект | короткостроковий проект | довгостроковий
Spiral | велика команда | | великий проект | | довгостроковий проект
DSDM | велика команда | | малий проект | великий проект | короткостроковий проект

```

Рис. 1.11. Фрагмент файлу із даними для порівняння у власному форматі

```

def calculatePatternConvergence(pattern1, pattern2):
 set1 = set(getAllSubsequences(pattern1))
 set2 = set(getAllSubsequences(pattern2))
 intersection = set1.intersection(set2)
 if len(intersection) == 0:
 longest = 0
 else:
 longest = len(sorted(intersection, key=len)[-1])
 return longest

def calculateFeatureIntersectionPower(pattern1, pattern2):
 return len(set(pattern1).intersection(set(pattern2)))

def calculateRate(patternConvergence, featureIntersectionPower):
 ratesDiscreteDistribution = [1,1,1,2,2,3,4,4,5,5,5]
 PC = patternConvergence
 FIP = featureIntersectionPower
 discreteDistributionPosition = round((PC * (10/7)*FIP)**0.5)
 DDP = discreteDistributionPosition
 return ratesDiscreteDistribution[DDP]

```

*Рис. 1.12. Фрагмент програми для порівняння, де показано функції для обрахунку PC, FIP, DDP*

Методологія \ Сценарій	1	2	3	4	Малі
Lean	Green	Yellow	Yellow	Orange	12
Waterfall	Yellow	Orange	Yellow	Green	12
Scrum	Yellow	Green	Yellow	Orange	12
Agile	Orange	Yellow	Green	Orange	11
Kanban	Yellow	Green	Orange	Orange	11
Test DD	Orange	Yellow	Yellow	Orange	10
Iterative та Incremental	Yellow	Yellow	Orange	Orange	10
V-Model	Orange	Orange	Orange	Yellow	9
RUP	Orange	Red	Yellow	Yellow	9
Spiral	Orange	Orange	Yellow	Orange	9
DSDM	Yellow	Orange	Orange	Orange	9

*Рис. 1.13. Порівняння загальних методологій для сценаріїв із малими командами*

Методологія \ Сценарій	5	6	7	8	Великі
Lean	Green	Green	Green	Green	16
Waterfall	Green	Yellow	Green	Green	15
V-Model	Green	Yellow	Green	Green	15
RUP	Green	Green	Green	Yellow	15
Spiral	Green	Green	Yellow	Yellow	14
DSDM	Yellow	Green	Green	Yellow	14
Test DD	Green	Yellow	Yellow	Yellow	13
Iterative та Incremental	Green	Yellow	Yellow	Orange	12
Agile	Orange	Yellow	Orange	Orange	9
Kanban	Green	Orange	Orange	Red	9
Scrum	Orange	Orange	Orange	Orange	8

Рис. 1.14. Порівняння загальних методологій для сценаріїв із великими командами

Методологія \ Сценарій	1	2	3	4	5	6	7	8	Всього
Lean	Green	Yellow	Yellow	Orange	Green	Green	Green	Green	28
Waterfall	Yellow	Orange	Yellow	Green	Green	Yellow	Green	Green	27
V-Model	Orange	Orange	Orange	Yellow	Green	Yellow	Green	Green	24
RUP	Orange	Red	Yellow	Yellow	Green	Green	Green	Yellow	24
Spiral	Orange	Orange	Yellow	Orange	Green	Green	Yellow	Yellow	23
DSDM	Yellow	Orange	Orange	Orange	Yellow	Green	Green	Yellow	23
Test DD	Orange	Yellow	Yellow	Orange	Green	Yellow	Yellow	Yellow	23
Iterative та Incremental	Yellow	Yellow	Orange	Orange	Green	Yellow	Yellow	Orange	22
Agile	Orange	Yellow	Green	Orange	Orange	Yellow	Orange	Orange	20
Kanban	Yellow	Green	Orange	Orange	Green	Orange	Orange	Red	20
Scrum	Yellow	Green	Yellow	Orange	Orange	Orange	Orange	Orange	20

Рис. 1.15. Порівняння загальних методологій для усіх сценаріїв

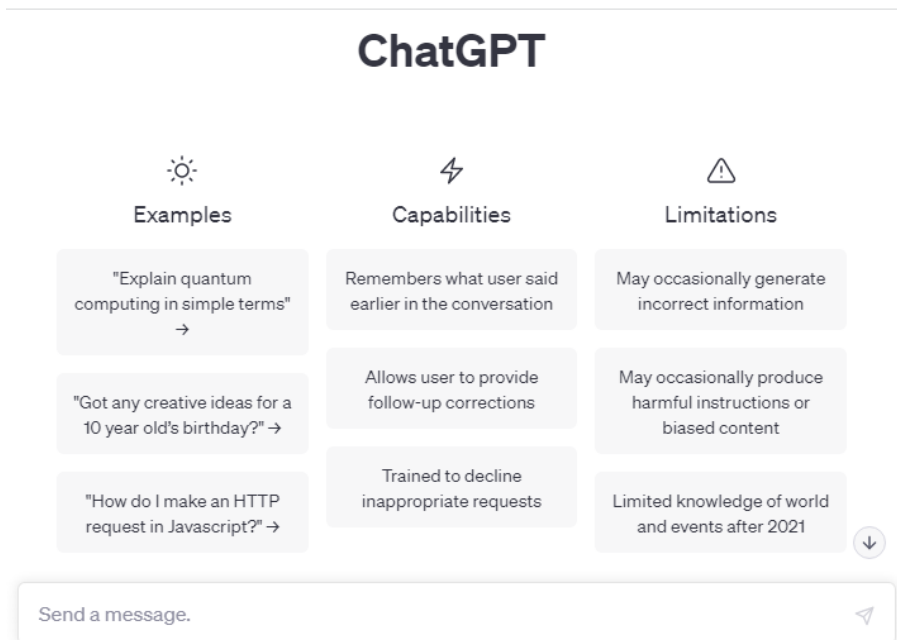


Методологія \ Сценарій	1	2	3	4	5	6	7	8	Всього
ADD	Green	Yellow	Green	Yellow	Green	Green	Green	Green	30
Type DD	Yellow	Yellow	Green	Yellow	Green	Green	Green	Yellow	28
DDD	Green	Yellow	Yellow	Yellow	Yellow	Green	Yellow	Yellow	26
FDD	Yellow	Orange	Yellow	Yellow	Yellow	Green	Green	Yellow	26
BDD	Orange	Yellow	Yellow	Orange	Green	Yellow	Yellow	Yellow	23
Test DD	Orange	Yellow	Yellow	Orange	Green	Yellow	Yellow	Yellow	23
MDD	Yellow	Yellow	Yellow	Yellow	Yellow	Yellow	Orange	Orange	22
PDD	Yellow	Yellow	Orange	Yellow	Red	Red	Orange	Orange	17

*Рис. 1.16. Порівняння орієнтованих методологій для усіх сценаріїв*

Методологія \ Сценарій	1	2	3	4	5	6	7	8	Всього
ADD	Green	Yellow	Green	Yellow	Green	Green	Green	Green	30
Type DD	Yellow	Yellow	Green	Yellow	Green	Green	Green	Yellow	28
Lean	Green	Yellow	Yellow	Orange	Green	Green	Green	Green	28
Waterfall	Yellow	Orange	Yellow	Green	Green	Yellow	Green	Green	27
DDD	Green	Yellow	Yellow	Yellow	Yellow	Green	Yellow	Yellow	26
FDD	Yellow	Orange	Yellow	Yellow	Yellow	Green	Green	Yellow	26
V-Model	Orange	Orange	Orange	Yellow	Green	Yellow	Green	Green	24
RUP	Orange	Red	Yellow	Yellow	Green	Green	Green	Yellow	24
BDD	Orange	Yellow	Yellow	Orange	Green	Yellow	Yellow	Yellow	23
Spiral	Orange	Orange	Yellow	Orange	Green	Green	Yellow	Yellow	23
Test DD	Orange	Yellow	Yellow	Orange	Green	Yellow	Yellow	Yellow	23
DSDM	Yellow	Orange	Orange	Orange	Yellow	Green	Green	Yellow	23
MDD	Yellow	Yellow	Yellow	Yellow	Yellow	Yellow	Orange	Orange	22
Iterative та Incremental	Yellow	Yellow	Orange	Orange	Green	Yellow	Yellow	Orange	22
Agile	Orange	Yellow	Green	Orange	Orange	Yellow	Orange	Orange	20
Kanban	Yellow	Green	Orange	Orange	Green	Orange	Orange	Red	20
Scrum	Yellow	Green	Yellow	Orange	Orange	Orange	Orange	Orange	20
PDD	Yellow	Yellow	Orange	Yellow	Red	Red	Orange	Orange	17

*Рис. 1.17. Порівняння усіх методологій для усіх сценаріїв*



**Рис. 3.1. Інтерфейс онлайн сервісу ChatGPT <sup>68</sup>**



**Рис. 3.2. Інтерфейс онлайн сервісу AgentGPT <sup>69</sup>**

<sup>68</sup> Introducing ChatGPT – OpenAI. OpenAI : веб-сайт. URL: <https://openai.com/blog/chatgpt> (дата звернення: 12.05.2023).

