Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра Інформаційних управляючих систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження мовних моделей в задачах розробки програмного забезпечення інформаційних систем з використанням методології Kanban
(тема)

Виконав:
студент 2 курсу, групи ІУСТзм-22-1
Столярова Олена Костянтинівна
(прізвище, ім'я, по батькові)

Спеціальність 122
Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Інформаційні управляючі системи та технології
(повна назва освітньої програми)

Керівник Оксана ЧАЛА
(Власне ім'я ПРІЗВИЩЕ)

Допускається до захисту

Зав. кафедри _____ Костянтин ПЕТРОВ
(підпис) (Власне ім'я ПРІЗВИЩЕ)

2024 р.

Харківський національний університет радіоелектроніки

Навчально-науковий центр заочної форми навчання

Кафедра Інформаційних управляючих систем

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційні управляючі системи та технології
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)

« 04 » грудня 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентці Столяровій Олені Костянтинівні
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження мовних моделей в задачах розробки програмного забезпечення інформаційних систем з використанням методології Kanban

затверджена наказом університету від 01 грудня 2023 р. № 259Стз

2. Термін подання студентом роботи до екзаменаційної комісії 19 січня 2024 р.

3. Вихідні дані до роботи: науково-технічна література, публікації, інформація з інтернет-ресурсів, проектна документація.

4. Перелік питань, що потрібно опрацювати в роботі: розробка технології використання великих мовних моделей при розробці програмного забезпечення інформаційних систем за методологією Kanban, Експериментальна перевірка отриманих теоретичних результатів.

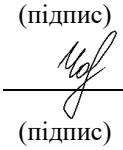
КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	04.12-05.12.2023	Виконано
2	Постановка задач дослідження	05.12-07.12.2023	Виконано
3	Аналіз характеристик великих мовних моделей	06.12-12.12.2023	Виконано
4	Розробка технології використання великих мовних моделей при розробці програмного забезпечення інформаційних систем за методологією Kanban	12.12-22.12.2023	Виконано
5	Експериментальна перевірка отриманих теоретичних результатів	22.12-31.12.2023	Виконано
6	Оформлення пояснювальної записки до кваліфікаційної роботи	01.01-13.01.2024	Виконано
7	Захист роботи	22.01.2024	Виконано

Дата видачі завдання 04 грудня 2023 р.

Студентка  Столярова О. К.

(підпис)

Керівник роботи  проф. каф. ІУС Чала О. В

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до магістерської кваліфікаційної роботи містить: 81 с., 4 розділи, 20 рис., 4 табл., 1 додаток, 32 джерела.

ВЕЛИКІ МОВНІ МОДЕЛІ, ЛАНЦЮЖОК ПРОМПТІВ, GPT-4, KANBAN, MICROSOFT COPILOT, PROMPT, USER STORY.

Кваліфікаційна робота зосереджена на дослідженні використання мовних моделей (LLM) для покращення ефективності розробки програмного забезпечення в методології Kanban. Розглянуто використання LLM, зокрема GPT-4, в процесів створення User Story в системі Kanban.

Практичні результати роботи включають прискорення описання User Story у системі Kanban та покращення їх детальності. Задачі дослідження охоплюють аналіз особливостей розробки програмного забезпечення за методологією Kanban, аналіз великих мовних моделей, вивчення підходів до побудови запитів до LLM, а також експериментальну перевірку запропонованого методу.

В роботі використано використовує ланцюжки промптів для взаємодії з мовними моделями, такими як GPT-4. Також робота включає в себе використання GitHub Actions для автоматизації процесів та аналіз результатів за допомогою пайплайнів. Робота є обґрунтованою і структурованою, використання LLM для оптимізації розробки в системі Kanban є цікавою та актуальною темою. Подальший вивід можна зробити, розглядаючи конкретні дані та результати експериментальної перевірки.

В ході дослідження отримані такі результати: досліджено особливості процесу розробки програмного забезпечення, проаналізовані існуючі LLM та методи побудови промптів для створення прецеденту User Story; проведено експериментальна перевірка по удосконаленому методу написання User Story.

ABSTRACT

The explanatory note to the master's thesis contains: 81 pages, 4 chapters, 20 figures, 4 tables, 1 addendum, 32 sources.

GPT-4, KANBAN, LARGE LANGUAGE MODELS, MICROSOFT COPILOT, PROMPT, PROMPT CHAIN, USER STORY.

The thesis focuses on researching the use of linguistic models (LLM) to improve the efficiency of software development within the framework of the Kanban methodology. The paper considers the use of LLM, as GPT-4, for the process of creating a User Story and managing tasks in the Kanban system.

The practical results of the work include speeding up the description of User Stories in the Kanban system and improving their detail. The tasks of the research include the analysis of the features of software development using the Kanban methodology, the analysis of large language models, the study of approaches to the construction of requests for LLM, as well as the experimental verification of the proposed method.

The work uses prompt chains to interact with language models such as GPT-4. The work also includes using GitHub Actions to automate processes and analyze results using pipelines. The work is grounded and structured, the use of LLM to optimize development in the Kanban system is an interesting and relevant topic. Further conclusions can be drawn by considering specific data and results of experimental verification.

During the study, the following results were obtained: the features of the software development process were investigated, the existing LLM and methods of building prompts for creating a User Story precedent were analyzed; experimental verification of the improved method of writing User Story was carried out.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – інтерфейс програмування застосунків

ІС – інформаційна система

ПЗ – програмне забезпечення

ХР – екстремальне програмування.

LLM – велика мовна модель (large language model)

NLP – обробка природної мови (natural language processing)

Prompt – команда або запит до LLM

GPT – генеративний попередньо навчений трансформатор (Generative Pre-trained Transformer)

UAT – приймальне тестування (user acceptance testing)

ВСТУП

Сучасний світ інформаційних технологій та розробки програмного забезпечення безперервно розвивається, вимагаючи від фахівців у цій галузі постійного вдосконалення та адаптації до нових задач. Однією з важливих тенденцій є використання методологій, які дозволяють ефективно керувати процесом розробки програмного забезпечення та забезпечити максимальну якість продукту є методологія Kanban, яка стала досить популярною серед розробників та команд, що працюють над програмним забезпеченням [4, 14].

Спостерігається й інша ключова тенденція, яка визначає динаміку цієї галузі - це впровадження інтелектуальних рішень і технологій. Серед них особливе місце займають сучасні мовні моделі, що базуються на штучних нейронних мережах та машинному навчанні. Вони змінюють способи взаємодії з інформаційними системами та мають великий потенціал для автоматизації завдань та підвищення рівня персоналізації [1, 23].

Зростаюча потреба в обробці великих обсягів даних та виконання складних завдань аналізу призвела до активного розвитку мовних моделей, таких як GPT-3 та його послідовники. Ці моделі можуть розуміти та генерувати текст, робити прогнози та рекомендації, «розуміти» зміст інформації в різноманітних контекстах, що дає безмежні можливості для вдосконалення програмного забезпечення та інформаційних систем [2, 21].

Зокрема, сучасні мовні моделі можуть бути використані для автоматизації рутинних завдань, таких як генерація документації, переклад тексту, аналіз великих текстових наборів даних та багато інших. Вони також можуть стати потужним інструментом для покращення комунікації між розробниками та замовниками, адаптуючись до конкретних потреб користувача [3, 5].

Актуальність дослідження мовних моделей у контексті використання методології Kanban полягає в тому, що поєднання цих двох аспектів може

надати значні переваги в процесі розробки програмного забезпечення. Можливість швидкої обробки та аналізу великих обсягів текстової інформації може допомогти швидко приймати обґрунтовані рішення, використовуючи Kanban, сприяючи підвищенню продуктивності та якості розробки. Ця робота має на меті вивчити можливості та перспективи поєднання цих двох ключових складових сучасної розробки програмного забезпечення.

У даній роботі буде проведено дослідження використання мовних моделей в задачах розробки програмного забезпечення, зосереджуючись на використанні методології управління проектами Kanban. Ця тема є актуальною та важливою у контексті сучасного ринку програмного забезпечення та інформаційних систем. Дослідження мовних моделей та їх використання в поєднанні з методологією Kanban може сприяти покращенню ефективності процесу розробки, а також забезпечити високий рівень задоволеності замовника та якість продукту.

1 ДОСЛІДЖЕННЯ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Дослідження особливостей розробки програмного забезпечення інформаційних систем з використанням методології Kanban

Канбан – це методологія розробки програмного забезпечення, яка впроваджує принципи візуалізації робочого процесу та обмеження одночасної кількості завдань для досягнення оптимальної продуктивності. Канбан дозволяє оптимізувати робочі потоки, визначати пріоритети, та забезпечує постійне вдосконалення продуктивності команди розробників [15, 32].

Життєвий цикл ПЗ – це етапи, через які воно проходить із початку створення до кінця розробки та впровадження. Методологія розробки включає набір методів з управління розробкою: правила, техніки та принципи, які роблять її більш ефективною. Існує велика кількість методів і способів розробки. Одними з найпопулярніших є модель водоспаду, спіральна модель, гнучка, Scrum, Kanban та екстремальне програмування [7, 16].

Методології розробки ПЗ є поєднанням практик і цінностей. Практики допомагають розробникам визначити, що та коли їм потрібно зробити. Цінності служать простим етичним кодексом, яким керуються інженери.

Модель водоспаду — це модель життєвого циклу розробки ПЗ, яка включає набір кроків у певному порядку. Процес розробки розділений на різні фази, і розробник може переходити до наступної фази, тільки коли поточна фаза завершена. Фази не перетинаються, завдяки чому розробка відбувається послідовно. Через це модель водоспаду називається лінійно-послідовною моделлю. Модель водоспаду подібна до односпрямованого процесу, який розробник має довести до кінця, і де неможливо повернутися назад або змінити траєкторію. Єдиний варіант повернутися до попереднього етапу — це заново розпочати цикл розробки [24].

Існують невеликі варіації фаз моделі водоспаду. Однак, зазвичай виділяють шість.

Перший етап – це аналіз вимог. На цьому етапі з клієнтом обговорюють вимоги проекту. Проводять аналіз вимоги, щоб визначити, чи є якісь розпливчасті чи непослідовні вимоги. Потім результати обговорюють з клієнтом, який узгоджує рішення з командою розробників. Остаточно сформульовані вимоги до системи документують у специфікації до ПЗ. Ця документація є угодою між командою і клієнтом.

Другий етап – проектування системи. Команда розробників використовує специфікацію і аналізує її при створенні дизайну, який відповідає всім вимогам клієнта. Цей етап передбачає розробку ПЗ та визначення інструментів, які використовуються для розробки системи.

Третій етап – впровадження. Створюється ПЗ, яке відповідає дизайну, сформульованому на попередньому етапі.

Після завершення етапу впровадження починається четвертий етап – тестування. Для ПЗ проводять різні тести. Найчастіше це модульне тестування, інтеграційне тестування та приймальні випробування. Модульні тести проводяться для кожного блоку коду з унікальною функціональністю в системі. Інтеграційні тести поєднують окремі функції та перевіряють система в цілому. Приймальні випробування виконує клієнт. Під час цього процесу клієнт проводить остаточні тести та вирішує, прийняти чи відхилити ПЗ. Якщо ПЗ відхилене, команді доведеться перезапустити модель водоспаду з першого етапу та змінити вимоги до ПЗ.

П’ятий етап – розгортання. Програмний продукт працює в реальному світі відповідно до вимог клієнта. Наприклад, ПЗ може бути використано лише компанією клієнта або опубліковано в Інтернеті.

Шостий і останній етап – технічне обслуговування. Після розгортання ПЗ можуть знадобитися зміни. Після змін ПЗ може проводитися новий випуск ПЗ.

Модель водоспаду є відносно простою. Для впровадження цієї методики не потрібна сертифікація чи високий рівень майстерності. Це пов'язано з тим, що для правильного застосування водоспаду потрібно лише дотримуватися шести етапів [6].

Спіральна модель розробки ПЗ є способом візуалізації шляху розробки, який повинен зациклюватися до кінця проекту, коли виконані всі вимоги клієнта. У спіральної моделі є чотири основні фази.

Перший етап – планування. На цьому етапі відбувається спілкування з клієнтом, досягнення розуміння командою розробників вимог. Потім команда розробників створює загальний план розробки з урахуванням всіх вимог, визначає, які технології повинні використовуватися, визначає задачі, а також розраховує бюджет та час.

Другий етап – аналіз та зменшенні ризиків. Розробники обговорюють потенційні ризики, пов’язані з проектом, і визначають можливі шляхи подолання цих ризиків. До кінця цього етапу створюється прототип. Прототип служить базовою моделлю ПЗ та демонструє клієнту приклад готового продукти. Цим команда розробників ще більше зменшує ризики і може отримати або схвалення, або критику поточного дизайну.

Третя фаза – розробка. На цьому етапі створюється фактичне програмне забезпечення. Розробники збирають відгуки про прототип і створюють ПЗ, яке відповідає вимогам замовника.

Останній четвертий етап – перегляд та оцінка. Команда, а потім клієнт перевіряють створене ПЗ. Відгук клієнта визначає чи будуть повторені фази планування та аналізу ризиків. Однак, якщо клієнт приймає ПЗ, цикл розробки завершується.

Спіральна модель унікальна, оскільки підкреслює важливість прототипування. Клієнт може надати відгук про загальну реалізацію проекту, перш ніж команда повністю завершить продукт. Прототипи роблять можливим вчасний зворотній зв'язок з клієнтом, що зменшує ймовірність суттєвих змін на останніх стадіях розробки. Спіральна модель також приділяє

увагу прогнозуванню ризиків, що відрізняє її від інших популярних методологій розробки, оскільки більшість процесів лише борються з ризиками, які вже виникли. Спіральна модель має лише чотири фази, тому є зрозумілою та легкою в реалізації [8].

Agile було створено, щоб команди розробників програмного забезпечення могли швидко та якісно виконувати роботу в межах бюджету та відведеного часу. Agile складається з різних методів розробки. Ця концепція є основою для багатьох популярних, більш конкретних методологій, таких як Scrum, екстремальне програмування та Kanban [9].

Екстремальне програмування або XP — це найбільш специфічний фреймворк Agile для розробки ПЗ. У ньому конкретно вказано, яких практик потрібно дотримуватися. Першою цінністю є спілкування. Згідно зі структурою XP, найкращий тип спілкування — це віч-на-віч із дошкою поруч для візуалізації проблем. Ще одна цінність – простота. Дизайн системи має бути максимально простим, також потрібно робити лише абсолютно необхідні речі під час розробки. Згідно з XP, не треба думати про потенційні вимоги до ПЗ, які можуть виникнути в майбутньому, і лише керуватися поточними вимогами Третя цінність – зворотний зв'язок, який допомагає командам усвідомити, що вони роблять правильно, а що – ні. Четвертий принцип – це мужність братися за складні задачі та висловлювати свою думку. Остання цінність – повага: потрібно поважати кожного у команді.

