The software product has been verified by conducting numerical experiments and analyzing the behavior of the mathematical model under different parameters. For this purpose, we calculated the values of the function at the test points and compared the results with the calculations in computer algebra systems. It is confirmed that the numerical solutions coincide with the analytical results.

The proposed software solution provides increased accuracy and performance of computations, allowing it to be used to analyze complex multifactorial physical systems.

Key words: *fractional differential equations, fractional derivative, mathematical modeling, catalytic porous medium, Fourier method, Laplace transform method, Mittag-Lefler function.*

Отримано: 2.02.2025

УДК 004.415.2+004.6
DOI: 10.32626/2308-5916.2025-27.40-57

Р. А. Крук,
Н. А. Жуковська, канд. техн. наук

Національний університет водного господарства
та природокористування, м. Рівне

ЗАСТОСУНОК-ПОСЕРЕДНИК ДЛЯ ТРАНСФОРМАЦІЇ ВИМОГ ДО ПЗ ATLASSIAN JIRA П'ЯТИКРОКОВИМ СЕМАНТИЧНИМ АНАЛІЗОМ ГЕНЕРАТИВНОЇ ВЕЛИКОЇ МОВНОЇ МОДЕЛІ

У статті представлено фреймворк для трансформації артефактів вимог у середовищі Agile-розробки в структуровану базу знань із використанням великих мовних моделей (LLM) та графових методів. Дослідження зосереджене на усуненні критичних обмежень сучасної інженерії вимог, зокрема фрагментованості інструментів, слабкої трасованості та недостатньої адаптивності до напівструктурованих даних. Запропонована система функціонує як проміжна ланка між Atlassian Jira та графовою базою знань, реалізуючи п'ятиетапну методику: кластеризацію вимог, прив'язку тестів, визначення залежностей, дедуплікацію та історичну реконструкцію. Основним компонентом виступає LLM, з якою система взаємодіє через JSON-орієнтований протокол, що включає інструкції з інтерпретації даних, очікувану структуру відповіді та дозволені дії (створення, оновлення, трасування тощо). Модульна система архітектури включає рівень користувацької взаємодії, агент-оркестратор, семантичне ядро, графову базу даних (Neo4j) та підсистему контролю доступу. Теоретична та експериментальна частини дослідження проведе-

ні за підтримки компанії SoftServe Inc. У межах експериментального запуску системи було оброблено понад 18 тисяч Jira-задач. Пілотне тестування засвідчило високу точність відповідей системи (90,4%) та потенціал значної економії часу для аналітиків і тестувальників. Було продемонстровано можливість реконструкції історичного стану вимог, виявлення повторів і порушень логіки залежностей, що особливо актуально для складних продуктів із багаторівневою структурою. Втім, система перебуває на стадії прототипу й має низку обмежень: зокрема, залежність від якості формулювання prompt-запитів (інструкцій), складність інтерпретації рішень LLM та потребу в ретельному налаштуванні безпечного доступу до даних. Отримані результати слід вважати попередніми; для перевірки масштабованості, стабільності та практичної доцільності підходу необхідне подальше тестування у різних проєктах, доменах та з більшим обсягом запитів.

Ключові слова: *інженерія вимог, Agile, LLM, аналіз вимог, Jira, семантичний аналіз.*

Вступ. Інженерія вимог до програмного забезпечення (Software Requirements Engineering, SRE) – це дисципліна, орієнтована на побудову знань, що трансформує потреби зацікавлених сторін у точні специфікації системи. Упродовж десятиліть дослідники прагнули моделювати вимоги у формалізованому алгоритмічному та математичному вигляді з метою підвищення строгості та автоматизації аналізу. Ранні підходи до моделювання вимог впроваджували формальні методи та знаннєво-орієнтовані системи для представлення вимог за допомогою логіки та правил.

Паралельно з цим сучасні тенденції у розробці програмного забезпечення, зокрема методології Agile [1], докорінно змінили спосіб збору та управління вимогами. Agile-команди надають перевагу легковагим, напівструктурованим артефактам (наприклад, користувацьким історіям та критеріям прийнятності) і постійній еволюції вимог, що створює нові виклики для традиційних підходів до моделювання [2-3]. Останнім часом методи штучного інтелекту (ШІ) – особливо обробка природної мови (natural language processing, NLP) [4], великі мовні моделі (LLM) та генеративний ШІ – демонструють потенціал у роботі з неструктурованими та напівструктурованими даними, властивими вимогам.