<sup>69</sup> AgentGPT. AgentGPT, reworkd : веб-сайт. URL: <https://agentgpt.reworkd.ai/uk> (дата звернення: 12.05.2023).

## Build me a ChatGPT-powered app that...

...writes me a poem for my birthday

Create 🤖

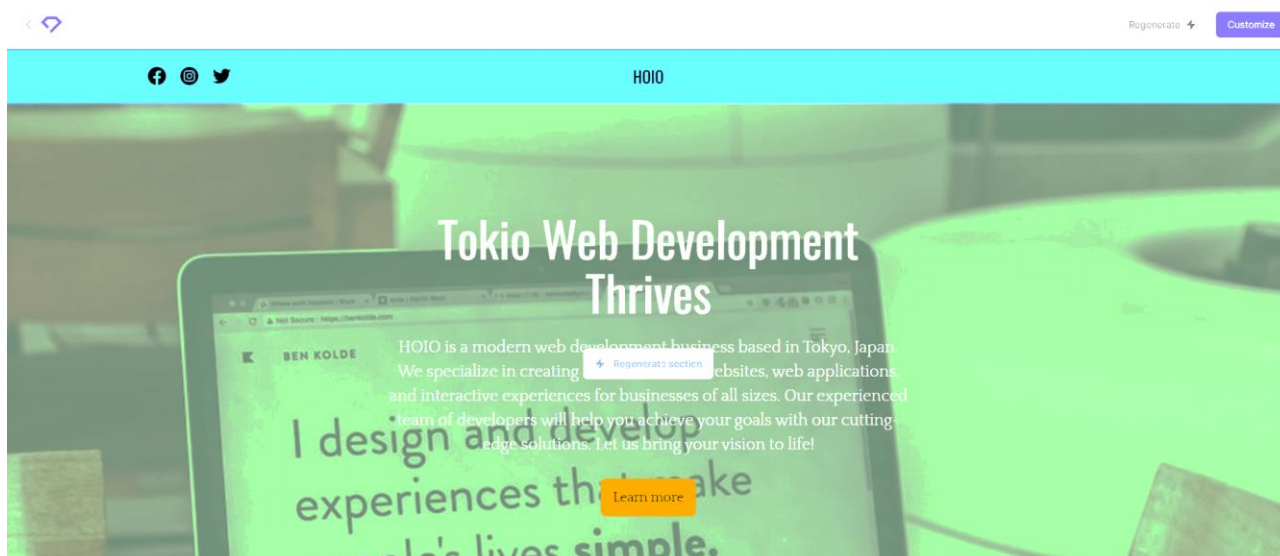
### Ideas

- Translates a word to Korean, Japanese and French
- Proof read my text
- Gives advice on your startup idea from Paul Graham
- Writes me a poem for my birthday
- Children's story generator

### Tips for creating valuable web apps with Pico:

- Provide a clear and concise description of your app
- Once you click "Create", you'll be able to iterate on your app. The best Picos are created after several iterations.
- Pico is best for single page web apps, so avoid mobile app ideas or complex applications

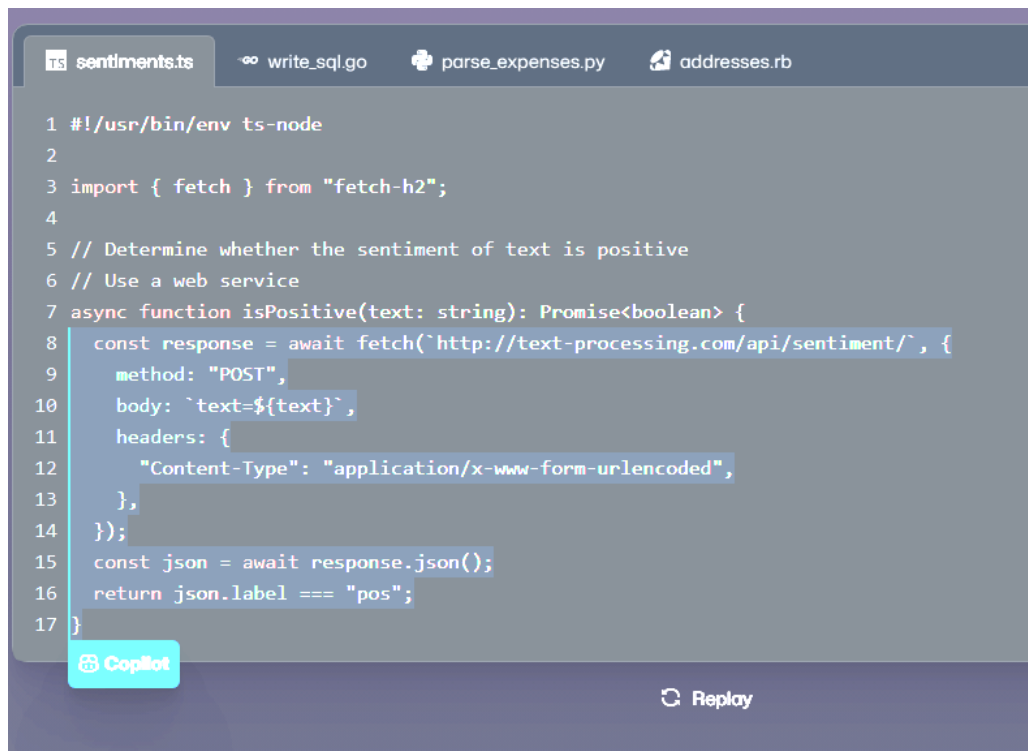
**Рис. 3.3. Інтерфейс онлайн сервісу MetaGPT <sup>70</sup>**



**Рис. 3.4. Приклад роботи онлайн сервісу Durable. Верхня частина сайту. <sup>71</sup>**

<sup>70</sup> MetaGPT. Metaidt : веб-сайт. URL: <https://www.metaidt.com/upgrade-your-account/> (дата звернення: 12.05.2023).

<sup>71</sup> Durable AI Website Builder and Small Business Software. Durable : веб-сайт. URL: <https://durable.co/> (дата звернення: 12.05.2023).