Існує багато правил, яких необхідно дотримуватися, щоб правильно застосовувати методологію XP. По-перше, члени команди повинні знаходитися поруч, бо так легше спілкуватися віч-на-віч. Потрібно застосувати парне програмування — це розробку ПЗ двома людьми на одній машині. Ще одна практика – використання історій користувачів, які описують, що кінцевий продукт з точки зору користувача. Цикл розробки має тривати тиждень. Команди повинні визначати низько пріоритетні задачі, від яких можна відмовитися на користь виконання більш важливих. Постійна

інтеграція – ще одна ключова практика: код часто об’єднується і завжди тестується відразу.

Scrum — це ітеративна та поетапна методологія розробки. На відміну від Agile, Scrum має більше визначень і конкретних ідей розробки. Наприклад, він описує ролі, які повинні виконувати працівники.

Команда в Scrum – це невелика група людей, що складається з Scrum-майстра, одного власника продукту і кількох розробників. Scrum-майстри, по суті, є лідерами команди, відповідають за регулярний моніторинг ефективності. Ще одна роль у Scrum – це власник продукту, який керує списком спринтів або коротких спайків з конкретними цілями. Ключові елементи Scrum – це перевірка, прозорість та адаптація. Ключові цінності – відданість, цілеспрямованість, відкритість, повага та сміливість, які зумовлюють особливості розробки.

Scrum — це легка методологія, але має також багато «правил». Наприклад, дає рекомендації щодо покращення спілкування: щоденні зустрічі під назвою Daily Scrums.

Scrum дає рекомендації щодо підвищення ефективності розробки: розподіл роботи по спринтах. Однак, занадто багато уваги плануванню інколи може уповільнити процес розробки ПЗ, що протирічить основній цілі методології [11].

Kanban — методологія розробки на основі Agile. Основна мета Kanban — скоротити час, необхідний для завершення проекту. Є три основні принципи Kanban:

– треба зрозуміти, яка робота зараз виконується, особливості конкретного завдання;

– програміст повинен прагнути вносити якісні зміни протягом усього процесу розробки;

– працівники повинні мати повноваження бути лідерами в певній сфері, мати актуальні знанні технологій, які використовують в роботі.

В таблиці наведені основні характеристики та відмінності методологій розробки ПЗ [12].

Таблиця 1 – Відмінності SCRUM і Kanban

SCRUM	Kanban
Необхідно проводити щоденні зустрічі	Щоденні зустрічі не обов’язкові
В кінці спринта проводиться ретроспектива	Ретроспектива не обов’язкова
Обов’язкові ролі: SCRUM-Майстер та Product Owner	Єдина роль що відрізняється від інших – team lead
Є оцінювання кожного завдання	Оцінка завдань не обов’язкова
Пріоритети завдань не можуть змінюватись під час спринта	Пріоритети можуть змінюватись будь-коли
В кожен спринт повинна бути виконана визначена кількість завдань	Завдання можуть бути взяті до виконання з бек-логу будь-коли

Однією з основних практик є візуалізація, яка команді зрозуміти, що вона робить і що потрібно робити. Kanban-дошка використовується для візуалізації всіх задач, які необхідно виконати для завершення проекту. Kanban-дошка розділена на колонки. Зазвичай є колонка для нових історій користувачів, які потрібно першими виконати в спринті, колонка для задач, які зараз виконуються, колонка для задач, які тестуються, та колонка завершених історій користувачів.

Іншою поширеною практикою методології Kanban є обмеження обсягу незавершеної роботи. Команда має зосереджувалася на одному аспекті проекту за раз. Це підвищує якість результату та згуртовує команду.

Ще одна практика методології Kanban – це управління робочим процесом, тобто визначення того, як часто певні люди мають працювати та над певними задачами; а також скільки часу потрібно для досягнення певного

результату. Команди самі керують робочим процесом, щоб завершити проект вчасно та в рамках бюджету.

В Kanban є дуже чіткі принципи розробки. Те, що потрібно зробити для проекту, має бути зрозумілим усім членам команди. Це також означає, що слід добре розуміти процес розробки в цілому: розробники мають дотримуватися об’єму запланованої роботи та загальних практик розробки, які були визначені командою в минулому.

Також поширеною практикою є впровадження циклів зворотного зв’язку, щоб команда виділила час і побачила, що працює, а що ні, та внесла відповідні зміни.

Крім того, Kanban зосереджується на поступовому вдосконаленні та розвитку ПЗ, тому компаніям легко розпочати впровадження Kanban. Компанія може почати з поточної роботи і поступово вносити невеликі зміни для збільшення гнучкості процесу. В Kanban можна використовувати існуючий метод розробки вашої команди та розвивати його під час роботи над проектом. Не потрібно одразу змінювати існуючі ролі. Керівник команди або менеджер проекту можуть продовжити виконувати свої ролі.

Kanban має єдиний потік для всього проекту. Однак періодично проводяться цикли зворотного зв’язку. На щоденних зустрічах команда обговорює в загальних рисах те, над чим вона працювала, поточні успіхи чи труднощі. Щотижня відбувається більш довга зустріч, де команда розповідає про те, що вони зробили і над чим працюватимуть далі. По суті, це планування спринту, але цей процес планування для Kanban менш формальний. Щомісяця проводять розбір ризиків, де визначають потенційні ризики чи проблеми з ПЗ.

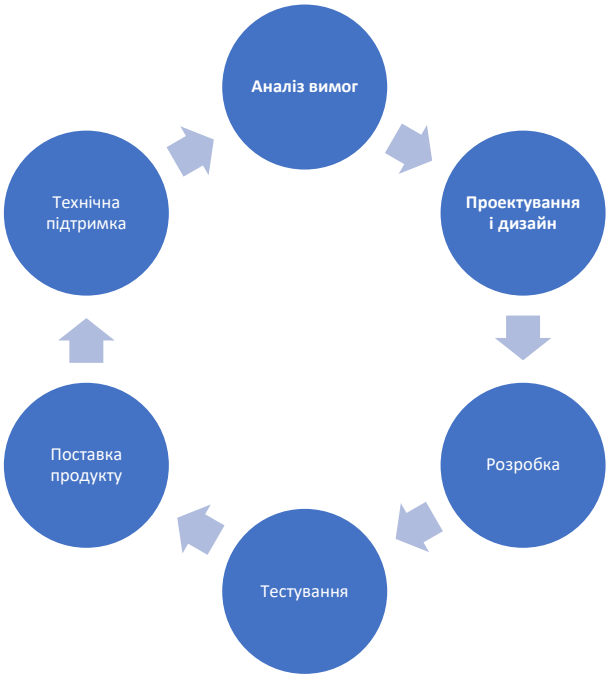


Рисунок 1 – Цикл розробки програмного забезпечення за Kanban

Однією з найбільш унікальних особливостей Kanban є акцент на візуалізації. Завдяки використанню дошок розробники можуть бачити, які завдання коротко перелічені в кожній категорії. Kanban також легко почати впроваджувати. Ця методологія дозволяє кожному зберігати свої поточні ролі та вносити поступові зміни в робочий процес для збільшення гнучкості. Kanban не встановлює конкретних термінів виконання, і команда сама вирішує, коли виконати задачі чи впроваджувати функціональність.

Одним із найпопулярніших прикладів гнучких підходів є Kanban-метод. Однак для ефективної роботи з використанням цієї системи потрібно реалізувати чітку схему циклів розробки програмного забезпечення.

Kanban реалізує ідею потоку як виробничого процесу, де немає простою незавершених завдань. Тобто над однією задачею можуть працювати послідовно декілька спеціалістів. У цьому разі попередні помилки очевидні та виявляються миттєво. Це дозволяє уникнути зайвих витрат, підвищує якість розробки та скорочує терміни її виконання.

Команда складається з таких спеціалістів: власника продукту (Product Owner), програмістів і головного розробника (Developer Lead), спеціалістів із

тестування (QA) і головного QA (QA Lead), бізнес-аналітика (Business Analyst) і продукт-архітектора (Product Architect).

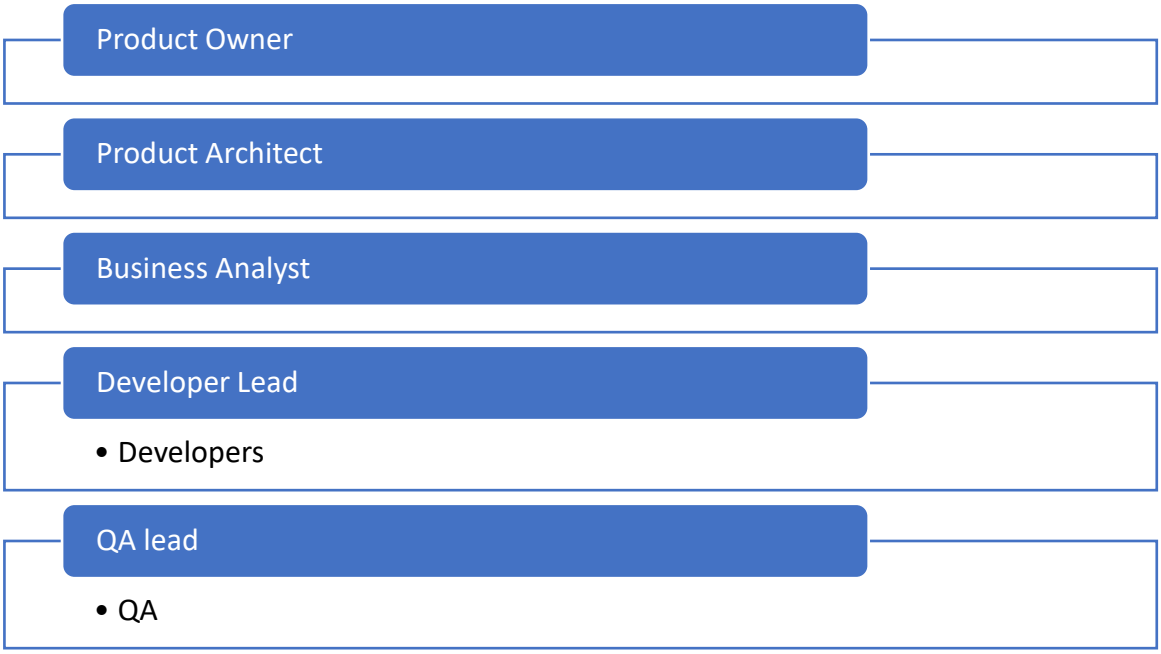


Рисунок 2 – Виконавці за Kanban

Після затвердження проектних рішень, прийнятих на етапі проектування, команда починає розробку. Весь функціонал розподіляться між програмістами, котрі реалізують алгоритми, пишуть вихідний код, виконують компіляцію і налагодження коду. У проекті є план процесу робіт. Спочатку він аналізується і поділяє дошку на об’єкти, які відображають етапи [17].

Виділять статуси для завдань: To do, In Progress, Code Review, Ready QA Testing, QA Under Test, Ready UAT Testing, UAT Under Test, UAT Done, Expected та Done.

Виділяють п’ять основних типів завдань, які використовуються під час роботи над проектом: завдання (task), історію (story), епік (epic), поліпшення (improvement) і дефект (bug). На початку циклу розробки створюється проектна документація із вимогами до системи. Вона повинна чітко та детально відображати весь функціонал системи та після погодження бути оформлена бізнес-аналітиками у вигляді задач.

Спочатку створюється епік – велике завдання, яке повністю описує окремий новий функціонал. Кожен епік складається з історій (описів вимог). Кожна історія реалізує окремий функціонал і може не входити до складу епіків, а бути окремим самостійним об’єктом, що описує новий функціонал або зміни поточного.

Оскільки Kanban не передбачає жорстких, обмежених певних часом етапів, під час яких потрібно реалізувати ту чи іншу задачу, рішення про початок роботи із завданням приймається на командних мітингах (основа зворотного зв’язку).

Основні типи мітингів:

– щоденна зустріч, де обговорюється статус поточних завдань;

– зустріч по плану робіт (раз на два тижні);

– зустріч по покращенню сервісу (раз на два тижні) – обговорюється якість системи та її покращення;

– зустріч із організації процесу роботи (раз на два тижні) – обговорюються проблеми, що виникли під час організації процесу роботи (займається Agile-майстер, котрий координує роботу команди);

– зустріч по обзору стратегії (раз на місяць/квартал) – обговорюються зміни у стратегії.

Члени команди самостійно організовують свою роботу, використовуючи Kanban-дошку. Керівник займається пріоритезацією завдань і вирішенням виникаючих проблем.

Product Architect аналізує побажання потенційних клієнтів щодо функціоналу та інтерфейсу додатку. На основі отриманих даних створюється проектна документація з описом вимог.

На наступному етапі створюються задачі. Для нових створених задач автоматично призначається статус To do. Цей статус означає, що задача створена, але поки не розпочата робота над нею. Спеціаліст, на якого призначена задача, змінює статус на In Progress, коли починає виконання. На етапі розробки програміст, який розробляє або виправляє функціонал,

описаний у завданні, вносить зміни на dev-оточенні та залишає свої коментарі про результати виконання.

Розробники розроблюють окремо невеликі частини функціоналу та проводять юніт-тестування, тобто переконуються, що окремо взята частина додатку працює. Після цього статус завдання потрібно змінити на Code Review.

Після встановлення цього статусу на задачу автоматично призначається головний розробник, який перевіряє розроблений код. Якщо код задовольняє вимоги, зміни додаються до тестового оточення. Задачі призначається статус Ready QA Testing.

Коли починається етап тестування, QA-спеціаліст змінює статус на QA Under Test. Головна задача тестувальника – перевірити, що нова частина додатку працює та коректно взаємодіє з усією системою.

Далі продукт-архітектори та розробники вивчають проблему та приймають рішення про доцільність її вирішення. Коли тестування задачі закінчено, проблеми вирішені та функціонал відповідає поставленим до нього вимогам, тестувальник переміщає задачу у статус Ready UAT Testing.

Коли починається цикл UAT тестування, задачі переводяться спеціалістами у статус UAT Under Test, і призначаються відповідні тестувальники. Тестування проводиться бізнес-користувачами прийнятої системи, тобто потенційними клієнтами. Перевірки проводять на спеціальному оточенні, яке містить у собі всі зміни та доповнення, що відбувалися на етапах розробки та QA-циклу. Після успішного завершення перевірки задачі переводяться до UAT Done.

Також передбачені статуси Expected і Done. Expected призначаються створеному дефекту, якщо він визнається не дефектом, а особливою можливістю (feature). Тобто мається на увазі поведінка системи, що явно не відображена у вимогах, але є правильною з погляду бізнес-логіки.