Метою цього дослідження є розробка та оцінка фреймворку, що використовує великі мовні моделі та графові методи для трансформації Agile-вимог до програмного забезпечення в структуровану базу знань, доступну для запитів. Фреймворк спрямований на підтримку трасування, узгодженості та семантичного розуміння змінюваних вимог у реальних Agile-проєктах.

Пов'язані роботи. У дослідженні пов'язаних наукових праць було проаналізовано сучасний стан алгоритмічних і математичних підходів до інженерії вимог до програмного забезпечення (SRE), з особливим акцентом на середовище Agile-розробки. Розглядалися традиційні формальні методи та оптимізаційні техніки [5], а також системи, що базуються на онтологіях [6], із висвітленням їхнього внеску та обмежень у контексті динамічних, напівструктурованих Agile-робочих процесів.

Новітні досягнення у сфері штучного інтелекту, зокрема рекурентні нейронні мережі [7], а особливо генеративні великі мовні моделі (LLM), відкрили нові можливості на всіх етапах життєвого циклу вимог. Наукова література демонструє зростаючий інтерес до використання LLM [8-9] для таких завдань, як автоматизоване виявлення вимог [10], їх класифікація, встановлення трасувальних зв'язків [11], а також трансформація в похідні артефакти (наприклад, діаграми UML, тестові сценарії). ШІ посилює виконання цих завдань, виявляючи закономірності у неструктурованих і напівструктурованих даних, зокрема користувацьких історіях, і забезпечує інтелектуальну підтримку у вдосконаленні та валідації вимог.

Попри цей прогрес залишаються значні прогалини. Більшість поточних підходів є фрагментованими: вони зосереджуються на окремих завданнях, замість того, щоб запропонувати інтегрований наскрізний фреймворк. Також їм притаманні такі проблеми, як відсутність інтерпретованості, обмежена здатність до адаптації в різних доменах і складність аудиту результатів, створених ШІ. Крім того, методи оцінювання є непослідовними, з недостатньою увагою до систем з участю людини або механізмів зворотного зв'язку протягом усього життєвого циклу.

Ця прогалина відкриває простір для створення нового фреймворку, який поєднує надійність представлень, орієнтованих на знання, а не лише накопичення даних, із гнучкістю та виразністю генеративного ШІ. Завдяки закріпленню результатів LLM у структурованих графах знань, а також інтеграції формальних методів для відстеження залежностей, контролю версій і семантичної кластеризації, така система може надати цілісну підтримку для інженерії вимог в Agile-середовищі – забезпечуючи міст між артефактами природної мови та формальним проєктуванням систем.

Перші етапи дослідження.

1. Опис проблеми та контекстний аналіз. На початковому етапі дослідження було проведено всебічний огляд літератури з метою виявлення стійких і нових викликів у сфері інженерії вимог до програмного забезпечення (SRE) в умовах Agile-розробки [12]. Огляд охопив понад два десятиліття академічних напрацювань з усього світу й дозволив виокремити одинадцять критичних вузьких місць, серед яких: слабкі практики документування, непослідовне управління ви-

могами та нестача інструментів для забезпечення трасованості. Ці проблеми виявилися особливо впливовими у великомасштабних Agile-проєктах, де відсутність структурованої документації та стрімкі зміни контексту підривають довгостроковий успіх ініціатив.

З метою контекстуалізації цих висновків на другому етапі дослідження було проведено емпіричне вивчення ситуації в межах української IT-спільноти [13]. Було здійснено опитування фахівців, що працюють у різних ролях в Agile-командах: менеджерів вимог, розробників, тестувальників, продакт-оунерів (власників продукту), технічних лідерів та архітекторів. Дослідження виявило, що документація залишається однією з головних проблем, особливо в проєктах на етапах MVP та попереднього вивчення. Також було підтверджено пряму кореляцію між проблемами в роботі з вимогами та нездатністю виконати зобов'язання перед клієнтом. Важливо, що практикуючі респонденти повідомили про регулярні втрати часу через погано структуровані або фрагментовані вимоги, а також назвали Atlassian Jira та Confluence найпоширенішими інструментами. Ці висновки стали основою для подальшої розробки формальної моделі, покликаної усунути вузькі місця Agile-проєктів у сфері інженерії вимог.