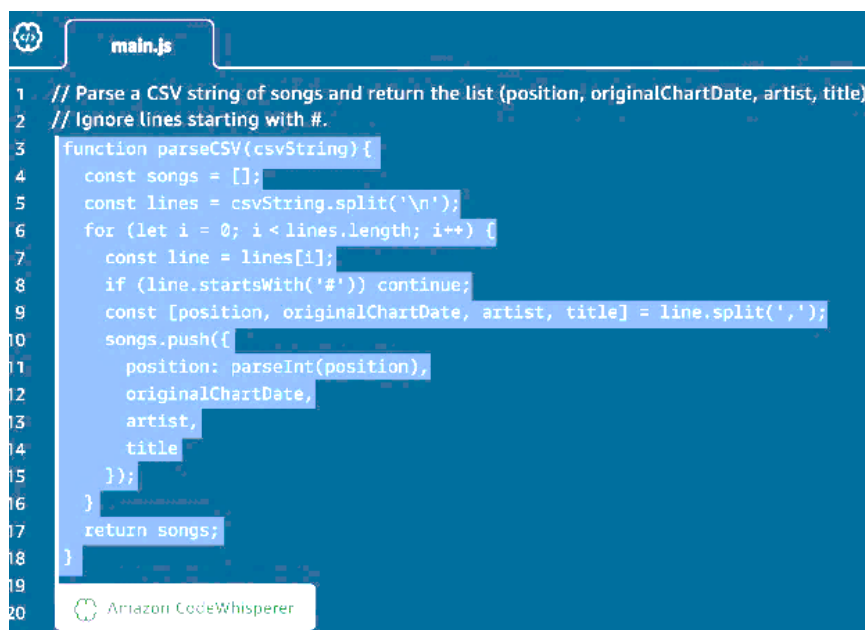


```
1 #!/usr/bin/env ts-node
2
3 import { fetch } from "fetch-h2";
4
5 // Determine whether the sentiment of text is positive
6 // Use a web service
7 async function isPositive(text: string): Promise<boolean> {
8 const response = await fetch(`http://text-processing.com/api/sentiment/`, {
9 method: "POST",
10 body: `text=${text}`,
11 headers: {
12 "Content-Type": "application/x-www-form-urlencoded",
13 },
14 });
15 const json = await response.json();
16 return json.label === "pos";
17 }
```

Copilot

Replay

Рис. 3.5. Приклад роботи GitHub Copilot на офіційній сторінці.<sup>72</sup>



```
1 // Parse a CSV string of songs and return the list (position, originalChartDate, artist, title).
2 // Ignore lines starting with #.
3 function parseCSV(csvString) {
4 const songs = [];
5 const lines = csvString.split('\n');
6 for (let i = 0; i < lines.length; i++) {
7 const line = lines[i];
8 if (line.startsWith('#')) continue;
9 const [position, originalChartDate, artist, title] = line.split(',');
10 songs.push({
11 position: parseInt(position),
12 originalChartDate,
13 artist,
14 title
15 });
16 }
17 return songs;
18 }
19
20
```

Amazon CodeWhisperer

Рис. 3.6. Приклад роботи Amazon CodeWhisperer на офіційній сторінці.<sup>73</sup>

<sup>72</sup> GitHub Copilot · Your AI pair programmer. GitHub : веб-сайт. URL: <https://github.com/features/copilot> (дата звернення: 12.05.2023).

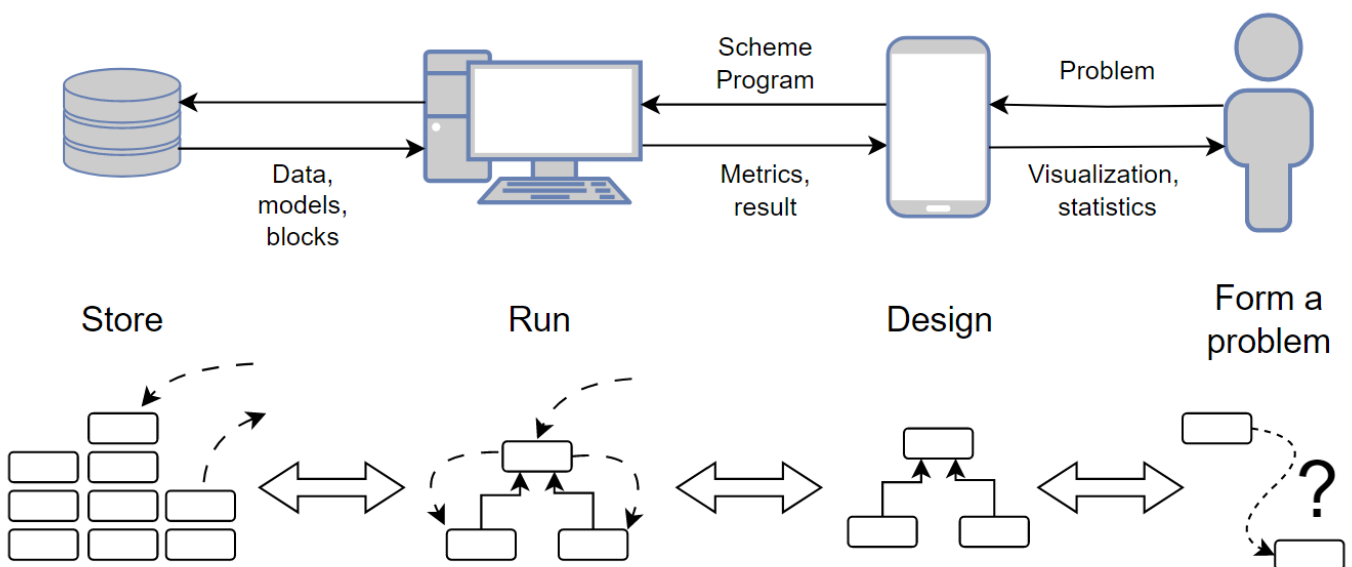
<sup>73</sup> AI Code Generator - Amazon CodeWhisperer - AWS. Amazon : веб-сайт. URL: <https://aws.amazon.com/ru/codewhisperer/> (дата звернення: 12.05.2023).

Таблиця 3.1. Результати перевірки фреймворків створення ПЗ.

Тести\Фреймворки	ChatGPT	AgentGPT	Durable	CodeWhisperer
1. BFS(){?}:	<b>2 / 4</b>	-	-	<b>4 / 4</b>
1) Додано структуру Node;	<b>+</b>			<b>+</b>
2) реалізовано через рекурсію;	<b>+</b>			<b>+</b>
3) функція видає значення цільового вузла;	-			<b>+</b>
4) завершене рішення.	-			<b>+</b>
2. Сайт із розрахунку ходів для шахів:	<b>1 / 7</b>	-	<b>1 / 7</b>	<b>0 / 7</b>
1) Структура сайту.	-		<b>+</b>	-
2) релевантний контент.	-		-	-
3) функціонал сайту.	-		-	-
4) алгоритм пошуку ходів.	<b>+</b>		-	-
5) інтерактивна дошка.	-		-	-
6) рейтингова таблиця.	-		-	-
7) завершене рішення.	-		-	-
3. Сайт компанії для продажу дронів:	<b>1 / 4</b>	-	<b>2 / 4</b>	<b>1 / 4</b>
1) Структура сайту.	<b>+</b>		<b>+</b>	<b>+</b>