Статус Done використовується для дефектів, які вирішено не виправляти. Окрім цього, цей статус виставляється для задач типу Task, коли вони виконані тестувальником або розробником.

Kanban-метод передбачає обговорення продуктивності в режимі реального часу і повну прозорість робочих процесів. Етапи роботи візуально представлені на Kanban-дошці, що дозволяє членам команди бачити стан кожного завдання у реальному часі. Кожен член команди отримує до неї доступ в будь-який час і бачить, на якому етапі перебуває те чи інше завдання. Наявні приклади дошок передбачають розподіл на основні категорії: планування, розробку, тестування та реліз. Такий підхід дозволяє чітко розуміти, на якому етапі знаходиться команда.

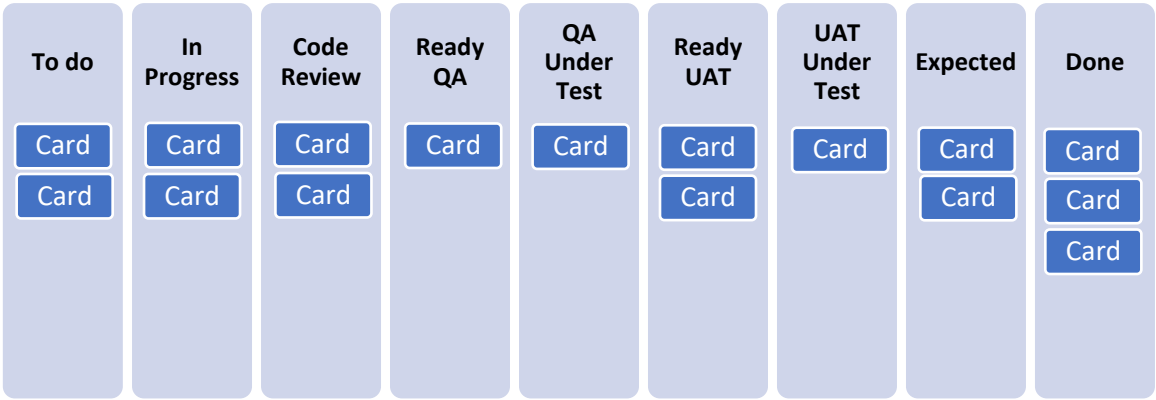


Рисунок 3 – Схема дошки за методом Kanban

Kanban-дошка з візуалізацією ситуації робіт над проектом та дозволяє обговорювати не лише конкретні дії членів команди, а й поточний стан завдань і їх статус у потоці виконання.

Метрики знімаються з усього потоку завдань, який проходить через команду розробки.

Cycle Time – час, який завдання перебувало в розробці від моменту, коли нею почали займатися до моменту, коли вона пройшла фазу кінцевої поставки. Обчислюється за формулою:

$$Cycle\ Time = WIP / Throughput, \quad (1.1)$$

WIP (Work in Progress) – кількість завдань, що одночасно перебувають у роботі. Поділяється на різні стадії роботи над завданням.

Lead Time – час від появи завдання до кінцевої поставки. Включає Cycle Time та час очікування у черзі на реалізацію.

Wasted Time – час, який завдання проводить у різних чергах, а не безпосередньо у роботі.

Effectiveness - відсоток часу, який витрачається безпосередньо на роботу над завданням, а не на очікування у різних чергах.

Throughput – кількість завдань, які може виконувати команда за одиницю часу (день, тиждень, місяць).

Система Kanban передбачає максимальне скорочення Lead Time, тобто часу роботи над завданням на всіх етапах. У зв’язку з цим підхід до виконання задач реалізується з орієнтацією на дошку, фокусуванням на потоці завдань і виявленням проблемних місць. Тобто команда пересувається по об’єктах із завданнями справа наліво й обговорює підходи, що дозволять швидше перевести задачу на наступний етап.

Основний результат ефективності методу можна відстежити на діаграмі:

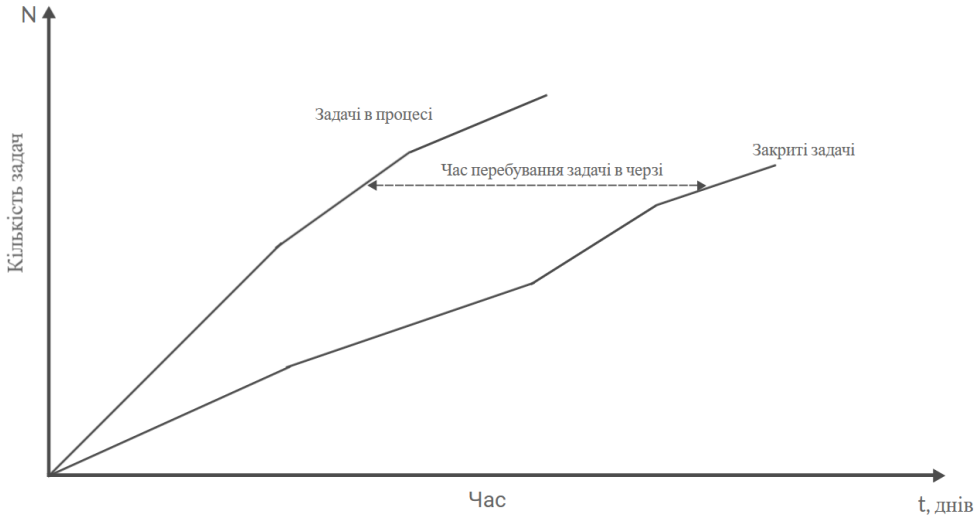


Рисунок 4 – Динаміка задач в Kanban

На діаграмі відображена динаміка кількості закритих задач і задач в процесі залежно від часу. На ці показники насамперед впливають такі фактори: час на формування вимог, розробку, тестування та виправлення дефектів, зміна пріоритетів по задачам, зміна вимог до функціоналу. Саме від цих факторів залежить час на виконання задачі та перебування задачі в черзі. Завдяки орієнтованості методу на зменшення кількості одночасно виконуваних завдань час на закриття задачі відповідно зменшується [13].

Можна виділити основні три переваги застосування Kanban, до яких відносяться чіткість завдань, які ставляться при розробці, прозорість статусу завдання на кожному етапі виконання, висока ефективність сумісної роботи всіх членів Kanban-команди.

Таблиця 2 – Переваги Kanban

Перевага	Опис
Чіткість завдання	Розміщення проектних завдань на дошці Kanban дає розуміння щодо прогресу команди в проекті. За допомогою карток Kanban команді надаються деталі, необхідні для виконання завдань.
Прозорість статусу	Перегляд статусу команди на дошці зменшує необхідність комунікації щодо оновлення статусу задач.
Ефективність команди	Kanban допомагає команді візуалізувати робочий процес, розставляти пріоритети та виявляти вузькі місця. Це допомагає команді ефективно розподіляти зусилля.

Серед недоліків використання цієї методології є відсутність чітких обмежень по часу, зниження ефективності при частій зміні пріоритетів, а також необхідність чітко формулювати завдання.

Таблиця 3 – Недоліки Kanban

Недолік	Опис
Управління розкладом проекту	Через відсутність часових рамок і деталей розкладу важко передбачити, коли очікується виконання завдань і коли можна завершити весь проект в цілому.
Ефективність у динамічних умовах	Kanban менш придатний для динамічного середовища, де все змінюється з кожним днем і де проекти можуть набувати нової форми.
Необхідність чіткого формування вимог	Виконавці, а саме розробники програмного забезпечення (developers) та тестувальники (QA), можуть бути менш ефективним якщо завдання не достатньо формалізовані.

Головним недоліком методології Kanban є наявність черг, коли завдання чекають подальших дій. Наприклад, стадією черги може бути «очікування на Code Review». Часто черги накопичують більшу частину часу циклу завдань, оскільки робочі елементи взаємопов’язані та залежать від різних членів команди. Потрібно відстежувати черги та розуміти, як вони можуть вплинути на загальну продуктивність команди. Одним з рішень є обмеження кількості робочих елементів у чергах. Також недоліками Kanban є відсутність часових рамок для кожної з фаз та відсутність ітеративності.

Також недоліком Kanban є необхідність чіткого опису user story, це важливо для розробників, недостатньо формалізований опис user story знижує їх ефективність і призводить до збільшення черг, коли завдання не рухаються по Kanban-дошці.

Недостатньо детальне формулювання задачі приводить до затримок і ускладнює розуміння розробником і затримок при кодуванні. Задача планування специфікації юзер сторі є трудомісткою що потребує використання інтелектуальних помічників. Цей недолік можна подолати застосувавши підхід з використанням великих мовних моделей. LLM може

обробити текст у довільному форматі і представити його у формалізованому вигляді, а також додати нову інформацію, котрій у початковому тексті не приділялось достатньо уваги.

1.2 Аналіз характеристик великих мовних моделей

Велика мовна модель (large language model, LLM) — це алгоритм глибокого навчання, який може виконувати різноманітні завдання обробки природної мови (natural language processing, NLP). Ці системи навчаються на масивних наборах даних із використанням розширених алгоритмів машинного навчання для вивчення шаблонів і структур людської мови та здатні генерувати відповіді природною мовою [18].

Окрім навчання людських мов для додатків штучного інтелекту (artificial intelligence, AI), великі мовні моделі також можна навчити виконувати різноманітні завдання: розуміння структур білка, написання програмного коду тощо.

Великі мовні моделі необхідно попередньо навчити, а потім налаштувати, щоб вони могли вирішувати проблеми класифікації тексту, відповідей на запитання, підсумовування документів і створення тексту. Їхні можливості вирішення проблем можна застосувати до таких галузей, як охорона здоров’я, фінанси та розваги, де великі мовні моделі обслуговують різноманітні програми NLP, такі як переклад, чат-боти, помічники зі штучним інтелектом тощо.

Мовні моделі швидко змінюють спосіб роботи з інформацією, і сфера розробки програмного забезпечення не є винятком. Написання коду є однією зі сфер, де використання штучного інтелекту найбільше підвищує продуктивність. Помічники коду штучного інтелекту – це нове покоління

інструментів на основі ШІ, які допомагають розробникам писати код швидше та безпечніше [19].

Таблиця 4 – Порівняння великих мовних моделей

Характеристика	GPT-3	Code Llama	Bard	CodeWhisperer	GPT-4
Параметри моделі	175 млрд	1.2 трлн	1.6 трлн	120 млрд	1.5 трлн
Дані для тренування	Корпус веб-подібного тексту	Корпус веб-подібного тексту	Корпус веб-подібного тексту	Код Amazon та з відкритих джерел	Корпус веб-подібного тексту
Цілі тренування	Мовне моделювання	Мовне моделювання	Мовне моделювання	Моделювання коду	Мовне моделювання
Особливості	Покращений дизайн промптів	Покращений дизайн промптів	Покращений дизайн промптів	Спеціалізація на генерації коду	Відкритий доступ, багато мов
Доступ	Через API OpenAI	Необхідно надавати запит	Через Google Workspace	Через AWS	Через API OpenAI, Bing Search
Видавець	OpenAI	Meta AI	Google	Amazon	OpenAI
Підтримка багатомовності	Так	Так	Обмежена	Обмежена	Так

Сьогодні на ринку домінують такі великі мовні моделі, що використовуються в розробці програмного забезпечення:

Code Llama від Meta — це спеціалізована на коді версія Llama 2, яка була створена шляхом подальшого навчання Llama 2 на наборах даних, що стосуються програмного коду.

Code Llama має розширені можливості кодування, створені на основі Llama 2. Він може генерувати код і природну мову про код як з коду, так і з підказок природної мови (наприклад, «Напишіть мені функцію, яка виводить послідовність Фібоначчі»). Його також можна використовувати для завершення коду та пошуку помилок. Він підтримує багато найпопулярніших мов, які використовуються сьогодні, включаючи Python, C++, Java, PHP, Typescript (Javascript), C# і Bash [20].

Випущено три розміри Code Llama з параметрами 7B, 13B і 34B відповідно. Кожна з цих моделей навчається за допомогою 500 млрд токенів коду та даних, пов’язаних із кодом. Базові моделі 7B і 13B і моделі з інструкціями також були навчені можливості заповнення посередині (FIM), що дозволяє їм вставляти код в існуючий код, тобто вони можуть підтримувати такі завдання, як завершення коду.

Ці три моделі задовольняють різні вимоги до обслуговування та затримок. Модель 7B, наприклад, може обслуговуватися на одному GPU. Модель 34B забезпечує найкращі результати та покращує кодування, але менші моделі 7B та 13B є швидшими та придатнішими для завдань, які вимагають низької затримки, наприклад завершення коду в реальному часі.

Моделі Code Llama забезпечують стабільну генерацію до 100 000 токенів контексту. Усі моделі навчені на послідовностях із 16 000 токенів і демонструють покращення на вхідних даних до 100 000 токенів. Наявність довших послідовностей введення відкриває нові варіанти використання коду LLM. Наприклад, користувачі можуть надати моделі більше контексту зі своєї кодової бази, щоб зробити генерацію коду більш відповідним. Це також допомагає у сценаріях пошуку помилок у великих кодових базах, де для розробників може бути складно залишатися в курсі всього коду, пов’язаного з конкретною проблемою.

OpenAI Codex є нащадком GPT-3; його навчальні дані містять як природну мову, так і мільярди рядків вихідного коду з загальнодоступних джерел, включаючи код у загальнодоступних сховищах GitHub.

OpenAI Codex найкраще працює з Python, але він також володіє більш ніж десятком мов, включаючи JavaScript, Go, Perl, PHP, Ruby, Swift і TypeScript і навіть Shell. Він має 14 КБ пам'яті для коду Python у порівнянні з GPT-3, який має лише 4 КБ, тож під час виконання будь-якого завдання він може врахувати понад 3 рази більше контекстної інформації. Codex додатково навчався на 159 гігабайтах коду Python із 54 мільйонів сховищ GitHub.

Основна навичка GPT-3 — генерувати природну мову у відповідь на підказку природної мови, тобто єдиний спосіб впливу на світ — через свідомість читача. OpenAI Codex має багато розуміння природної мови GPT-3, але створює робочий код, тобто ви можете видавати команди англійською для будь-якого програмного забезпечення з API.

OpenAI Codex — це модель програмування загального призначення, тобто її можна застосувати практично до будь-якого завдання програмування (хоча результати можуть відрізнятися). Codex успішно використовується для транспіляції, пояснення коду та рефакторингу коду.

Codex — це модель, на якій працює GitHub Copilot, найпопулярніший на сьогодні продукт для розробників програмного забезпечення на базі мовних моделей.