2. Аналіз критичних точок потоків даних вимог до ПЗ в методології Agile. Третій етап дослідження був зосереджений на формалізації потоку даних про вимоги до програмного забезпечення в Agile-проєктах з метою виявлення вузьких місць, які перешкоджають узгодженості та трасованості протягом усього життєвого циклу розробки [14-16]. Автори запропонували концептуальну модель потоку даних, яка відображає еволюцію вимог: від початкового виявлення, через уточнення та декомпозицію, до фінальної реалізації та валідації. Модель була побудована з використанням UML-діаграм контексту, потоків даних, активностей та «stock-flow»-структур (рис. 1), доповнена нотаціями BPMN і прив'язана до типових ролей та інструментів в Agile-середовищі.

Ключовим внеском цієї роботи стало запровадження структурованого підходу до виявлення так званих «точок розриву» (breakpoints) у потоці вимог – критичних моментів, у яких найбільш імовірні втрата даних, неоднозначність або помилки в комунікації. У дослідженні було підкреслено важливість синхронізації неформальної комунікації з формалізованою документацією вимог в інструментах на кшталт Jira та Confluence. Також було запропоновано необхідність проміжного програмного шару (middleware), який автоматизує переходи між неструктурованим введенням та структурованими базами знань.

Ця робота заклала теоретичну основу для створення інтелектуальних систем, здатних у режимі реального часу здійснювати трасування, кластеризацію та аудит вимог, що еволюціонують з плином часу, у контексті Agile-розробки.

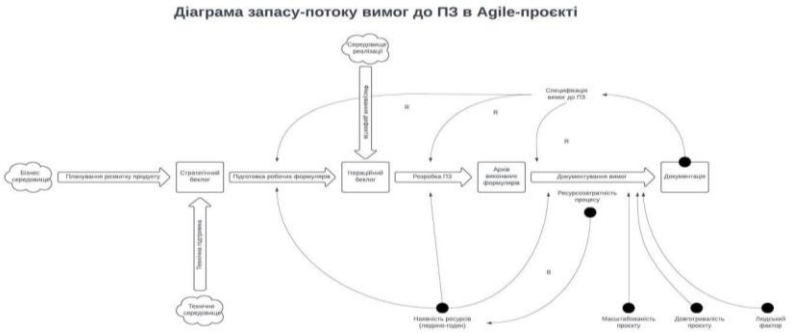

Рис. 1. *Діаграма запасів-потоків вимог до ПЗ в типовій системі управління вимогами до ПЗ методології Agile*

Методологія дослідження. Дослідження проводилося в рамках Agile-проєкту компанії SoftServe Inc., у якому Jira використовувалася як основна система для управління вимогами. Метою дослідження було створення архітектури знаннєво-орієнтованої системи підтримки вимог, що використовує генеративну велику мовну модель (LLM) як семантичного інтерпретатора вимог. З цією метою було реалізовано проміжну програму, яка функціонувала між LLM та Jira API [17] і вивантажувала задачі та підзадачі у напівструктурованому форматі YAML/JSON. Детальна візуалізація компонентів експериментального застосунку наведена на рис. 2. На її основі була розроблена перша версія архітектури системи, подана на рис. 3.

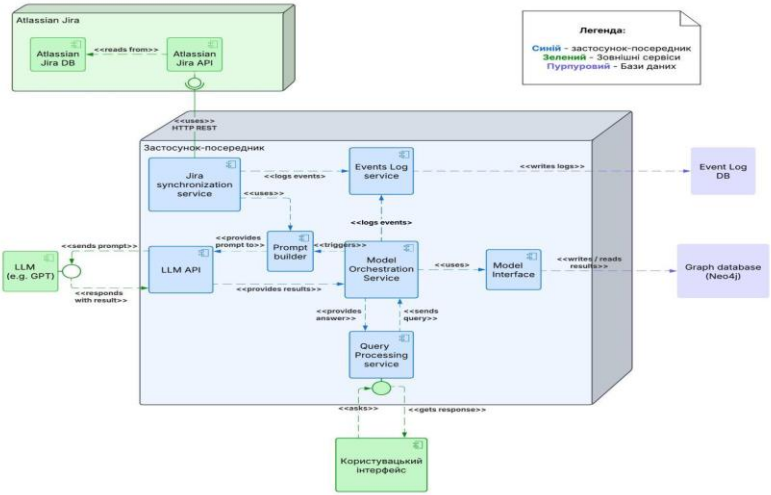

Рис 2. *Діаграма компонентів експериментального застосунку управління вимогами з використанням запропонованого п'ятикрокового підходу*