2) релевантний контент.	-		<b>+</b>	-
3) функціонал сайту.	-		-	-
4) завершене рішення.	-		-	-
4. prinTable(), formatDataFromText() <- unittest:	<b>5 / 6</b>	-	-	<b>3 / 6</b>
1) Використання бібліотеки модульного тестування;	<b>+</b>			<b>+</b>
2) перехоплення виводу;	<b>+</b>			-
3) формування даних для функції format...();	<b>+</b>			-
4) взаємодія двох функцій;	<b>+</b>			<b>+</b>
5) перевірка проміжних значень;	<b>+</b>			<b>+</b>
6) завершене рішення.	-			-
5. wekaAPIRealization() <- JUnit:	<b>3 / 8</b>	-	-	<b>4 / 8</b>

1)	Розуміння призначення бібліотеки Weka;	+			-
2)	використання бібліотеки модульного тестування;	+			+
3)	використання коректних структур;	-			+
4)	перехоплення виводу;	-			-
5)	формування власного коректного набору даних або вказування на існуючий;	+			+
6)	прогнозування очікуваного результату;	-			-
7)	перевірка проміжних значень;	-			+
8)	завершене рішення.	-			-
6.	createPalindrome(){ ?} <- Google Test example:	<b>2 / 4</b>	-	-	<b>0 / 4</b>
1)	розуміння суті завдання;	+			-
2)	результат має властивості паліндрому;	+			-
3)	довжина результату збігається із необхідною;	-			-
4)	завершене рішення.	-			-
7.	HashTable.h <- Google Test:	<b>5 / 5</b>	-	-	<b>5 / 5</b>
1)	використання бібліотеки модульного тестування;	+			+
2)	формування власного набору даних;	+			+
3)	прогнозування очікуваного результату;	+			+
4)	перевірка проміжних значень;	+			+
5)	завершене рішення.	+			+
8.	HashTable.h <-> ThreadPool.h <- Google Test:	<b>5 / 6</b>	-	-	<b>4 / 6</b>

1) використання бібліотеки модульного тестування;	+			+
2) формування власного набору даних;	+			+
3) прогнозування очікуваного результату;	+			+
4) перевірка проміжних значень;	+			+
5) взаємодія двох класів;	+			-
6) завершене рішення.	-			-
9. HopfieldNetwork <- JUnit:	<b>5 / 6</b>	-	-	<b>5 / 6</b>
1) використання бібліотеки модульного тестування;	+			+
2) формування власного набору даних;	+			+
3) прогнозування очікуваного результату;	+			+
4) перехоплення виводу;	+			+
5) перевірка проміжних значень;	+			+
6) завершене рішення.	-			-
<b>Всього</b>	<b>29 / 50</b>	<b>0 / 50</b>	<b>3 / 11</b>	<b>26 / 50</b>



*Рис. 4.1. Умовна схема роботи фреймворку (Додаток Д).*

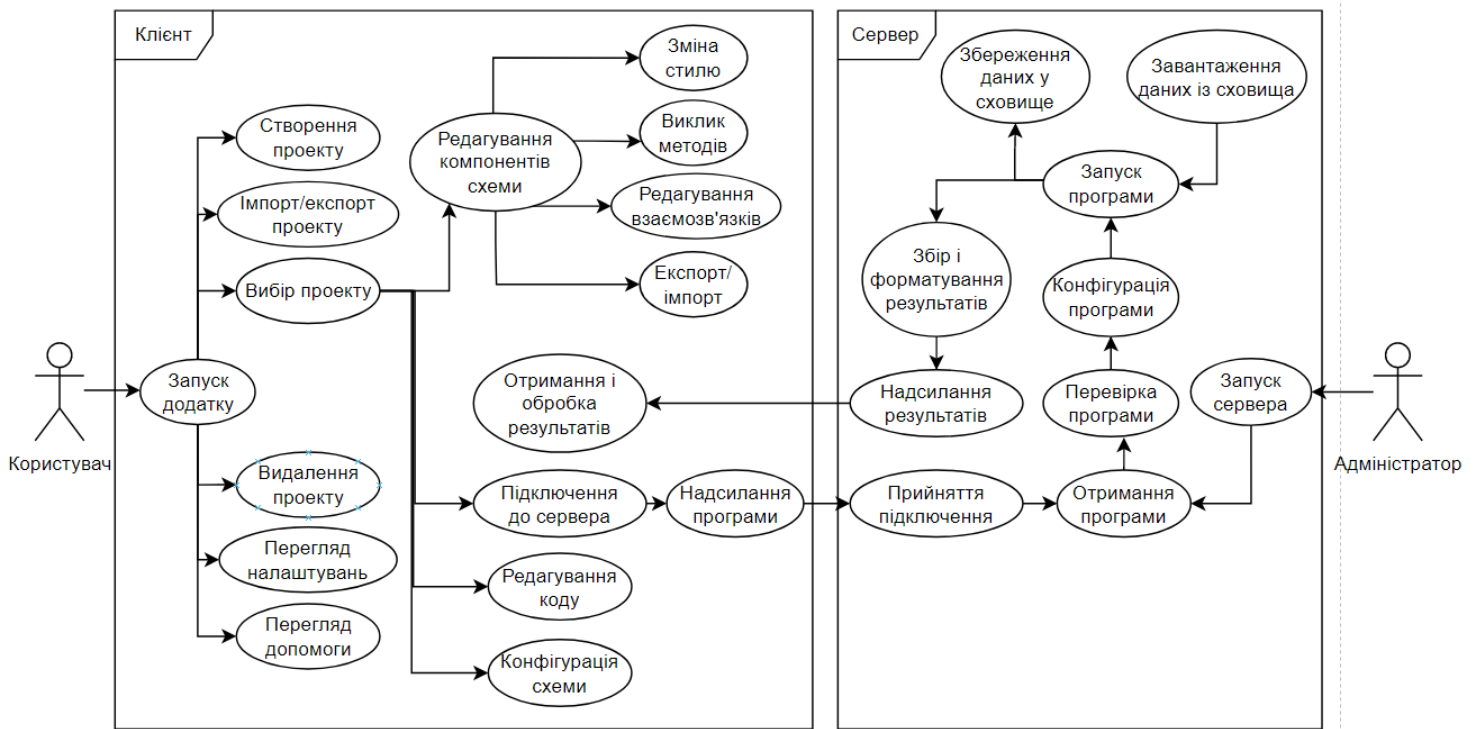


Рис. 4.2. Діаграма прецедентів UML фреймворку (Додаток Д).



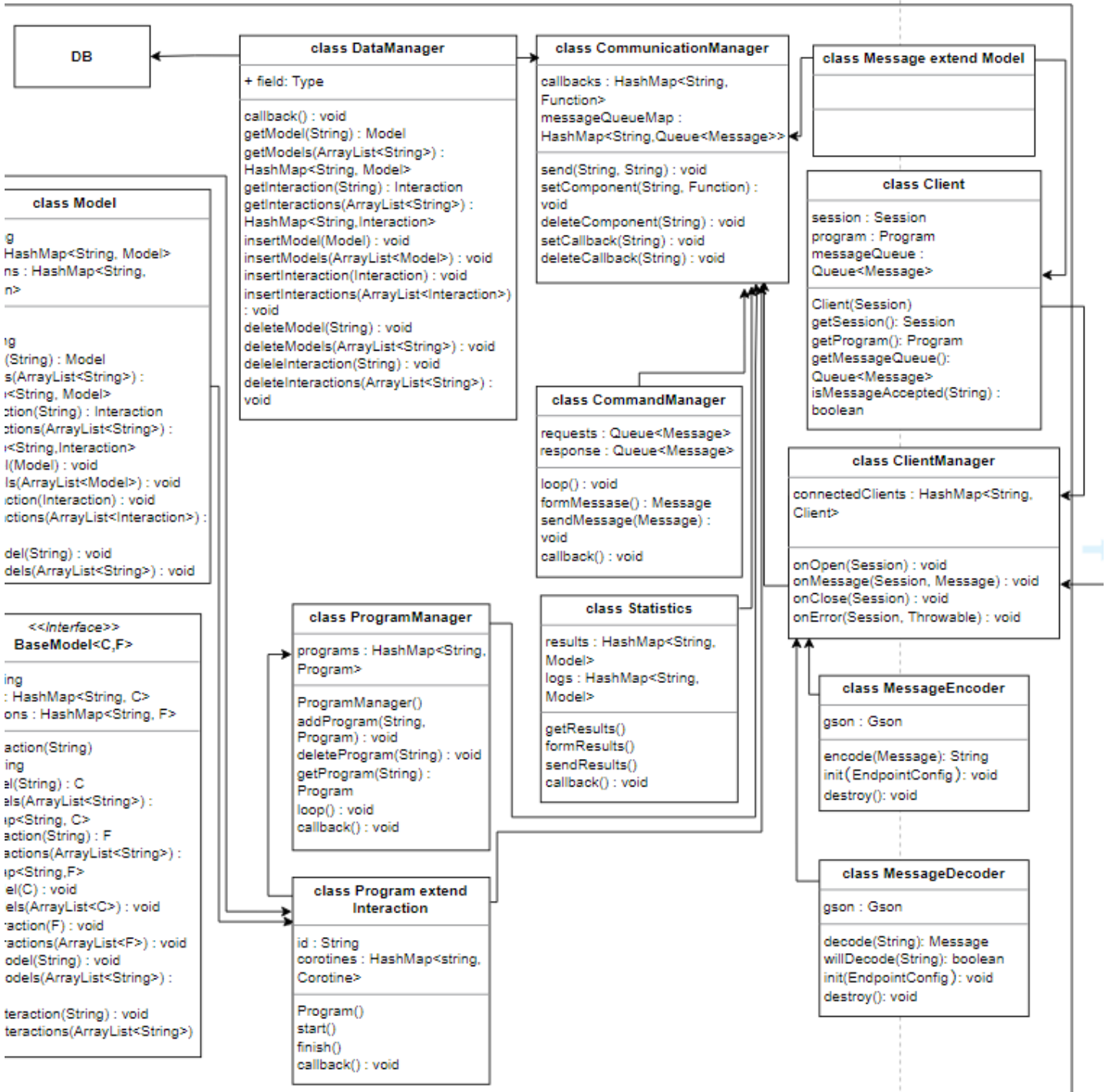


Рис. 4.3. Діаграма класів UML фреймворку. Сервер (Додаток E).