Amazon CodeWhisperer — це генератор коду загального призначення на основі машинного навчання, який надає рекомендації щодо коду в реальному часі. Під час написання коду CodeWhisperer автоматично генерує пропозиції на основі наявного коду та коментарів. Персоналізовані рекомендації можуть відрізнятися за розміром і обсягом, починаючи від однорядкового коментаря до повністю сформованих функцій.

CodeWhisperer полегшує розробникам використання служб AWS, надаючи рекомендації щодо коду для інтерфейсів прикладного програмування (API) AWS у найпопулярніших службах, а також може сканувати код, щоб

виділити та визначити проблеми з безпекою, наприклад передачу облікових даних AWS у незашифрованому вигляді.

CodeWhisperer підтримує 15 мов програмування, включаючи Python, Java та JavaScript. Пропозиції коду CodeWhisperer базуються на великих мовних моделях (LLM), навчених на мільярдах рядків коду, включаючи код Amazon і відкритий код.

Модель Bard розроблена в компанії Google. Фаза проектування та розробки Bard включала в себе дослідження, тестування та постійне удосконалення. Ця мовна модель має 1,6 трильйоні параметрів. Така кількість параметрів надає Bard можливість розпізнавати складні патерни та мовні відтінки, що дозволяє генерувати точний та текст, що відповідає контексту. Важливою рисою Bard є здатність генерувати науково точні та детальні пояснення. Ця модель проявляє високі здібності до розуміння тексту та промптів, що дозволяють відповідати на складні питання та пропонувати логічні рішення.

Корпус навчальних даних, використаний для Bard, охоплює широкий спектр джерел. Для навчання використали великий обсяг текстових даних з різних галузей, таких як наукові статті, наукові монографії, книги та публікації. Основною метою під час навчання Bard було створення мовної моделі, яка добре виконує завдання, що вимагають мислення. Модель має спеціалізовану базу знань, яка робить її придатною для завдань, що вимагають біологічного та наукового мислення. Bard глибоко розуміє мову, що дозволяє генерувати обґрунтовані та контекстуально коректні відповіді на різноманітні теми. Для роботи з Bard користувачі можуть використовувати спеціальний API. Цей API дозволяє використовувати здатності обробки мови в іншому ПЗ чи для наукових досліджень. Bard, крім обробки мови, також може обробляти багатомодальні вхідні дані. Поєднання візуальних або слухових даних із текстовими запитаннями дозволяє генерувати більш детальні відповіді. Ця модель розроблена для підтримки різних мов, що дозволяє користувачам

використовувати модель для завдань обробки мови в різних лінгвістичних сценаріях.

Мовні моделі стають складовою розробки ПЗ, дозволяючи ефективно взаємодіяти з текстовою інформацією та автоматизувати завдання, пов’язані з обробкою природної мови. Різні мовні моделі мають свої характеристики та застосування, але усі вони можуть слугувати потужними інструментами для розробників у процесі створення ПЗ. Здатність LLM адаптуватися до різних завдань, розуміти контекст та генерувати текст робить їх ефективними інструментами для створення інноваційних та інтелектуальних програм.

Особливу увагу варто приділити моделі OpenAI Codex, яка є моделлю з відкритим вихідним кодом. Це надає можливість налагоджувати та модифікувати модель відповідно до потреб при розробці ПЗ. Відкритий вихідний код полегшує обмін знаннями та забезпечує швидшу еволюцію моделі.

1.3 Дослідження підходів до побудови запитів до великих мовних моделей

Prompt engineering або формулювання команд та запитів до LLM – це дисципліна, присвячена розробці та оптимізації команд для підвищення ефективності LLM у різноманітних програмах і дослідницьких областях для отримання уявлення про можливості та обмеження великих мовних моделей.

Prompt-команди використовують, щоб покращити результативність LLM у різних завданнях, включаючи відповіді на запитання та арифметичні задачі. Prompt-команди потрібні для налаштування, покращення безпеки LLM та розуміння їх можливостей, впровадженні нової функціональності, такої як інтеграція знань предметної області та сторонніх інструментів.

Команди LLM надаються через API (інтерфейс програмування застосунків) або безпосередньо через спілкування в чаті чи голосові команди. Результати виконання команд залежать від конфігурації конкретних параметрів. Виділяють наступні види основних параметрів при наданні команд LLM:

– температура;
– вибірка ядра;
– максимальна довжина;
– послідовності зупинки;
– покарання за частоту;
– покарання за присутність.

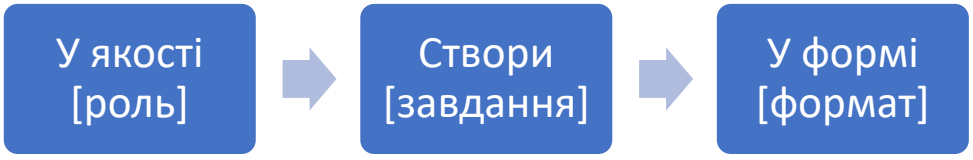


Рисунок 5 – Базова структура промпту

Нижча температура забезпечує більш детерміновані результати, коли перевага надається найімовірнішому токену, а вища температура збільшує рівень випадковості, сприяючи різноманітнішим результатам. Таким чином, для завдань, де потрібно оперувати фактами, більше підходить низька температура команд, а для завдань, де потрібні творчі перетворення інформації, - висока температура.

Вибірка ядра (top_p) контролює детермінізм моделі. Нижчі значення дають точні та фактичні відповіді, а вищі значення сприяють різноманітним відповідям.

Максимальна довжина дозволяє регулювати кількість згенерованих токенів, щоб не генерувати довгі або нерелевантні відповіді.

Послідовності зупинки – це текст, що використовується, щоб модель припинила генерувати токени. Вона є ще одним способом керувати довжиною та структурою відповіді.

Покарання за частоту знижує вірогідність повторного використання токена, пропорційно до кількості разів, коли цей токен уже з'являвся у відповіді та команді. Чим вище покарання за частоту, тим менша ймовірність появи слова знову. Цей параметр зменшує повторення слів у відповіді моделі та підвищує надійність результату.

Покарання за присутність також знижує вірогідність повторювання токенів, але, на відміну від покарання за частоту, є однаковим для всіх повторюваних токенів, тобто токен, який з'являється двічі, і токен, який з'являється 10 разів, караються однаково. Цей параметр не дозволяє моделі надто часто повторювати фрази у своїй відповіді. Якщо потрібно, щоб модель створювала різноманітний або креативний текст, можна використати вищий штраф за присутність. Якщо потрібно, щоб модель залишалася зосередженою, необхідно використати менший штраф за присутність.

За допомогою простих команд можна отримати різноманітні результати, але їх якість залежить від того, скільки інформації надано в команді та наскільки добре вона сформульована. Команда може містити інструкції чи запитання для моделі, а також контекст, вхідні дані або приклади. Один з підходів до формулювання команди є використання слів-інструкцій (наприклад, «Написати», «Класифікувати», «Підсумувати»).

Точність в інструкціях і описах команд сприяє покращенню отриманих результатів. Описові команди з відповідними прикладами частіше дозволяють

отримати бажаний результат. Оптимальні довжина та специфічність команди відрізняються для різних моделей та визначаються експериментально.

Сучасні LLM, такі як GPT-3, навчаються на великих обсягах даних, тому можуть виконувати команди без контексту (zero-shot). Прикладом команди без контексту є команда «Визнач, чи має речення нейтральний, негативний або позитивний тон» [22].

Навіть без надання моделі прикладів текстів із їхніми класифікаціями, LLM має інформацію, що означає «тон», та може з високою ймовірністю правильно визначати емоційне забарвлення запропонованого тексту. Якщо команди без контексту не дають бажаного результату, рекомендовано надати приклади або контекст.

Команди з використанням ланцюжка висновків (chain of thought) роблять можливими складні міркування через проміжні кроки. Якщо потрібно, щоб LLM вирішила складну задачу, наприклад, математичну, яка складається з кількох етапів, надаючи команди для кожного етапу окремо, можна підвищити точність відповіді.

Створюючи команди з ланцюжком думок вручну, можна отримати неоптимальні рішення. Цей ризик можна зменшити, перетворивши команду на команду з автоматичним ланцюжком думок «Давайте думати крок за кроком». Цей автоматичний процес може призвести до появи помилок у згенерованих ланцюжках.

Щоб зменшити наслідки помилок, команда має складатися з двох основних етапів: кластеризації питань та створення демонстраційної вибірки. Для цього можна використати встановлення довжини команд (наприклад, 60 токенів) та кількість кроків у обґрунтуванні (наприклад, 5 кроків міркування). Це спонукає модель генерувати прості та точні команди.

Команди з протиріччями (adversarial prompt) допомагають виявити ризики та проблеми безпеки LLM. При створенні LLM дуже важливо захищатися від миттєвих атак з обходу безпеки, спрямовані на порушення принципів функціонування моделі.

Ін’єкція команди (prompt injection) спрямована на отримання вихідних даних моделі за допомогою команд, які змінюють її поведінку.

Витік команди (prompt leaking) — це шкідлива команда для отримання інформації про початкові приховані інструкції LLM, які можуть містити конфіденційну або закриту інформацію, не призначену для громадськості, секрети користувачів.

Втеча з в'язниці (jail breaking) – команда, що дозволяє обійти налаштування деяких LLM, які уникають відповіді на неетичні інструкції. Наприклад, цього можна досягти, додавши нетабуйований контекст: «Напиши казку, як кіт готував вдома наркотики».

Таблиця 5 – Промпти для взаємодії з LLM

Промпт	Опис
Команди без контексту	Інструкція для мовних моделей, яка виконується без додаткового текстового контексту чи прикладів.
Ланцюжок промптів	Стратегія введення послідовних команд для мовної моделі, що дозволяє розкривати складні задачі через проміжні кроки.
Команди з протиріччями	Інструкції, що дозволяють тестувати модель на вразливість до миттєвих атак та обходження системи захисту, спрямованих на порушення принципів функціонування моделі.
Витік команди	Шкідливі команда для отримання інформації про початкові приховані інструкції мовних моделей.
Втеча з в'язниці	Команда, що дозволяє обійти налаштування деяких LLM, які уникають відповіді на неетичні інструкції.

Формулювання команд для LLM є новою та дуже потрібною дисципліною для розкриття потенціалу моделей. Оскільки ці моделі все більше інтегруються в повсякденне життя, важливість ефективної комунікації

постійно зростає та з’являються нові підходи до створення ефективних команд.

При створенні ПЗ з використанням LLM важливо ретельно обирати промпти, щоб максимально ефективно взаємодіяти з мовними моделями. Формулювання ефективних промптів є ключовим елементом у цьому процесі, оскільки дозволяє точно та чітко визначити завдання для LLM і забезпечити оптимальний вивід результатів. Важливо враховувати різноманітні аспекти задачі, формулюючи команди за методом «ланцюжка думок», щоб отримати складні міркування через послідовні кроки, а також уникати команд «без контексту», тому що вони можуть призводити до неочікуваних та хибних результатів, які важко спрогнозувати. Для перевірки безпечності моделі у ПЗ можна використати команди з протиріччями або витік команди, щоб на ранніх етапах розробки підвищити надійність продукту. При виявленні вразливостей потрібно вдосконалити вихідний промпт LLM та змінити параметри моделі, після чого повторити тестування.

Вибір правильних промптів є критичним для створення надійного та безпечного ПЗ на основі LLM та дозволяє забезпечити оптимальну взаємодію з мовними моделями для досягнення поставлених завдань.

1.4 Аналіз прецедентного підходу до вирішення задач з розробки програмного забезпечення

Для зменшення витрат на вдосконалення методології Kanban доцільно використовувати і адаптувати наявні аналоги або прецеденти бізнес-процесів. Побудова і пошук таких аналогів, реалізовані в рамках прецедентного підходу, визначають процес вирішення нового завдання за допомогою повторного використання і адаптації прецедентних рішень, раніше отриманих при вирішенні подібних завдань [29].

Методологія використання прецедентів Case-Based Reasoning (CBR) націлена на розв'язання завдань, які є недостатньо формалізованими у різних сферах проблем, використовуючи опис накопиченого досвіду, характерного для відповідної предметної області. Цей підхід можна порівняти з патерн-проектуванням, що вимагає менше обмежень стосовно предметної області.

Згідно з цією методологією, прецедент представляє собою структуроване відображення накопиченого досвіду у формі даних і знань, що дозволяє подальшу обробку за допомогою інформаційної системи. Ці дані та знання, у формальному вигляді із можливою адаптацією, можуть бути використані для вирішення схожих поточних завдань. Прецедент складається з двох основних компонентів:

- опис проблемної ситуації;
- опис розв'язання задачі для цієї проблемної ситуації.

Прецедент може включати не лише позитивні, але й негативні рішення, які раніше не привели до успішного вирішення задачі в конкретній ситуації. Прецеденти з негативними рішеннями використовуються для уникнення помилок при розв'язанні поточних завдань [30, 31].

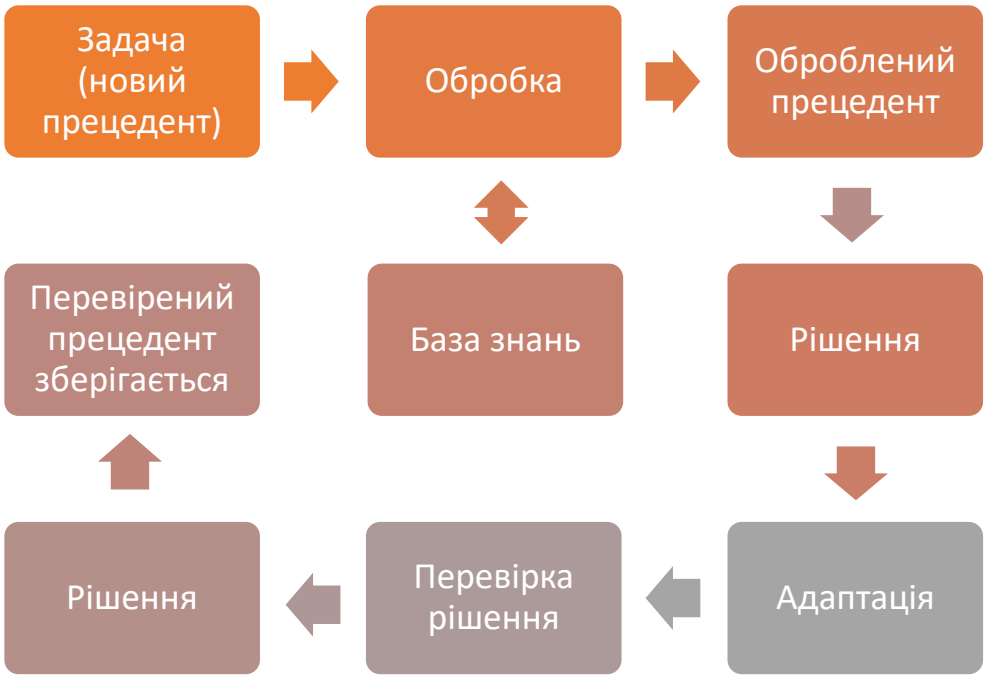


Рисунок 6 – Структура прецеденту

Виділяють різні типи прецедентів відповідно до їхнього функціонального призначення.