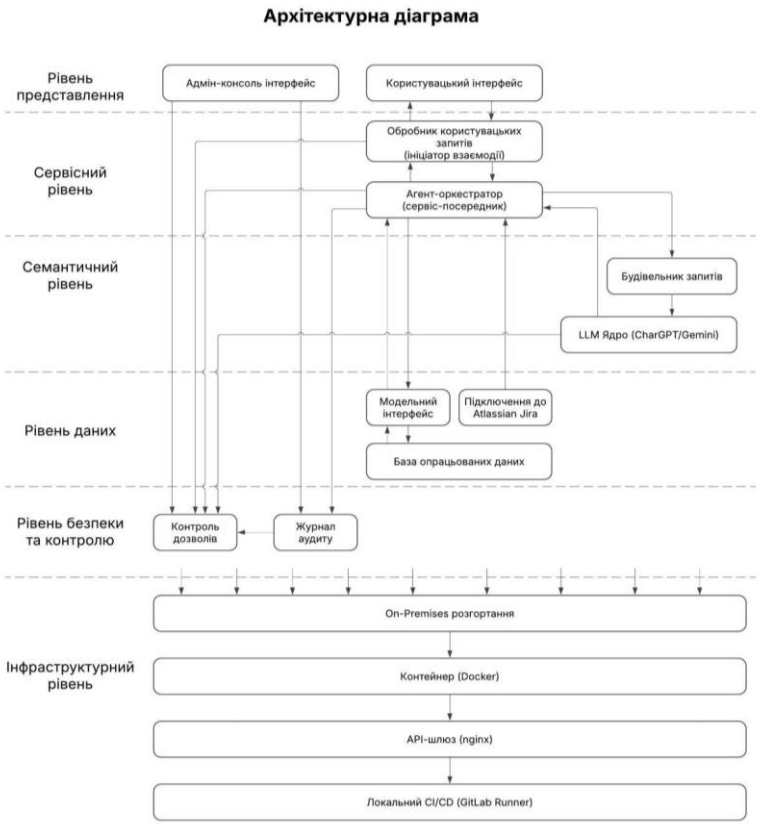

Рис. 3. *Архітектурна діаграма експериментального застосунку управління вимогами з використанням запропонованого п'ятикрокового підходу*

1. Огляд архітектури. Архітектура побудована за принципом розмежування функціональних рівнів і забезпечує наскрізну обробку, трасування та інтерпретацію вимог за допомогою великих мовних моделей (LLM), графової бази даних та сервісного посередника.

На рівні представлення функціонують два основні інтерфейси: адміністративна консоль та користувацький інтерфейс. Через них здійснюється налаштування системи та взаємодія кінцевих користувачів, таких як аналітиків, тестувальників і розробників. Запити користувача обробляються через спеціалізований модуль – обробник користувацьких запитів, який визначає тип наміру та передає запит до агента-оркестратора.

На сервісному рівні функціонує агент-оркестратор, який виступає центральним координатором запитів, маршрутів даних та логіки виконання. Він забезпечує зв'язок між інтерфейсами, інтелектуаль-

ним модулем LLM і підсистемами зберігання та аналізу даних. Оркестратор керує будівельником запитів, який здійснює структуровану генерацію prompt'ів на основі вхідних даних, що подаються до LLM-ядра (наприклад, ChatGPT або Gemini), інтегрованого через API відповідного SaaS-рішення.

Семантичний рівень реалізує смислову обробку вимог. Тут функціонує LLM-ядро, що виконує класифікацію, трасування, семантичне групування, порівняння, дедуплікацію та рефакторинг вимог. Крім того, ядро отримує вхідні дані з підключення до Atlassian Jira, де витягуються напівструктуровані артефакти (epic, story, task, test case, spike) у форматі YAML/JSON. Слід зазначити, що з точки зору системи, що розробляється даний компонент є так званим blackbox-рішенням, оскільки вся його логіка відбувається на стороні відповідної SaaS-системи, з якою інтегрується система, що розробляється.

Рівень даних представлений базою оброблених даних, яка зберігає історично накопичені вимоги, результати обробки, залежності та метаінформацію. Усі функціональні одиниці моделюються у вигляді вузлів графа, а залежності – як орієнтовані ребра. Графова модель дає змогу здійснювати історичні запити, аналізувати взаємозв'язки між одиницями, а також підтримувати механізми відстеження змін і динамічного оновлення знань.