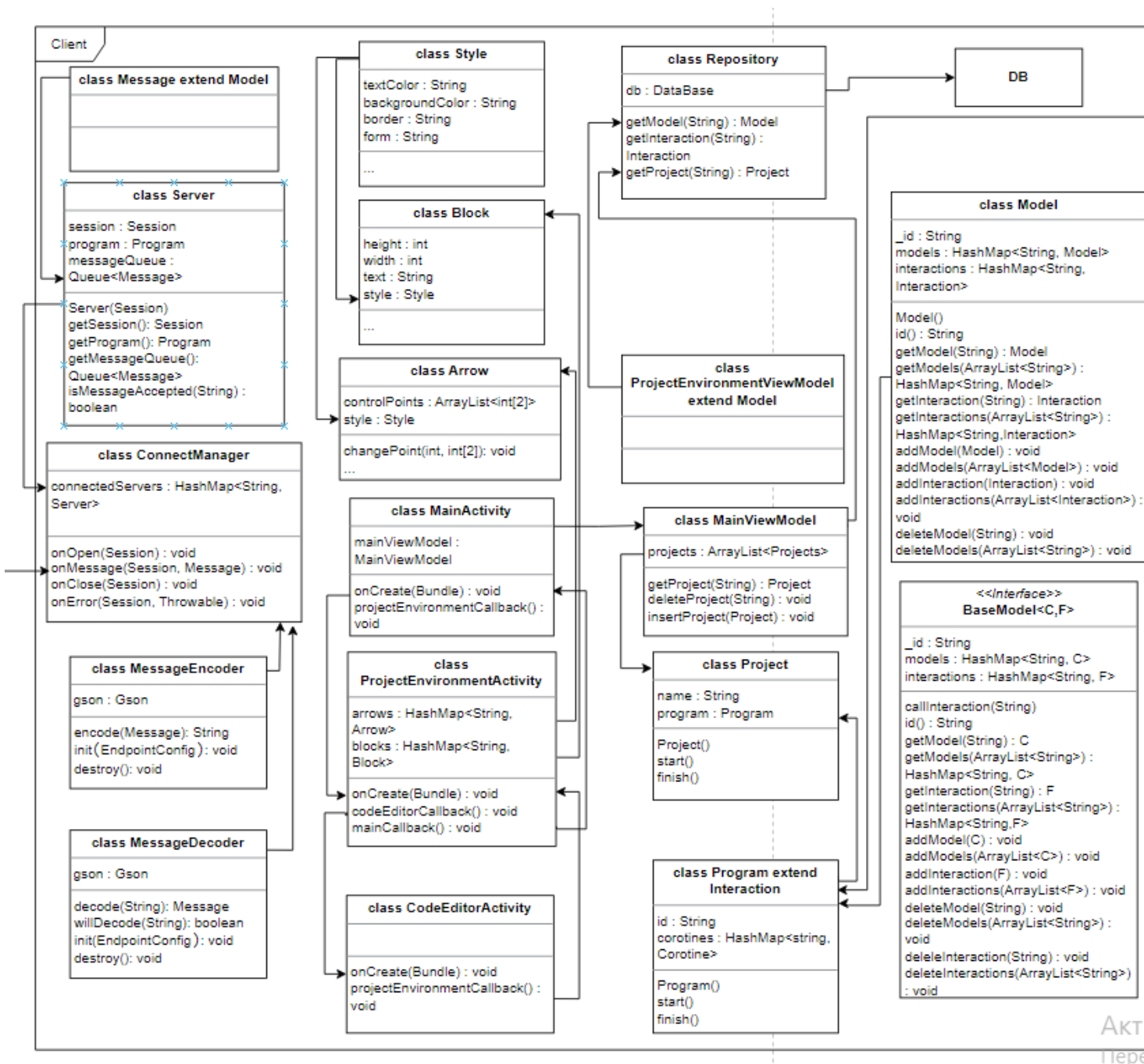
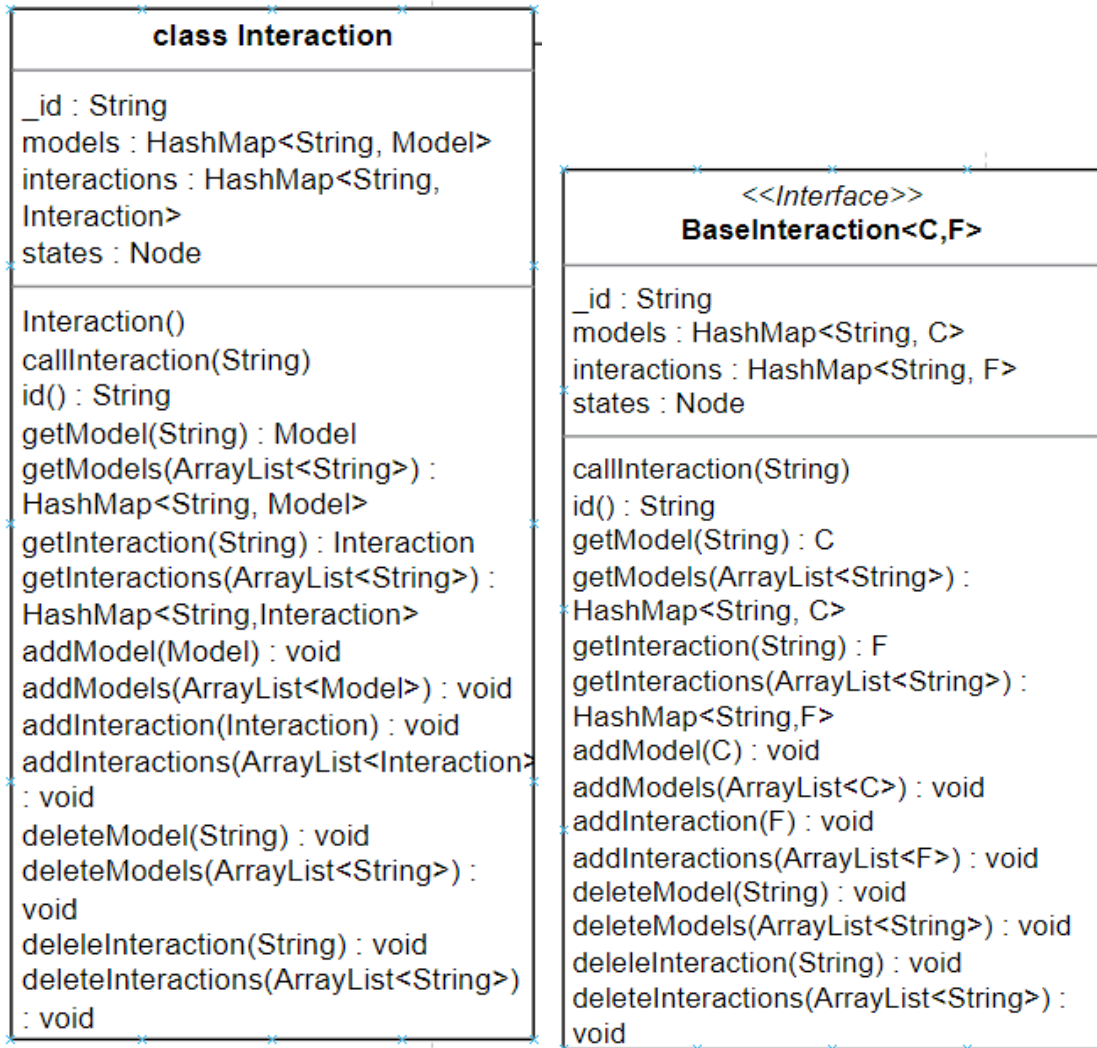
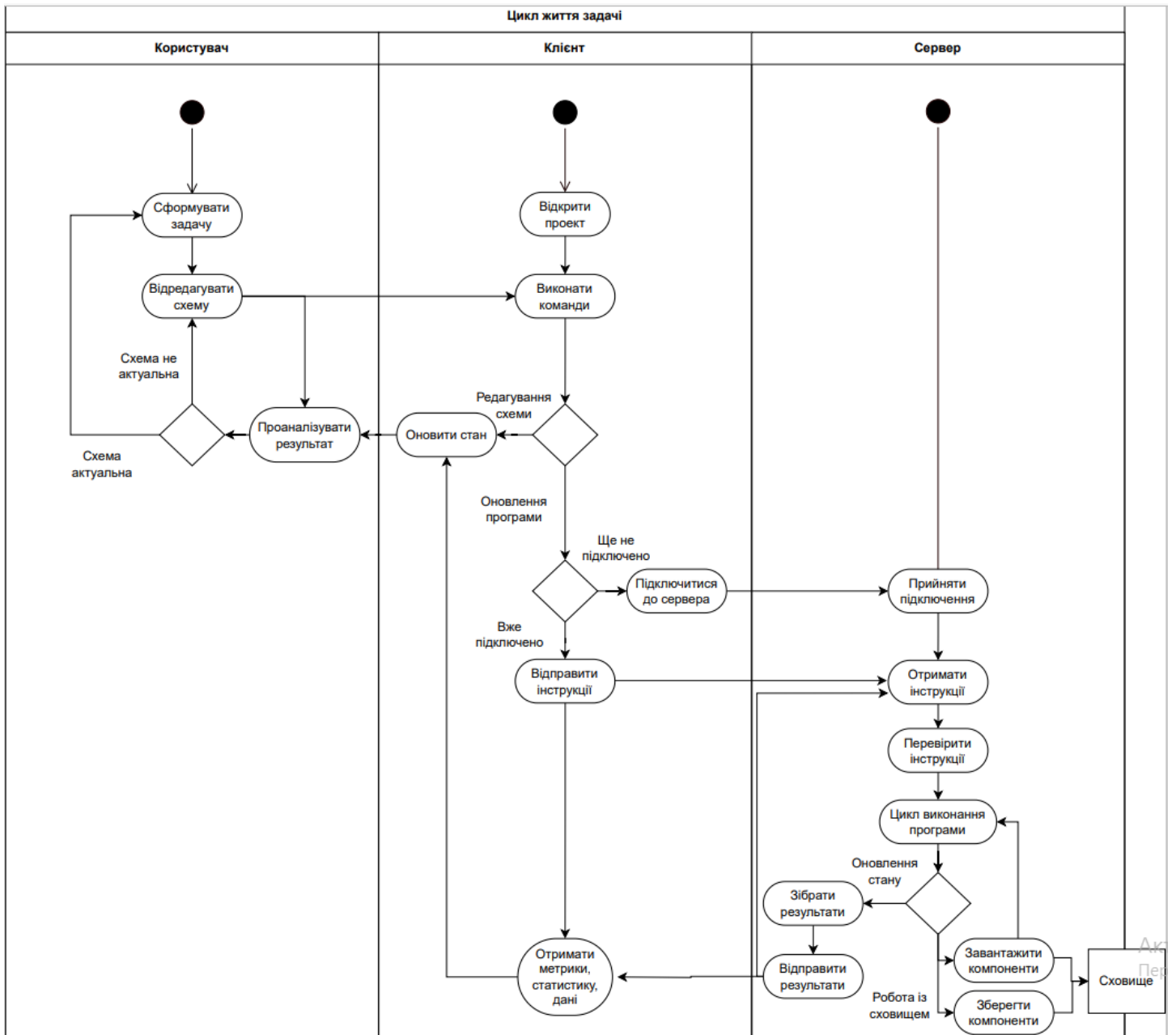