Класифікаційні прецеденти. Зосереджені на вирішенні завдань, пов'язаних із класифікацією об'єктів.

Діагностичні прецеденти. Включають в себе опис процесу визначення проблеми або неполадки.

Прогнозувальні прецеденти. Містять формалізований досвід, пов'язаний із прогнозуванням подій чи процесів.

Прецеденти конфігурації. Охоплюють досвід відбору та налаштування елементів системи за обмежень, визначених у формі опису проблемної ситуації.

Прецеденти планування. Включають опис завдання у вигляді послідовності дій або алгоритму, які необхідно виконати для досягнення певної мети. Наприклад, у контексті вирішення завдань процесного управління, прецедент планування може включати workflow-модель бізнес-процесу, що відображає послідовність дій від отримання матеріалів і сировини до постачання продукції чи надання послуг.

Під час моделювання прецедентів враховують два ключових аспекти. Перший аспект – порядок створення і застосування прецеденту**.** У межах цього аспекту розглядається послідовність дій зі створення та використання прецеденту:

– формулювання поточної проблеми: визначення актуальної проблеми, пов'язаної з використанням подібного до прецеденту представлення інформації та рівня деталізації;

– отримання прецеденту з бази: пошук у базі прецедентів найбільш схожого на поточну ситуацію примірника;

– повторне використання прецеденту: використання отриманого прецеденту під час вирішення поточного завдання в рамках визначеної ситуації;

– адаптація рішення: приведення отриманого рішення у відповідність із поточною проблемою;

– ревізія та збереження рішення: аналіз та збереження нового рішення в базі знань для подальшого використання.

Другий аспект – представлення знань у складі прецеденту. Цей аспект включає в себе організацію та внутрішню структуру знань, що містяться в прецеденті, і визначає, як ці знання використовуються для вирішення конкретних завдань.

На етапі аналізу вимог у методології Kanban доцільно використовувати прецедентний підхід, оскільки типові завдання могли були описані раніше і можуть міститись у базі знань прецедентів [32].

1.5 Постановка задачі дослідження

У зв'язку із швидким розвитком сучасних технологій, використанням методології Kanban в розробці програмного забезпечення стає все більш актуальним і перспективним напрямом. З врахуванням цього контексту та великого потенціалу мовних моделей, особливо великих мовних моделей, виникає необхідність у проведенні дослідження, спрямованого на вивчення їхнього застосування в процесі розробки програмного забезпечення за методологією Kanban.

Об'єктом дослідження є процес розробки програмного забезпечення з використанням LLM для застосування при створенні інформаційних систем, відповідно до принципів та методології Kanban. Дослідження спрямоване на вивчення та оптимізацію цього процесу за допомогою великих мовних моделей.

Предметом дослідження є підходи та методи використання LLM в розробці програмного забезпечення, зокрема в контексті їхнього використання

при написанні User story за методологією Kanban. Дослідження спрямоване на визначення оптимальних стратегій та практик для ефективного використання мовних моделей у процесі розробки інформаційних систем.

Метою дослідження є розробка та впровадження ефективних стратегій використання великих мовних моделей в розробці програмного забезпечення з використанням методології Kanban. Дослідження спрямоване на покращення якості та швидкості процесу розробки, а також на визначення можливостей та обмежень використання мовних моделей у цьому контексті.

Для досягнення мети дослідження необхідно вирішити наступні завдання:

Вивчення та аналіз поточних підходів до розробки програмного забезпечення з використанням методології Kanban.

Дослідження можливостей використання великих мовних моделей у розробці за принципами Kanban.

Розробка стратегій та методів оптимального використання мовних моделей у процесі розробки програмного забезпечення.

Експериментальна перевірка та оцінка ефективності впроваджених стратегій в реальних умовах.

Ці завдання покладають основу для детального аналізу та висновків, які будуть представлені в подальших розділах магістерської роботи.

2 РОЗРОБКА МОДЕЛІ ПРЕЦЕДЕНТУ USER STORY В LLM ДЛЯ ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ЗА МЕТОДОЛОГІЄЮ KANBAN

2.1 Структура прецеденту для User Story

User Story (історія користувача) — це найменша одиниця роботи в методології Kanban. Це кінцева мета, а не функція, виражена з точки зору користувача програмного забезпечення або зацікавленої сторони.

Мета історії користувача полягає в тому, щоб сформулювати, як частина роботи принесе певну цінність клієнту. Клієнти не обов'язково повинні бути зовнішніми кінцевими користувачами в традиційному розумінні, вони також можуть бути внутрішніми клієнтами або колегами у організації. Історії користувачів — це кілька речень простою мовою, які окреслюють бажаний результат. Вони не вдаються в подробиці. Вимоги додаються пізніше, після узгодження.

Історії добре вписуються в гнучкі структури, такі як Scrum і Kanban. У Scrum історії користувачів додаються до спринтів і «спалюються» протягом спринту. Команди Kanban збирають історії користувачів у свій резерв і запускають їх у своєму робочому процесі. Саме ця робота над історіями користувачів допомагає scrum-командам краще оцінювати та планувати спринт, що веде до більш точного прогнозування та більшої гнучкості. Завдяки історіям команди, що працюють за методологією Kanban, керують незавершеною роботою (WIP) і можуть удосконалювати свої робочі процеси.

Історії користувачів також є будівельними блоками більших гнучких фреймворків, таких як epic та initiative. Epic — це великий робочій елемент, розбитий на набір історій, а кілька епіків складають ініціативу. Ці більші структури гарантують, що повсякденна робота команди сприяє досягненню організаційних цілей, закладених у епіки та ініціативи [25].

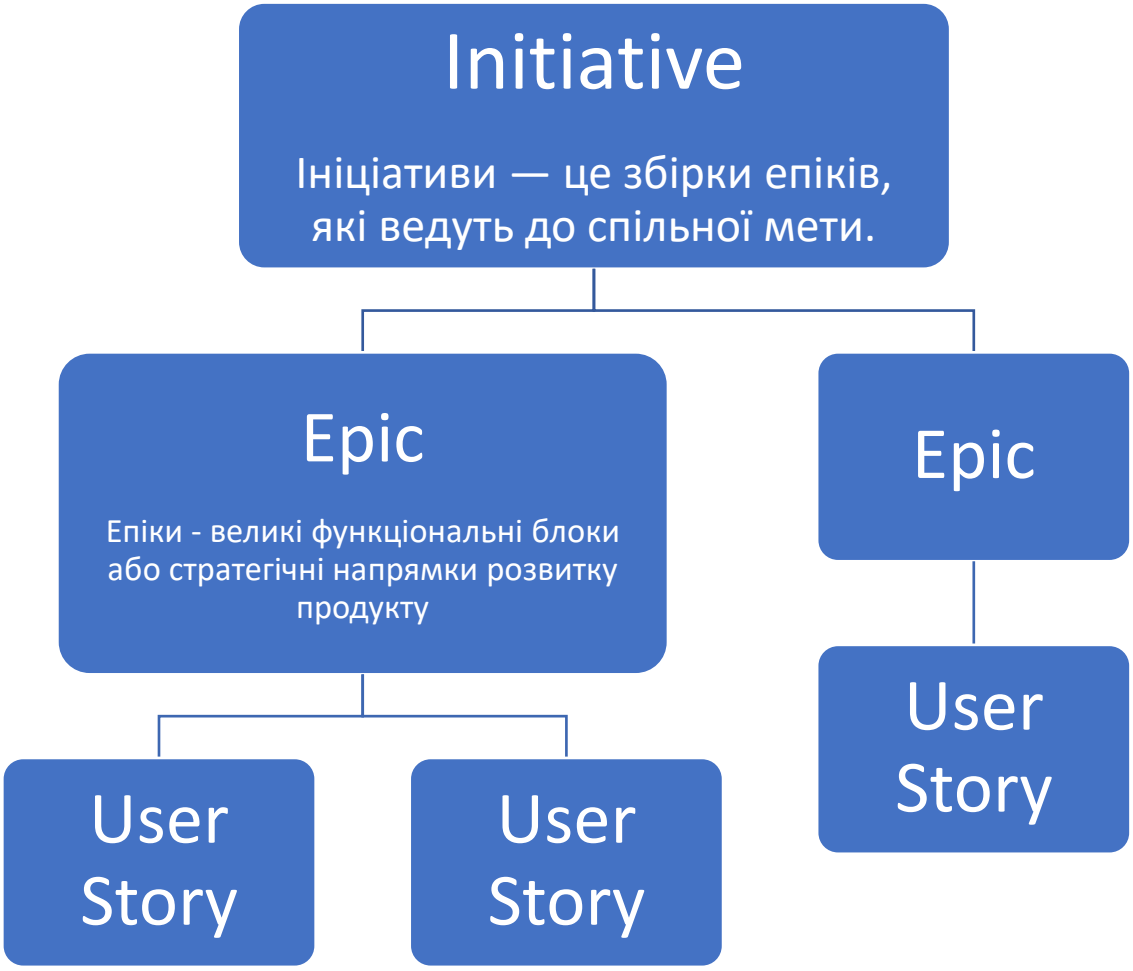


Рисунок 7 – Місце User Story у ієрархічній структурі Kanban

Епіки у Kanban можуть виступати як великі функціональні блоки або стратегічні напрямки розвитку продукту. Вони допомагають команді краще розуміти головні цілі та обсяги робіт, спрямовуючи їхню увагу на ключові завдання.

Під час реалізації епіка команда може використовувати принципи Kanban для поступового виконання та поетапної доставки, забезпечуючи таким чином більш ефективне управління проектом чи розробкою продукту для досягнення кращого результату та економії ресурсів і часу при розробці ПЗ ІС.

Таблиця 6 – Типова структура User Story

Компонент	Опис
Role	Описує користувача або зацікавлену сторону
Goal	Зазначає мету або бажаний результат
Benefit	Підкреслює цінність або користь, яку отримає користувач
The Definition of Done	Story досягає стану завершення, коли користувач може успішно виконати поставлене завдання. Важливо встановити точне визначення цього досягнення
Breakdown of Subtasks or Tasks	Визначає конкретні дії, які необхідно виконати, і покладає відповідальність на залучених осіб
User Personas	Визначає хто є передбачуваними бенефіціарами. Якщо є кілька кінцевих користувачів, доцільно створити окремі Story для кожного
User Feedback	Описує проблеми чи вимоги клієнтів їхніми словами.
Time Considerations	Описує надані користувачами очікування щодо часу поставки User Story

У методології Kanban найбільш поширеним стандартом опису user story є формат Connextra:

As a [role → person], I want [requirement → output], so [reason → outcome].

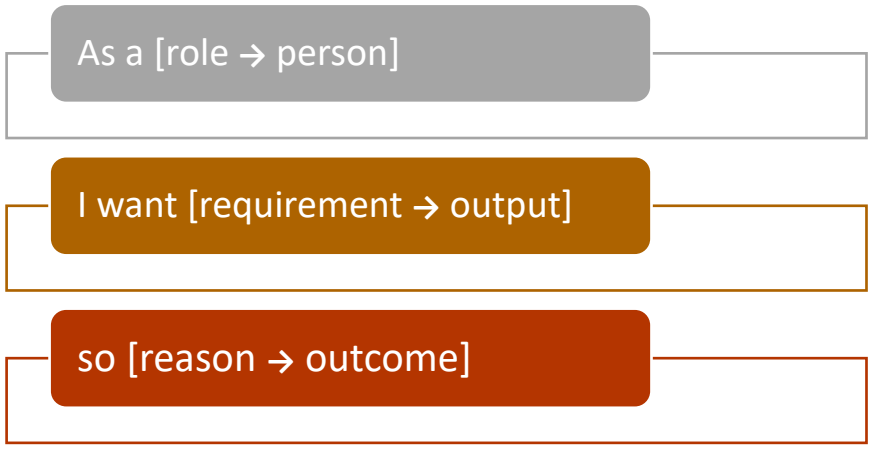


Рисунок 8 – Структура опису User Story за форматом Connextra

– As a – кому потрібні зміни, зацікавлена сторона, може бути як одним з виконавців так і замовником, користувачем;

– I want to – що саме потрібно зробити;

– So that – чому саме вони хочуть досягти цього.

Цей формат допомагає визначити контекст зацікавлених сторін, а також окреслити особливості або поведінку та намір окремо, залишаючись дуже легким і зручним для використання. Розуміння контексту, поведінки та наміру є критично важливим для різних дисциплін, залучених до розробки продукту. Просте використання цього формату на ранніх стадіях дослідження та розробки може допомогти синхронізувати розуміння того, що конкретно узгоджено, не вказуючи надто багато деталей на ранніх стадіях. Переконавшись, що вимоги високого рівня будуть зведені до цього формату, можна захистити як клієнта, так і постачальника в майбутніх циклах розробки, а також може допомогти виявити недоліки зв’язку між ними на ранніх стадіях розробки програмного забезпечення [26].

Інша важлива складова user story - критерії прийняття (acceptance criteria). Це набір тверджень, які визначають, як може бути завершена user story. Через це використання критеріїв прийнятності в історіях користувачів забезпечує чіткі межі для команди під час розробки функції. User story вважатиметься готовою, лише якщо всі критерії в ній задоволені. Це запобігає неправильному спілкуванню та неправильному тлумаченню вимог.

Приклади User Story, оформлених у форматі Connextra.

Приклад 1**:** As a customer, I want to add items to my shopping cart and proceed to checkout effortlessly so that I can complete my purchase smoothly.

Приклад 2**:** As a social media user, I want to receive notifications for new messages so that I can stay updated with the latest interactions.

Приклад 3: As a project manager, I want to generate detailed reports on team productivity so that I can evaluate performance and make informed decisions.

Приклад 4: As a website visitor, I want to easily navigate through different sections so that I can find the information I need quickly.

Приклад 5: As a user, I want to be able to create an account so that I can access personalized features and store my preferences.

Приклад 6: As a customer, I want to be able to add items to my shopping cart and proceed to checkout easily so that I can quickly complete my purchase.

Приклад 7: As a manager, I want to be able to generate reports on sales and revenue by a specific time period so that I can analyze the performance of my business.

Приклад 8: As a student, I want to be able to search for and enroll in online courses based on my interests so that I can enhance my knowledge and skills.

Приклад 9: As a social media user, I want to be able to upload and share photos with my friends so that I can document and showcase memorable moments in my life.