Рівень безпеки та контролю забезпечується двома критичними підсистемами: контролем дозволів і журналом аудиту. Вони реалізують політики доступу, автентифікації до Jira API, а також фіксують усі транзакції в системі для подальшої перевірки, що є особливо важливим для галузей із підвищеними вимогами до валідації.

На інфраструктурному рівні архітектура на даному етапі дослідження передбачає лише on-premises розгортання (тобто стособом розгортання ПЗ безпосередньо на локальній інфраструктурі), яке реалізується у контейнерах Docker і обслуговується через API-шлюз (nginx). Підтримується локальний CI/CD-процес на основі GitLab Runner, що дає змогу оперативно оновлювати компоненти системи без порушення її роботи.

2. Протокол обміну даними системи на LLM-компонента. У межах запропонованої архітектури ключову роль у взаємодії між системою та великою мовною моделлю (LLM) відіграє структурований обмін даними у форматі JSON. Цей механізм забезпечує формалізовану комунікацію між агентом-оркестратором та LLM-ядерним модулем, яка є критичною для коректного семантичного аналізу вимог та управління графом знань.

Система формує спеціально сконструйовані prompt-бандли, які передаються до LLM у вигляді JSON-документів. Ці бандли містять:

1. Інструкції щодо інтерпретації вхідних даних (наприклад, опис задач Jira, параметри тестів, критерії прийнятності).
2. Контекст запиту (тип взаємодії: класифікація, трасування, дедуплікація, історичний аналіз тощо).
3. Структуровану інформацію про функціональні одиниці (наприклад, дерево епіків, метадані про задачі, існуючі вузли графа).
4. Очікуваний формат відповіді, з визначеними дозволеними значеннями, які можуть бути інтерпретовані оркестратором.

Ключовим є те, що кожен запит супроводжується prompt-інструкцією, яка пояснює LLM правила прочитання JSON-структури, інструкції щодо семантичного аналізу та структуру відповіді й допустимі типи дій, які система має виконати на основі результату.

Приклад відповіді LLM для оновлення одного вже існуючого функціонального вузла:

```
{
 "action": "update_node",
 "target_node_id": "FUNC_021",
 "new_requirements": [
 "The system shall support two-factor authentication for admin users.",
 "Timeout configuration must be adjustable per environment."
 ],
 "impact_assessment": {
 "affected_nodes": ["FUNC_004", "FUNC_005"],
 "dependency_type": "data_reuse"
 }
}
```

Або для створення нового функціонального вузла:

```
{
 "action": "append_node",
 "parent_reference": "FUNC_128",
 "functional_unit": {
 "id": "",
 "title": "User onboarding workflow",
 "requirements": [...],
 "linked_tests": [...],
 "dependencies": [...]
 }
}
```

Отриманий JSON обробляється агентом-оркестратором, який інтерпретує значення поля «action» та виконує відповідні дії, передає

дані до бази оброблених даних та/або оновлює графову структуру у відповідному модулі та ініціює рефлексивну перевірку або аудит у випадках, де це потрібно (наприклад, якщо змінюється критичний вузол або виникає циклічна залежність).

Завдяки такому протоколу система реалізує двосторонню, семантично керовану інтеграцію з LLM, де prompt є не лише запитом, а й контрактом між LLM і системною логікою. Це дозволяє гарантувати не лише узгодженість дій, а й можливість масштабування, тестування, трасування та логування кожної транзакції взаємодії.

У майбутньому така структура також створює основу для адаптивних стратегій prompt-інженерії та автоматичного навчання, в якому сама система збиратиме найуспішніші шаблони prompt'ів і результатів, формуючи основу для автономного уточнення логіки обробки.

3. Кроки опрацювання даних. Методологія обробки вимог складалася з п'яти ітеративних кроків, заснованих на найкращих практиках і критичних аспектах, підкреслених у [18]:

1. *Побудова ієрархічної графової моделі вимог.*

На першому етапі велика мовна модель (LLM), за допомогою інструктивного prompt engineering, виконує кластеризацію задач, що належать до одного епіку, формуючи деревоподібну модель функціональної ієрархії системи. Для цього використовується модифікована версія агломеративного кластеризаційного алгоритму з метрикою семантичної подібності, що обчислюється через векторні подання текстів задач (Sentence-BERT). Отримана структура подається як орієнтований граф $G = (V, E)$, де вершини V – це функціональні одиниці, а ребра E позначають відношення підпорядкування або композиції.