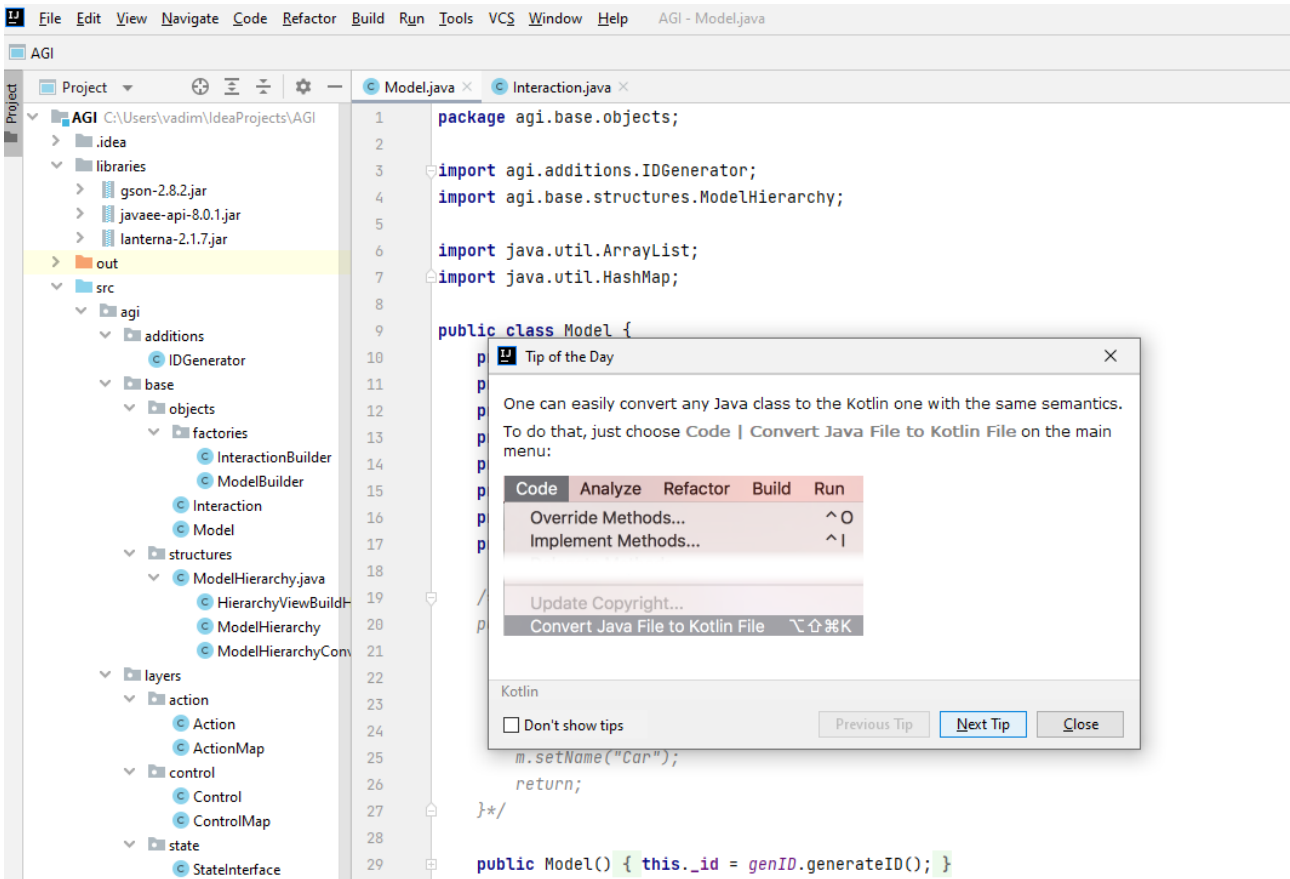
Рис. 4.4. Діаграма класів UML фреймворку. Клієнт (Додаток Е).