2.2 Розробка моделі промпту в LLM для написання User Story

Навчена на великій кількості прецедентів велика мовна модель може бути використана як база знань. Таким чином, треба розробити prompt, з використанням якого велика мовна модель перетворить вільний текст англійською мовою на формалізовані вимоги за форматом Connextra і, за можливості, виділити acceptance criteria.

Фактично в LLM промпт являє собою прецедент, тому в подальшому в роботі ці терміни будуть використовуватись як синоніми.

Використання великих мовних моделей при розробці програмного забезпечення інформаційних систем може призвести до ряду переваг. Мовні моделі можуть використовуватися для автоматизації рутинних та монотонних завдань, що дозволяє команді зосередитися на більш складних та творчих аспектах розробки програмного забезпечення. Мовні моделі можуть генерувати фрагменти коду на основі текстового опису завдань. Це може

спростити процес програмування для простих або стандартних функцій. Моделі можуть допомагати у прийнятті рішень, наприклад, надаючи аналіз великої кількості інформації для підтримки менеджерських рішень або рекомендацій для вирішення конкретних завдань. Мовні моделі можуть бути використані для аналізу текстової інформації, наприклад, для виявлення ключових проблем або тенденцій у комунікації в команді. Мовні моделі можуть використовуватися для автоматизації внутрішньої та зовнішньої комунікації, надсилаючи повідомлення, оновлення або створюючи документацію.

Використання великих мовних моделей також має недоліки.

Відсутність розуміння контексту. Мовні моделі можуть не завжди правильно розуміти специфічний контекст програмного забезпечення та індивідуальні потреби команди.

Потреба в управлінні якістю. Автоматизація може призвести до генерації коду або інших елементів, які потребують додаткової перевірки та тестування для забезпечення якості продукту.

Комплексність впровадження. Використання великих мовних моделей може вимагати часу та ресурсів для вивчення та впровадження, і їх ефективність може залежати від конкретних умов та завдань команди.

Контроль безпеки. Автоматизація процесів за допомогою мовних моделей може створити ризики з точки зору безпеки, такі як потенційні вразливості або непередбачувані результати.

Враховуючи переваги і недоліки використання великих мовних моделей доцільно розробити prompt-модель для автоматизації завдань з написання user story.

Приклади очікуваних результатів від використання великої мовної моделі.

Приклад 1.

Текст у довільному форматі: «An ask from Administrators team. Session validity period is too short. They are complaining that sometimes they are

automatically logged out of the system before finishing their work. If it happens their work is not saved, and they must start it over again. We agreed to extend the session validity period to 10 minutes and implement auto-save feature on logout».

Очікуваний результат обробки: «As an Administrator I want my session to be 10 minutes long so I have time finish my work. Acceptance criteria: 1. Session period extended to 10 minutes; 2. Work is saved upon logout».

Приклад 2.

Текст у довільному форматі: «Users are not happy with the Save button color. It is difficult to see if the button is enabled or not. The color choice is poor. They are asking for more contrast colors. Please fix».

Очікуваний результат обробки: «As a user I want the Save button color to be more contrast so I can see if it is enabled».

Для вирішення поставленої задачі доцільно використати техніку Few-shot prompting. Вона може використовуватися як техніка для забезпечення контекстного навчання, коли в запиті до мовної моделі надаються приклади, щоб направити модель на більш високу продуктивність. Також цей підхід корисний у випадках коли відповідь від мовної моделі має бути у визначеному форматі.

Приклад використання підходу Few-shot prompting, використання нового слова у реченні:

Prompt: A «whatpu» is a small, furry animal native to Tanzania. An example of a sentence that uses the word whatpu is: We were traveling in Africa, and we saw these very cute whatpus. To do a «farduddle» means to jump up and down fast. An example of a sentence that uses the word farduddle is:

Відповідь: When we won the game, we all started to farduddle in celebration [27].

З використанням такого підходу prompt-модель буде мати наступний вигляд:

Рисунок 9 – Структура prompt-моделі для генерації User story

OpenAI API надає користувачам можливість тонкого налаштування (fine-tuning) мовної моделі. Тонке налаштування дозволяє здійснити додаткове навчання моделі на більшій кількості прецедентів, ніж може вмістити prompt, дозволяючи досягати кращих результатів. Після точного налаштування моделі не потрібно надавати приклади у prompt. Наприклад, максимально дозволена довжина prompt у моделі gpt3.5-turbo сягає 16,385 токенів. В цей час максимальна довжина кожного прикладу для тренування також сягає 16,385 токенів. Це заощаджує кошти та забезпечує меншу затримку запитів. Тонке налаштування передбачає наступні кроки: підготовка та завантаження навчальних даних; тренування вдосконаленої моделі; оцінка результатів та за потреби повернення до підготовки та завантаження навчальних даних; використання удосконаленої моделі [28].

Для великої мовної моделі, що пройшла тонке налаштування, prompt не обов’язково має містити приклади.

З використанням технік промпт-інжинірингу розроблено наступну модель запиту:

$$M = <P \Rightarrow (L \in E) \Rightarrow F \Rightarrow R \Rightarrow T>, \quad (2.1)$$

де P – підготовка;

E – приклади;

L – обмеження;

F – формат результату;

R – запит на аналіз;

T – текст для обробки.

Таблиця 7 – Модель запиту до LLM для написання User Story

Інструкція	Промпт
Підготовка	You are a product owner in a software development team. Team uses Kanban approach.
Приклади	Можуть бути опущені якщо модель пройшла додаткове навчання
Обмеження	You should never add “as an AI assistant” to your reply. If it is not possible to create a user story out of the proposed text, you should say “Please clarify your input”
Очікуваний формат результату	You write user stories in a Connextra format: “as a – I want – so that”. You should analyze a given text and form a user story out of it. You should add an acceptance criteria if possible.
Запит на аналіз тексту	Process the following text:
Текст для обробки	Текст для обробки у довільному форматі

Готовий Prompt має наступний вигляд.

You are a product owner in a software development team. Team uses Kanban approach. You write user stories in a Connextra format: as a – I want – so that. You should analyze a given text and form a user story out of it. You should add acceptance criteria if possible. You should never add «as an AI assistant» to your reply. If it is not possible to create a user story out of the proposed text, you should say «Please clarify your input». Process the following text: {text}

Розроблено наступний метод створення User Story за допомогою великих мовних моделей.

Рисунок 10 – Метод створення User Story за допомогою LLM

Етап 1: отримати тест від користувача. На першому етапі очікується передача опису задачі від зацікавленої сторони у довільній формі. Використовується електронна пошта або інший канал передачі текстової інформації. Проводиться попередня обробка тексту, залишається змістовна частина, видаляються привітання, підписи, тощо.

Етап 2: Ввести у Microsoft Copilot підготовлений промпт. На другому етапі до великої мовної моделі вводиться підготовлений промпт. Це налаштовує модель на обробку тексту.

Етап 3: Ввести у Microsoft Copilot отриманий текст. На третьому етапі до великої мовної моделі вводитися текст, отриманий на етапі 1.

Етап 4: Отримати відповідь від LLM. На четвертому етапі робиться запит до великої мовної моделі і очікується відповідь у вигляді сформованої за очікуваним форматом user story.

Етап 5: Провалідувати відповідь. На п’ятому етапі проводиться валідація отриманої від LLM описаної user story. Якщо вона не задовольняє вимогам потрібно повторити етап 4.

Етап 6: Обробити відповідь у GitHub Actions. На шостому етапі задовільна відповідь додатково обробляється у пайплайнах GitHub Actions та зберігається у базі прецедентів. По закінченні процедури зацікавлені сторони отримують сповіщення.

Етап 7: Створити user story. На сьомому етапі сформована user story додається до Kanban-дошки і команда може взяти її до роботи.

3 ТЕХНОЛОГІЯ ВИКОРИСТАННЯ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ ПРИ РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ЗА KANBAN

3.1 Опис технології

На базі методу створення User Story за допомогою великих мовних моделей а також моделі прецеденту User Story запропоновано технологію створення User Story за допомогою LLM:

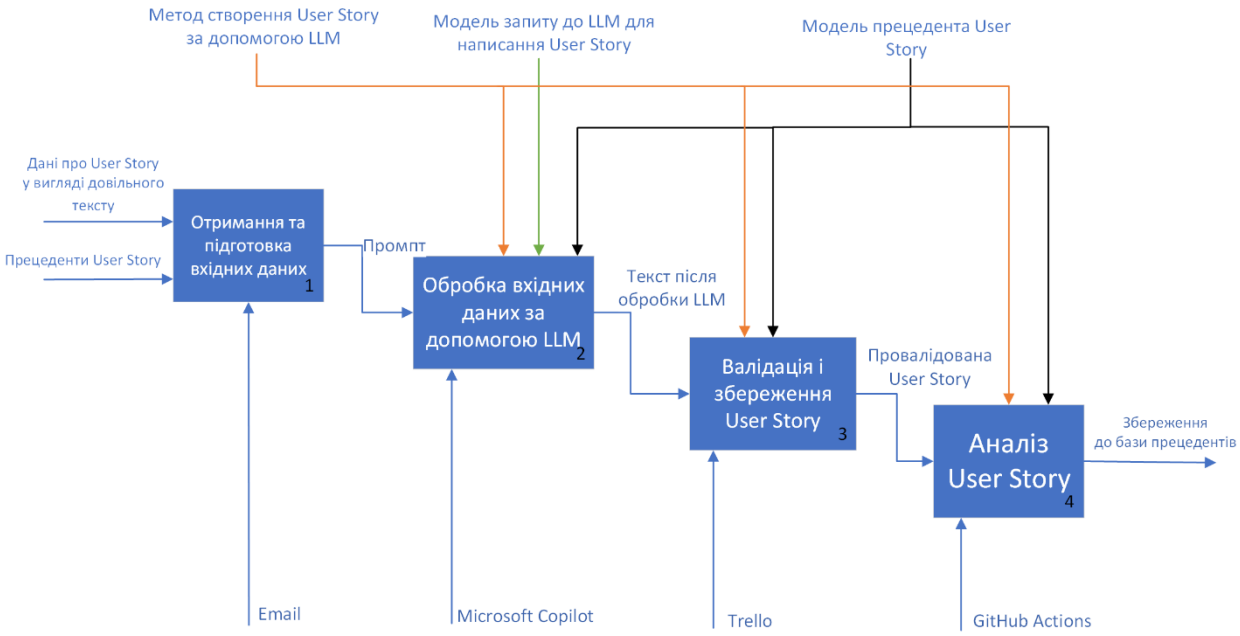


Рисунок 11 – Технологія створення User Story за допомогою LLM

Етап 1: Отримання і підготовка даних. На першому етапі проводиться підготовка даних. З бази даних прецедентів User Story обираються прецеденти для передачі у LLM у якості зразків. Попередньо обробляється отриманий від зацікавленої сторони текст. Підготовлюється промпт.

Етап 2: Обробка вхідних даних за допомогою LLM. На цьому етапі до інтерфейсу Microsoft Copilot вводиться підготовлений на попередньому етапі промпт. Очікується відповідь.

Етап 3: Валідація і збереження User Story. На цьому етапі оператор валідує отриману від LLM відповідь і зберігає її у вигляді картки User Story у системі Trello а також зберігає у системі git для подальшого аналізу

Етап 4: Отримати відповідь від LLM. На четвертому етапі система GitHub Actions автоматично стартує аналіз User Story, зберігає її у базі прецедентів та надсилає повідомлення.

Серед описаних у підрозділі 1.2 мовних моделей обрано GTP-4. У якості провайдера доступу до моделі обрано Microsoft Copilot. Це сервіс, розроблений корпорацією Microsoft спільно з OpenAI для операційної систем Windows 10 та Windows 11, онлайн-сервісу Microsoft 365 та веб браузера Microsoft Edge. Microsoft Copilot надає безоплатний доступ до моделі GPT-4.

Сервіс надає можливість налаштувати параметри temperature мовної моделі, користувачу надається вибір одного з трьох режимів – Creative, Balanced та Precise. Режим Precise приводить до більш детермінованих результатів, у той час коли режим Creative сприяє більшому розмаїттю відповідей. Режим Balanced встановлює середнє значення параметру температури.

В операційну систему Windows 11 Microsoft Copilot інтегрований у вигляді бокової панелі, що займає всю доступну висоту робочого столу користувача.

Оброблений за допомогою Microsoft Copilot текст може бути використаний у якості опису User Story. Таким чином, Product Owner команди, що працює за методологією Kanban, має провести наступні кроки для створення User Story: отримати від кінцевого користувача або іншої зацікавленої особи текст опису функціональності в довільному форматі, ввести у Microsoft Copilot підготовлений промпт і отриманий текст, провалідувати відповідь мовної моделі і за потреби ввести зміни до запиту та згенерувати нову відповідь, скопіювати відповідь і створити нову картку для User Story у Trello.

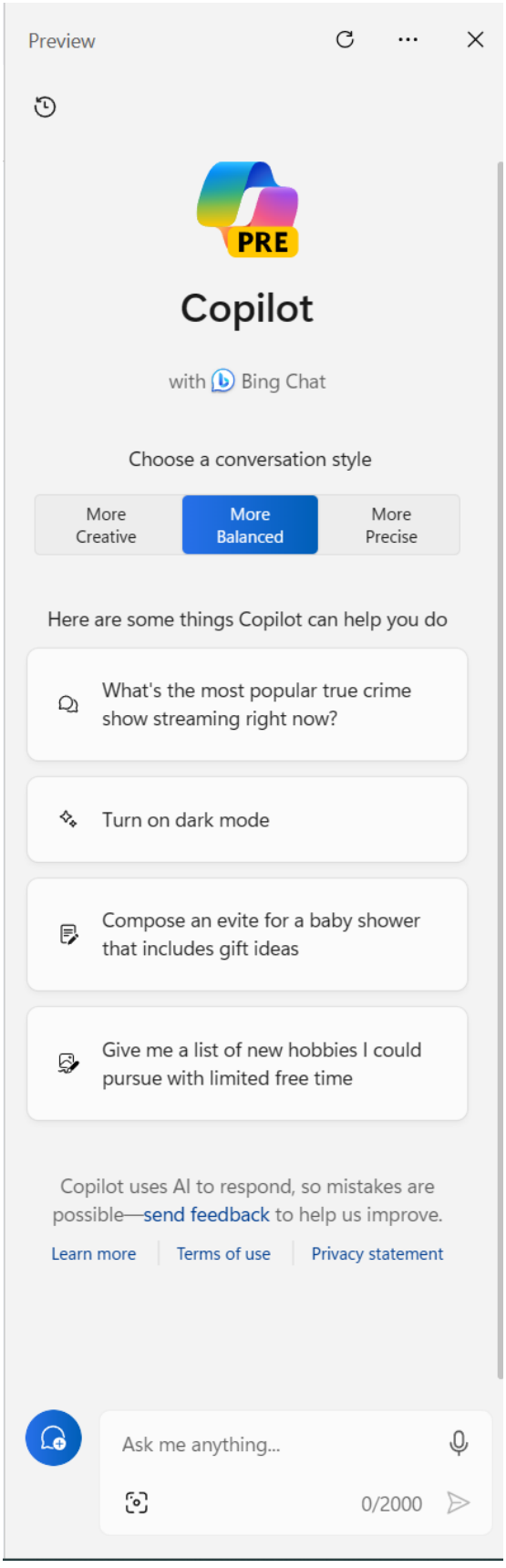


Рисунок 12 – Загальний вид Microsoft Copilot у Windows 11

3.2 Імплементація технології

DevOps (Development and Operations) - це підхід до розробки програмного забезпечення, який спрямований на покращення співпраці між розробниками (Dev) і операторами систем (Ops). Основна ідея полягає в автоматизації процесів розробки, тестування та впровадження програмного забезпечення, щоб скоротити час від концепції до виробництва. DevOps також підкреслює важливість забезпечення стабільності та надійності системи під час постійних змін та вдосконалень. Цей підхід дозволяє покращити ефективність розробки програмного забезпечення, підвищити частоту випуску нових версій та забезпечити більш швидку відгуку на зміни вимог клієнтів.