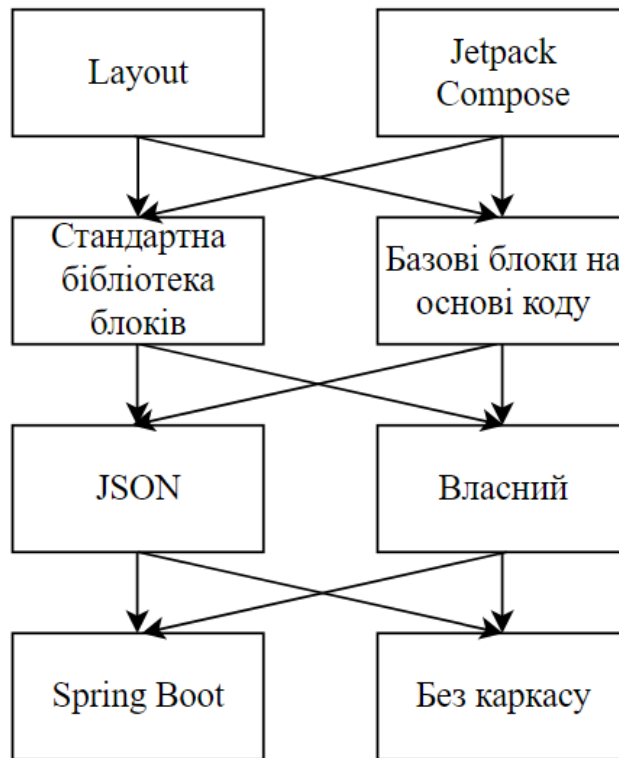
*Рис. 4.5. Діаграма класів UML фреймворку. Interaction (Додаток E).*



*Рис. 4.6. Діаграма станів UML циклу життя задачі (Додаток Є).*



*Рис. 4.7. IntelliJ IDEA IDE.*



*Рис. 6.1. Морфологічна карта реалізацій основних функцій.*

Таблиця 6.1. Позитивно-негативна матриця.

Функції	Варіанти реалізації	Переваги	Недоліки
$F_1$	А	Має багато готових рішень. Використовується більшістю компаній.	Використовує незручний формат представлення. Обмежений у редагуванні під час виконання.
	Б	Зручний для розробника. Може змінюватися під час виконання. Майбутній стандарт.	Молода бібліотека, через що мало готових рішень, недостатньо поширена, можливі баги.
$F_2$	А	Не потрібно працювати із кодом. Менше неполадок. Легше налагоджувати.	Обмежене рішення. Важко інтегрувати сторонні програми. Не можна додати власні базові блоки або це ускладнено.
	Б	Можна швидко інтегрувати стороні програми і розширити список базових блоків.	Потрібно працювати із кодом. Можливі неполадки.
$F_3$	А	Універсальний масштабований стандарт індустрії. Має готові бібліотеки для роботи, трансформації, заповнення.	Однаково прийдеться узгоджувати формат вмісту.
	Б	Можна реалізувати ефективний формат для графічних даних, алгоритмів із підтримкою більшої кількості примітивних типів даних.	Додаткові затрати на розробку. У результаті однаково буде мати слабкі сторони.
$F_4$	А	Готові рішення перевірені досвідом. Інверсія керування. Легко налаштовувати.	Потрібно вчити нову технологію.
	Б	Свобода дій при побудові власної архітектури.	Потрібно створювати каркас самостійно через що більше витрат і рішення може бути не таким ефективним.

Таблиця 6.2 - Основні параметри програмного продукту.

Назва Параметра	Умовні позначен ня	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія трансляції схем	X1	К блоків / с	5	12	20
Швидкодія обчислення числових даних	X2	Mflops	30	50	70
Об'єм пам'яті	X3	Гб	40	30	20
Частота оновлення стану	X4	оновлень / с	30	45	60
Потенційний об'єм програмного коду	X5	кількість рядків коду	20000	17000	13000

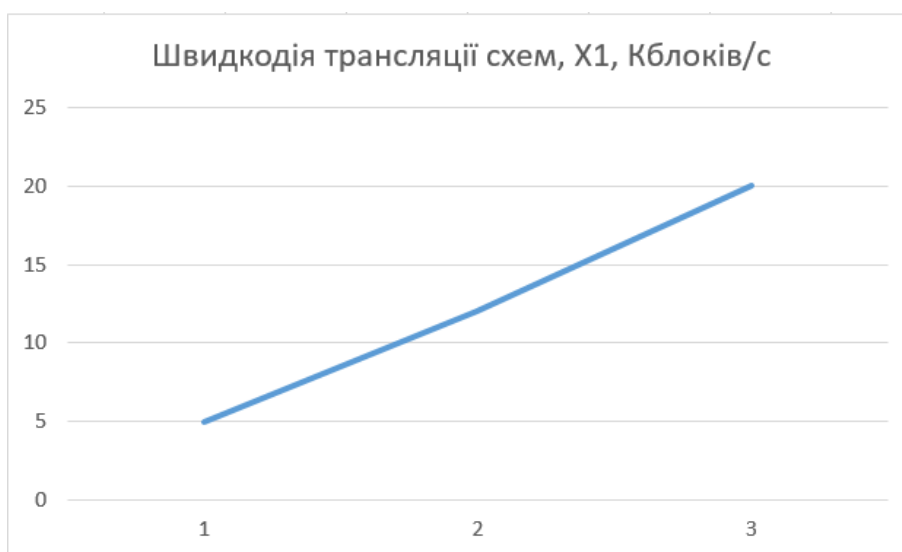
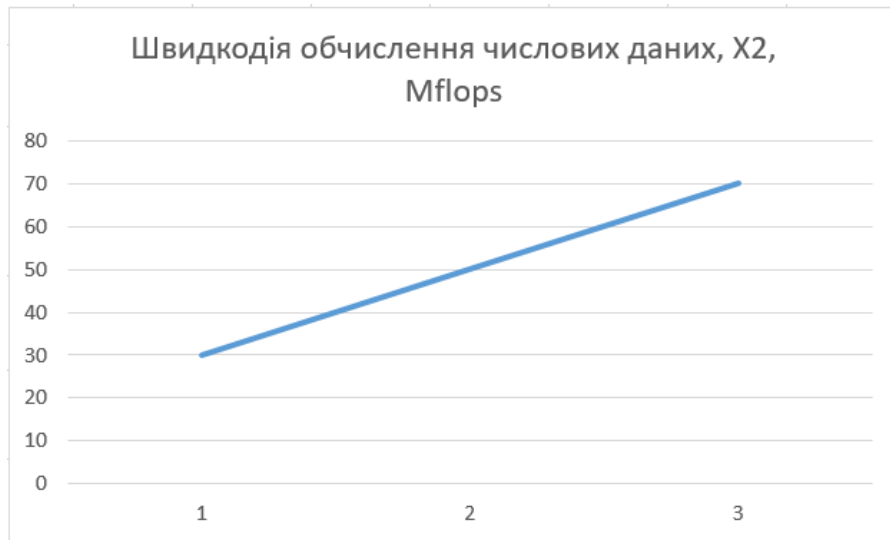
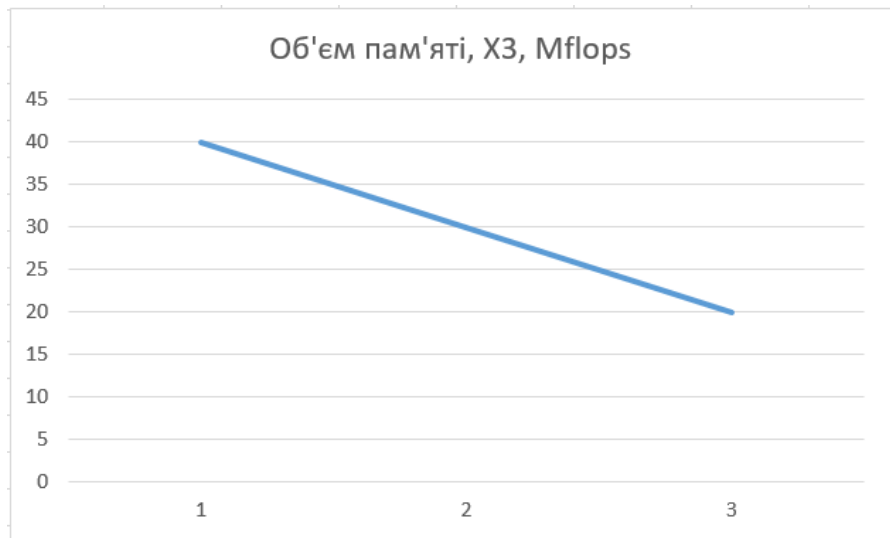


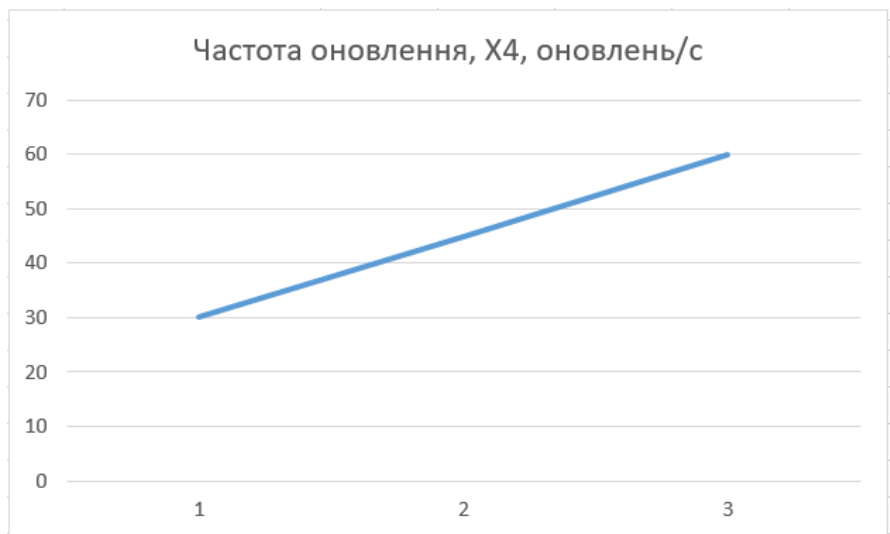
Рис. 6.2. X1 – швидкодія трансляції схем.



*Рис. 6.3. X2 – швидкодія обчислення числових даних.*



*Рис. 6.4. X3 – об'єм даних.*



*Рис. 6.5. X4 – частота оновлення.*



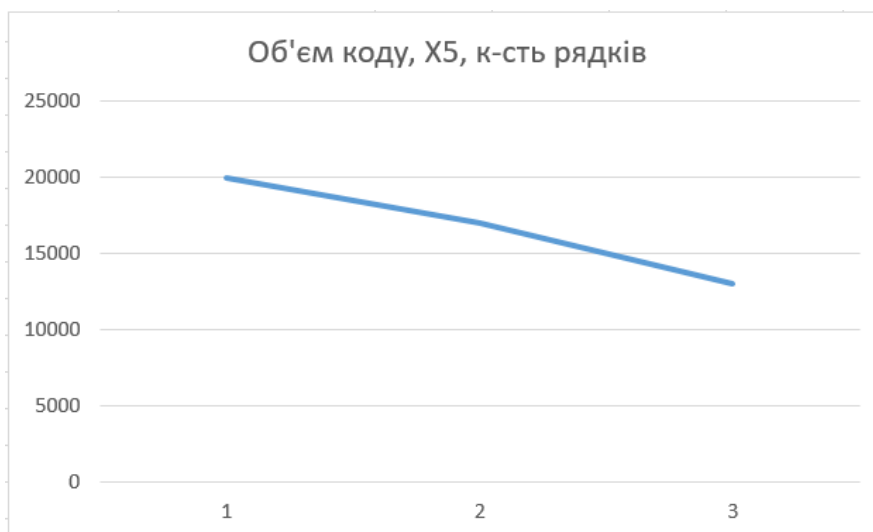


Рис. 6.6. X5 – об'єм коду.

Таблиця 6.3 - Результати ранжування параметрів.