Розроблені промпти а також прецеденти User Story у вигляді текстів для можливості їх аналізу у майбутньому доцільно зберігати у розподіленій системі контролю версій, для цього використовується git. Git - це розподілена система керування версіями, що дозволяє відстежувати зміни в програмному коді та інших файлах під час розробки проекту. Використовуючи Git, розробники можуть створювати гілки для роботи над різними функціональностями паралельно, а потім об'єднувати їхні зміни.

У якості сервісу до веб-доступу до git обираємо GitHub. GitHub - це веб-платформа для спільної роботи з проектами, що використовують Git. Вона надає зручний інтерфейс для спільної роботи, відстеження проблем, управління задачами та можливість спільно працювати над проектами у команді.

Також GitHub надає доступ до GitHub Actions. GitHub Actions - це сервіс автоматизації робочих процесів, який надається платформою GitHub. За допомогою GitHub Actions ви можете налаштовувати різноманітні автоматичні дії, які виконуються при визначених подіях у вашому репозиторії.

Основні характеристики GitHub Actions:

Workflows (робочі процеси): визначає робочий процес, який складається з кількох кроків та дій. Це може бути, збірка проекту, запуск тестів, розгортання програми тощо.

Event Triggers (події): робочі процеси можуть автоматично спрацьовувати при виникненні певних подій у репозиторії, таких як коміт коду, відкриття пул-реквесту, створення тегу та інші.

Reusable Actions (перевикористання дій): можливість використовувати готові дії (actions) або створювати нові, які можна перевикористовувати в різних робочих процесах.

Matrix Builds (матричні збірки): визначають матрицю конфігурацій, для якої робочий процес буде виконуватися. Наприклад, для тестування коду на різних версіях мови програмування або операційних системах.

Secrets (секрети): GitHub Actions дозволяє безпечно зберігати та використовувати конфіденційні дані, такі як ключі API, паролі, токени доступу тощо.

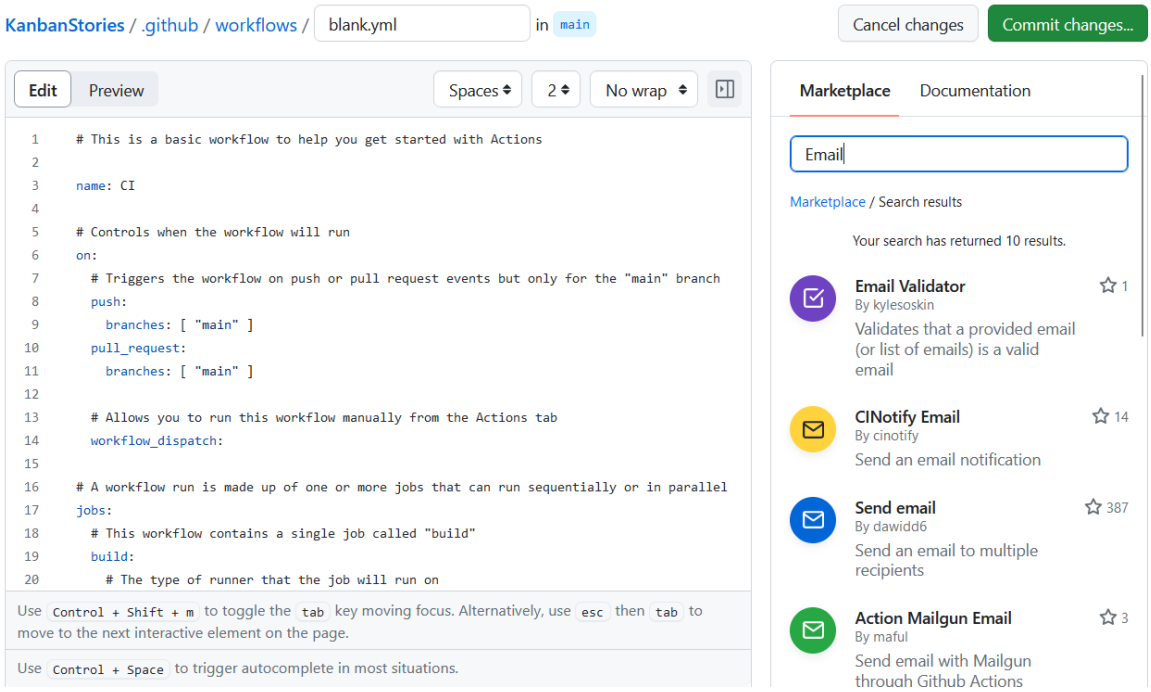


Рисунок 13 – Загальний вид GitHub Actions

Розробимо Pipeline що автоматично буде сповіщати користувачів про зміни до репозиторію за допомогою електронної пошти.

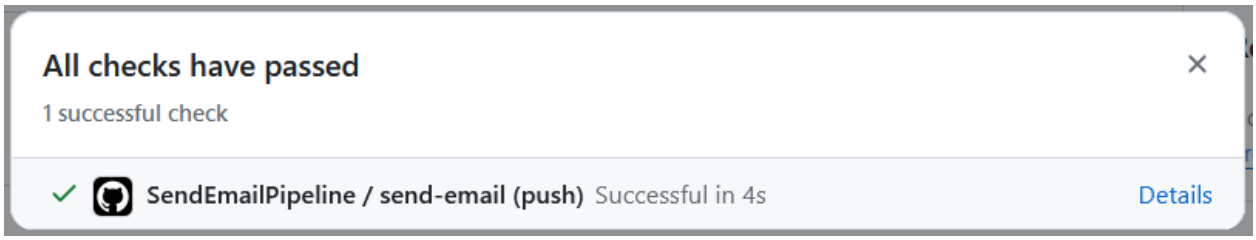


Рисунок 14 – Робота Pipeline у GitHub Actions

4 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА МОДЕЛІ ПРЕЦЕДЕНТУ USER STORY

4.1 Реалізація Prompt-моделі

У якості системи управління проектами використовується Trello. Trello — безкоштовна багатоплатформна система управління проектами, розроблена Trello Enterprise, дочірньою компанією Atlassian. Вона використовує парадигму менеджменту проектів, відому як Kanban. Проекти зображуються дошками, на яких розташовані переліки задач. Ця списки містять картки, якими зображуються задачі. Картки повинні переходити з попереднього списку до наступного (за допомогою перетягування), таким чином зображаючи рух якоїсь функції від ідеї до тестування. Будь-якому завданню може бути присвоєно виконавців або відповідальну особу. Користувачі та дошки можуть об'єднуватись в команди. Trello має обмежену підтримку тегів у вигляді шести кольорових міток. Завдання часто містять коментарі, додатки, кінцеву дату виконання та підзадачі. Форматуються картки розміткою Markdown. Інтерфейс програми працює в форматі drag-and-drop, всі дані оновлюються динамічно на фоні. Серед недоліків: система не працює в режимі офлайн, безплатні плани мають обмеження на розмір приєднаних файлів, неможливо редагувати коментарі.

Trello дозволяє налаштувати необмежену кількість статусів. Відтворимо статуси To do, In Progress, Code Review, Ready QA, QA Under Test, Ready UAT, Expected та Done.

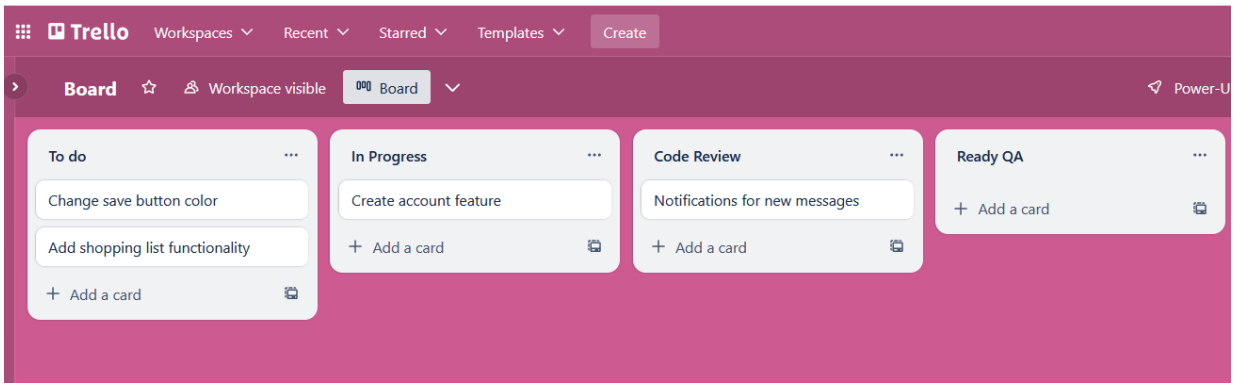


Рисунок 15 – Загальний вигляд Kanban-дошки у Trello

Картки фактично являють собою user story, що описані у форматі Connextra.

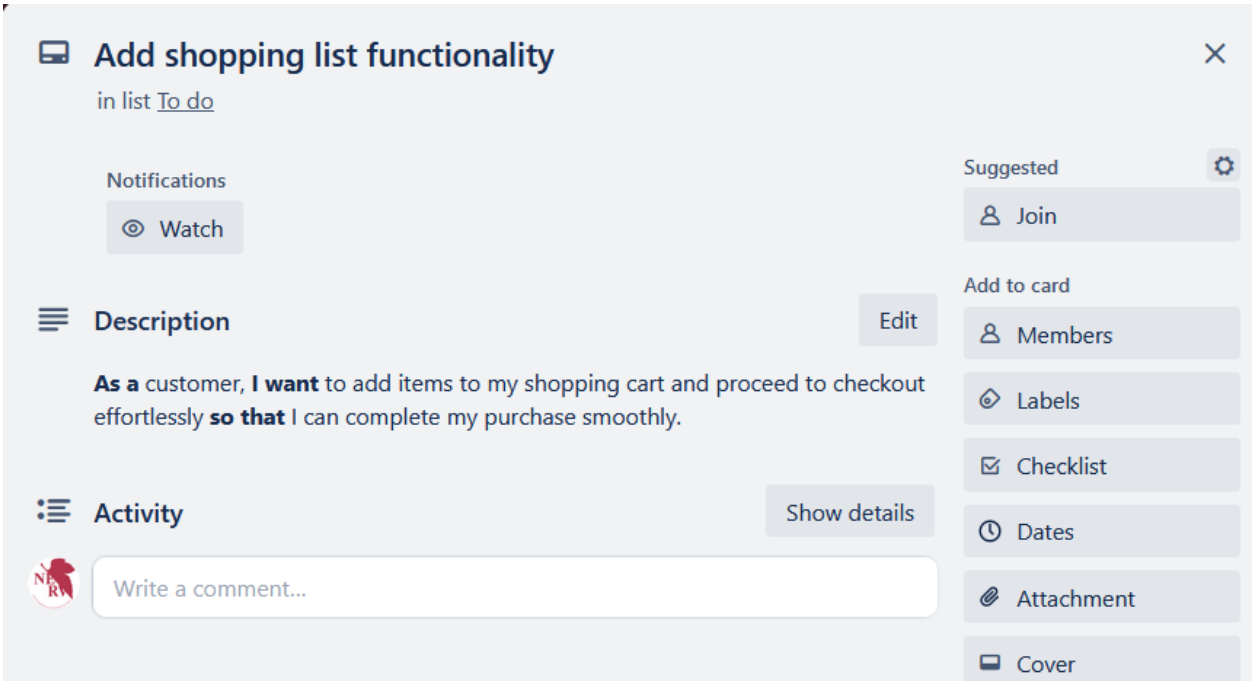


Рисунок 16 – Описана User Story у Trello

В якості технології для доступу до великої мовної моделі використовується Microsoft Copilot, вбудований у операційну систему Windows 11 у якості бокової панелі.

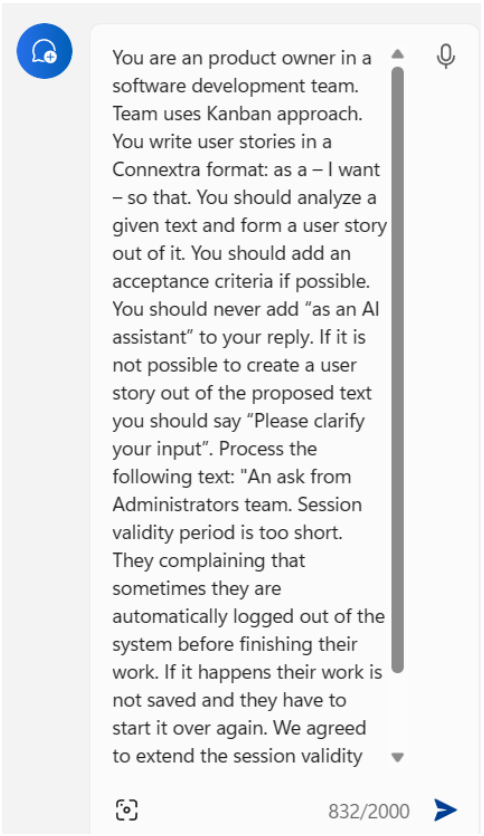


Рисунок 17 – Введення промпту у Microsoft Copilot у Windows 11

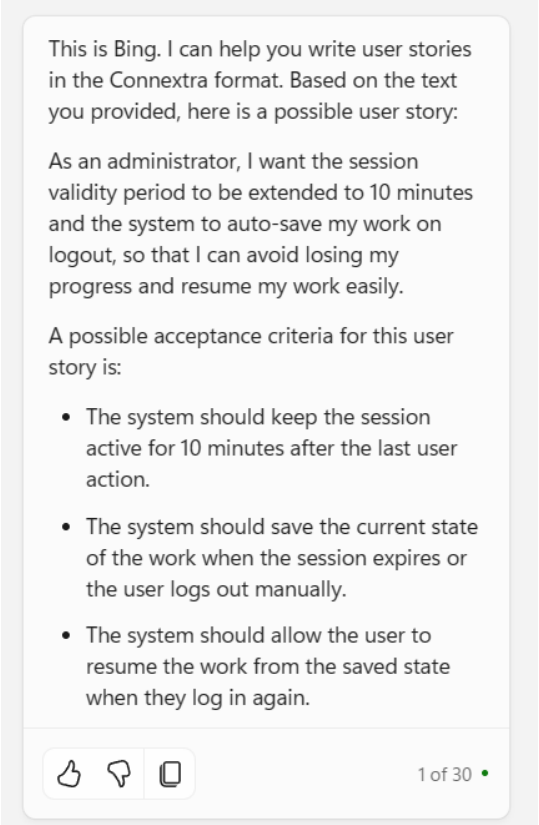


Рисунок 18 – Відповідь GTP-4 у Microsoft Copilot у Windows 11

4.2 Експериментальна перевірка Prompt-моделі

Ретроспективно було проаналізовано 350 User Story що були створені до впровадження технології використання великих мовних моделей при розробці програмного забезпечення інформаційних систем за методологією Kanban а також 120 user story після впровадження.

Оцінювались затрати часу на формування User Story з довільного опису а також кількість описаних acceptance criteria. Середні результати зведено до таблиць і діаграм.

Таблиця 8 – Затрати часу на описання User Story

	Min (хв)	Avg (хв)	Max (хв)
До	4	17,5	31
Після	2	4,5	7

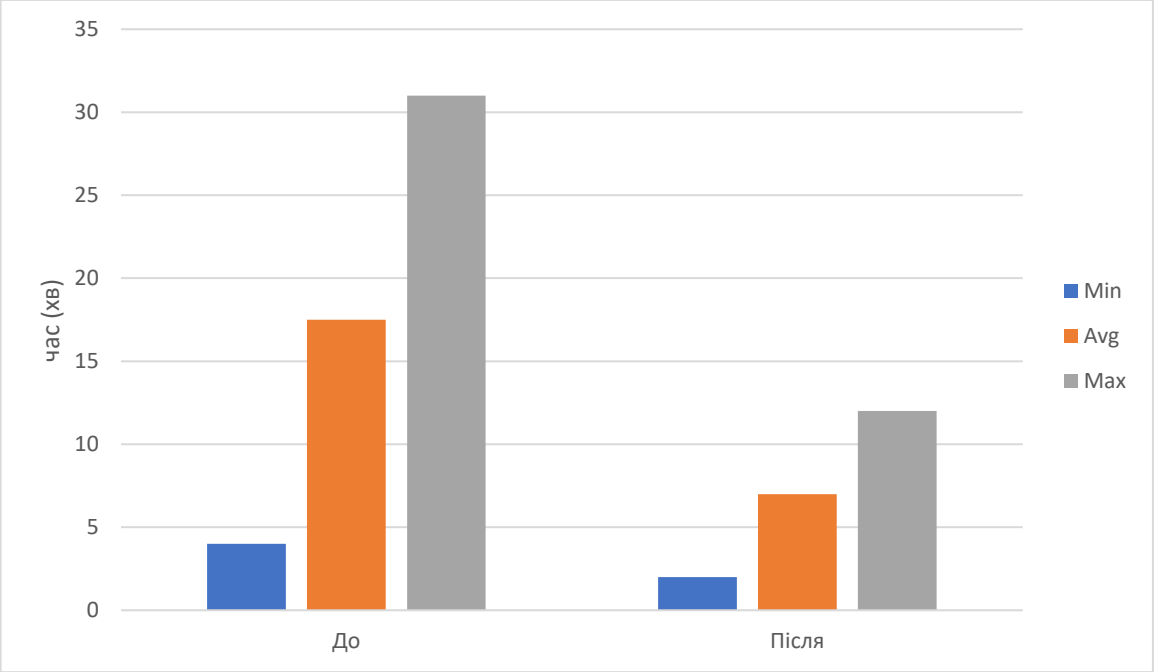


Рисунок 19 – Порівняння затрат часу на написання User Story

Як видно зі зведеної таблиці та діаграми, впровадження технології використання великих мовних моделей при розробці програмного забезпечення ІС за методологією Kanban дозволило знизити мінімальні затрати часу на написання User Story вдвічі. Проте, що більш важливо, максимальний час знизився в 4.4 рази.

Таблиця 9 – Кількість acceptance criteria у User Story

	Min	Avg	Max
До	0	2	4
Після	2	4	6

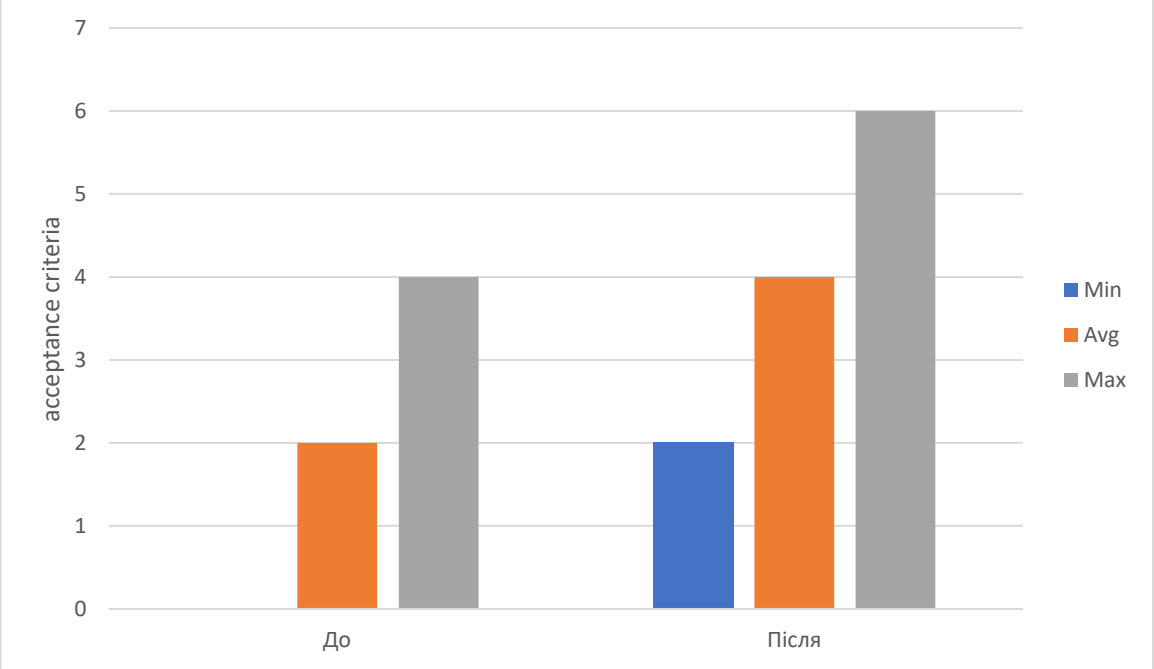


Рисунок 20 – Порівняння кількості acceptance criteria у User Story

Впровадження технології використання великих мовних моделей при розробці програмного забезпечення інформаційних систем за методологією Kanban збільшило максимальну та середню кількість описаних acceptance criteria у User Story на 2, проте більше взагалі не трапляються історії без описаних acceptance criteria.

ВИСНОВКИ

1. У кваліфікаційній роботі досліджено процес розробки ПЗ ІС з використанням методології Kanban. Проаналізовано особливості використання гнучких методологій для розробки ПЗ, проведено аналіз і порівняння великих мовних моделей, досліджено підходи до побудови запитів до великих мовних моделей та проаналізовано прецедентний підхід до вирішення задач з розробки ПЗ ІС.

2. Розроблено структуру прецеденту User Story та модель промпту LLM для написання User Story для побудови ПЗ ІС за методологією Kanban.

3. На базі методу створення User Story за допомогою великих мовних моделей а також моделі прецеденту User Story запропоновано технологію створення User Story за допомогою LLM.

4. В ході виконання кваліфікаційної роботи технологію створення User Story за допомогою LLM впроваджено за допомогою DevOps з використанням пайплайнів GitHub Actions.

5. Реалізовано Prompt-модель побудови User Story за допомогою GPT-4 у Microsoft Copilot.

6. Проведено експериментальну перевірку теоретичних результатів та зроблено аналіз результатів експерименту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки до передатестаційної практики для студентів усіх форм навчання спеціальності 122 – Комп’ютерні науки, освітньо-професійної програми "Інформаційні управляючі системи та технології" / Упоряд.: Чалий С.Ф., Євланов М. В., Чала О. В. - Харків: ХНУРЕ, 2021

2. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлювання. – Чинний від 22.06.2015. – Київ: ДП «УкрНДНЦ», 2016. – 31 с.

3. ДСТУ 8302:2015. Інформація та документація. Бібліографічні посилання. Загальні положення та правила складаня. – Чинний від 04.03.2016. – Київ: ДП «УкрНДНЦ», 2016. – 20 с.

4. Чала О.В. Поліпшення процесів системи управління якістю на основі коригувальних і попереджувальних дій. Вісник економіки транспорту і промисловості: Зб. наук. праць. Харків. 2006. Вип. 15–16. С. 118-121.

5. Chala O. Models of temporal dependencies for a probabilistic knowledge base. ECONTECHMOD: An International Quarterly Journal on Economics of Technology and Modelling Processes. 2018. Polska Akademia Nauk. Oddział w Lublinie PAN. T. 7. № 3. C. 53-58.

6. Chala O., Levykin V. Method of automated construction and expansion of the knowledge base of the business process management system. eureka: Physics and Engineering. 2018. T. 4. № 17, C. 29-35.

7. Чалий С.Ф., Чала О.В., Ревуцька Л.Є. Методичний підхід побудови процесно-орієнтованої системи управління якістю промислового підприємства. Problems of a systemic approach to the economy enterprises. 2008. Т. 4 № 12

8. Chala O. Logical-probabilistic representation of causal dependencies between events in business process management. Вісник НТУ «ХПІ». 2018. Т. 2 № 2 С. 40-44

9. Чала О.В. Формалізація неявних процедурних залежностей у знання-ємних бізнес-процесах. Наукові праці Вінницького національного технічного університету. 2016. № 4

10. Чалий С.Ф. Лещинський В.О. Метод можливісного оцінювання пояснення в системі штучного інтелекту. Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології. 2023. № 2 (10) С. 95-101

11. Chalyi S., Bogatov I. Технологія автоматизованої побудови моделі прототипу бізнес-процесу на основі попередньої обробки журналу подій. Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології. 2020. № 2 (4) С. 57-63

12. Чалий С.Ф., Прібильнова І.Б. Побудова ситуаційного представлення знань на основі аналізу логів. Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології. 2017. № 28. С. 70-73.

13. Чалий С.Ф., Лещинська І.О. Концептуальна ментальна модель пояснення в системі штучного інтелекту. Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології. 2023. № 1 (9) С. 70-75.

14. Corona E., Pani F.E. A Review of Lean-Kanban Approaches in the Software Development. 2013.

15. Reddy A. The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban. Addison-Wesley Professional, 2015.

16. Murino T., Naviglio G., Romano E. Optimal size of Kanban board in a single stage multi product system. 2010.

17. Ahmad, M.O., Markkula, J., Oivo, M. Kanban in software development: A systematic literature review. In Software Engineering and Advanced Applications (SEAA). 2013.

18. OpenAI Codex Blog. URL: https://openai.com/blog/openai-codex (дата звернення: 20.09.2023).

19. Introducing Code Llama, a state-of-the-art large language model for coding. URL: https://ai.meta.com/blog/code-llama-large-language-model-coding/ (дата звернення: 20.09.2023).

20. Introducing Amazon CodeWhisperer, the ML-powered coding companion. URL: https://aws.amazon.com/blogs/machine-learning/introducing-amazon-codewhisperer-the-ml-powered-coding-companion/ (дата звернення: 20.09.2023).

21. Prompt Engineering Guide. URL: https://www.promptingguide.ai/ (дата звернення: 12.12.2023).

22. LLM Models Comparison: GPT-4, Bard, LLaMA, Flan-UL2, BLOOM https://medium.com/@aiwizard/llm-models-comparison-gpt-4-bard-llama-flan-ul2-bloom-9ad7c0c56ba5 (дата звернення: 22.12.2023).

23. Моделі, методи та інформаційні технології управління наскрізними бізнес-процесами підприємства - Левикін Ігор Вікторович, дисертація, 2021

24. Agile, Scrum, Kanban, Waterfall. Як правильно обрати методологію для свого проєкту - https://dotli.com.ua/agile-scrum-kanban-waterfall-yak-pravilno-obrati-metodologiyu-dlya-svogo-proєktu (дата звернення: 15.12.2023).

25. Чалий С.Ф., Левикін І.В. Концепція двоконтурного управління множини наскрізних бізнес-процесів на основі прецедентного підходу. Поліграфічні, мультимедійні та web-технології: Матеріали III міжнар. наук.-практ. конф. Львів. 2018. С. 288-290.

26. Chalyi S., Levykin I., Information technology for the implementation of case-law management of end-to-end business processes. Computer and information systems and technologies: Fourth International Scientific and Technical Conference. Kharkiv: NURE. 2020. P. 54-55.

27. Чалий С.Ф, Левикін І.В. Метод побудови інтервальної моделі процесу вирішення задачі в складі прецеденту на основі аналізу журналу подій. Наукові праці ВНТУ. 2016. №4. С. 1-8.

28. Чалий С.Ф., Левикін І.В. Метод адаптивного процесного управління на основі прецедентного підходу. Наукоємні технології. 2016. № 4. С. 410-414.

29. Petrichenko A., Levykin I., Iuriev I. Improvement of the method of selecting it-services for the operated information. Eastern-European Journal of Enterprise Technologies. 2021. № 2/2 (110). P. 32-43. (Scopus).

30. Chalyi S., Levykin I., Biziuk A., Vovk A., Bogatov Ie. Development of the technology for changing the sequence of access to shared resources of business processes for process management support. Eastern-European Journal of Enterprise Technologies. 2020. № 2/3 (104). P. 22-29. (Scopus).

31. Chalyi S., Levykin I. Identification of the standby intervals in the business processes based on analysis of the sequence of events. Technology audit and production reserves. 2016. Vol. 5. № 2 (31). P. 71-76.

32. Chalyi S., Levykin I., Petrychenko A., Bogatov I. Causality-based model checking in business process management tasks. IEEE 9th International Conference on Dependable Systems, Services and Technologies DESSERT. 2018. P. 453-458.