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів $R_i$	Відхилення $\Delta_i$	$\Delta_i^2$
			1	2	3	4	5	6	7			
X1	Швидкодія трансляції схем	К блоків / с	4	5	3	4	5	4	3	28	7	49
X2	Швидкодія обчислення числових даних	Mflops	5	4	5	3	4	5	5	31	10	100
X3	Об'єм пам'яті	Гб	2	1	2	1	2	3	2	13	-8	64
X4	Частота оновлення стану	оновлень / с	3	3	4	5	3	2	4	24	3	9
X5	Потенційний об'єм програмного коду	кількість рядків коду	1	2	1	2	1	1	1	9	-12	144
	Разом		15	15	15	15	15	15	15	105	0	366

Таблиця 6.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	<	>	>	<	<	<	0,5
X1 і X3	>	>	>	>	>	>	>	>	1,5
X1 і X4	>	>	<	<	>	>	<	>	1,5
X1 і X5	>	>	>	>	>	>	>	>	1,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	>	>	>	<	>	>	>	>	1,5
X2 і X5	>	>	>	>	>	>	>	>	1,5
X3 і X4	<	<	<	<	<	>	<	<	0,5
X3 і X5	>	<	>	<	>	>	>	>	1,5
X4 і X5	>	>	>	>	>	>	>	>	1,5

Таблиця 6.5 - Розрахунок вагомості параметрів.

Параметри $x_i$	Параметри $x_j$					Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	X5	$b_i$	$K_{Bi}$	$b_i^1$	$K_{Bi}^1$	$b_i^2$	$K_{Bi}^2$
X1	1	0.5	1.5	1.5	1.5	6	0.24	27.5	0.24	124.75	0.24
X2	1.5	1	1.5	1.5	1.5	7	0.28	34	0.3	155.5	0.3
X3	0.5	0.5	1	0.5	1.5	4	0.16	17.5	0.15	80.25	0.15
X4	0.5	0.5	1.5	1	1.5	5	0.2	22	0.19	100	0.19
X5	0.5	0.5	0.5	0.5	1	3	0.12	14	0.12	64.5	0.12
Всього:						25	1	115	1	525	1

Таблиця 6.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП.

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	Б	X4	45	24	0.19	4.56
F2	А	X1	10	28	0.24	6.72
	Б	X2	100	31	0.3	9.3
F3	А	X3	50	13	0.15	1.95
F4	А	X5	15000	9	0.12	1.08

### Список використаної літератури (файл Малюнки, таблиці.docx)

1. AgentGPT. AgentGPT, reworkd : веб-сайт. URL: <https://agentgpt.reworkd.ai/uk> (дата звернення: 12.05.2023).
2. AI Code Generator - Amazon CodeWhisperer - AWS. Amazon : веб-сайт. URL: <https://aws.amazon.com/ru/codewhisperer/> (дата звернення: 12.05.2023).
3. Durable AI Website Builder and Small Business Software. Durable : веб-сайт. URL: <https://durable.co/> (дата звернення: 12.05.2023).
4. GitHub Copilot · Your AI pair programmer. GitHub : веб-сайт. URL: <https://github.com/features/copilot> (дата звернення: 12.05.2023).
5. Introducing ChatGPT – OpenAI. OpenAI : веб-сайт. URL: <https://openai.com/blog/chatgpt> (дата звернення: 12.05.2023).
6. MetaGPT. Metaidt : веб-сайт. URL: <https://www.metaidt.com/upgrade-your-account/> (дата звернення: 12.05.2023).
7. TDDx2, BDD, DDD, FDD, MDD и PDD, или все, что вы хотите узнать о Driven Development. Хабр : веб-сайт. URL: <https://habr.com/ru/articles/459620/> (дата звернення: 03.05.2023).