2. *Прив'язка функціональних вимог.*

На другому етапі, з використанням класифікації природною мовою, LLM автоматично ідентифікує та прив'язує тестові сценарії (зокрема критерії прийнятності, QA-чеклісти та BDD-сценарії) до відповідних функціональних одиниць графа. Було реалізовано функцію зіставлення на основі косинусної подібності між векторами вбудовування вимог і описів задач. Функція відображення f: $T \rightarrow V$, де T – множина тестових сценаріїв, забезпечує найкращу семантичну відповідність між тестом і функціональною одиницею.

3. *Визначення функціональних залежностей.*

Далі моделюються горизонтальні залежності між функціональними одиницями на основі спільних або пов'язаних сутностей, змінних чи API-ендпойнтів. Застосовується підхід зваженого графа залежностей $D = (V, Ed, w)$, де Ed – це ребра, що позначають міжфункціональні зв'язки, а вага w кожного ребра відображає кількість спільних залежних

об’єктів. Було проведено семантичну нормалізацію назв змінних та API з використанням лексичної онтології предметної області.

4. Дедуплікація та трасування вимог.

Четвертий крок зосереджений на виявленні дубльованих або еквівалентних вимог, розпорошених серед різних функціональних одиниць. LLM виконує порівняння вимог за допомогою нечіткого зіставлення рядків (алгоритм Левенштейна), а також семантичних метрик. Для кожної пари подібних вимог r_i, $r_j \in R_r$ обчислюється коефіцієнт подібності $\sigma(r_i, r_j)$, і якщо він перевищує заданий поріг, вони вважаються дублями, з визначенням напряму походження шляхом аналізу історії змін у Jira. У такий спосіб формується мережа трасованості вимог, що дозволяє автоматично підтримувати їхню узгодженість.

5. Аудит та історична реконструкція.

Останній етап передбачає створення історичних знімків стану знань про систему на будь-який момент часу. Розроблено механізм автоматичної версіфікації графа функціональних одиниць, пов’язаних вимог і встановлених залежностей. Усі дані зберігаються в графовій базі даних (Neo4j), де кожна зміна має часову мітку. Завдяки цьому можливо здійснювати запити на кшталт: «Якою була структура вимог 03.01.2025?» або «Коли була додана вимога щодо MFA?» – просто здійснюючи обхід історичних шарів графа.

Результати експерименту. Під час початкового розгортання проміжна програма (middleware) обробила 18 199 Jira-квитків (типи: story, task, test case, spike, epic), зібраних за три Agile-квартали реального проєкту розробки (Q3 2024 – Q1 2025). Весь процес витягання даних та побудови графа знань зайняв 313,3 години (≈13 діб) в автономному режимі. Основна частина часу припала на семантичну обробку, керовану LLM, із середнім часом обробки одного квитка від 43 до 84 секунд.

Після завершення формування бази знань зацікавлені сторони проєкту – зокрема аналітики, розробники та QA-інженери – взаємодіяли з системою через інтерфейс запитів природною мовою (консоль). LLM автоматично визначала цільову функціональну одиницю та тип наміру в межах чотирьох підтримуваних категорій запитів:

1) візуалізація структурної ієрархії;
2) отримання поточних вимог;
3) витягування залежностей даних та логіки;
4) реконструкція історичних знімків.

У рамках пілотної оцінки, що включала 355 реальних користувацьких запитів, система надала точні та повні відповіді на 321 із них, досягнувши рівня коректності 90,4%, що суттєво перевищило очікування авторів (попередньо прийнятним вважався ≈70%). Користувачі

повідомляли про економію часу від 10 хвилин до 1 години на один запит порівняно з традиційним ручним переглядом беклогу. Попри обнадійливі результати, вони залишаються попередніми, оскільки базуються лише на одному проєкті з обмеженим обсягом даних і запитів. Потрібне подальше довгострокове тестування для перевірки масштабованості та надійності системи в ширшому контексті.

Побудований граф включав понад 18 000 вузлів та приблизно 45 000 орієнтованих ребер, що репрезентують ієрархічні, залежні та тестові зв'язки. Середній ступінь вузла становив 2,9. Граф мав модульну структуру – 1 200 слабко зв'язаних компонентів, які відповідали функціональним доменам, побудованим на основі епіків. Під час аналізу сильно зв'язаних компонентів (SCC) було виявлено кілька циклічних залежностей, що вказують на потенційні помилки проєктування в початковому беклозі (переліку завдань або обсязі робіт).

Візуалізацію випадково вибраного підграфа наведено на рис. 3-5. Вузлами візуалізованого підграфа є функціональні одиниці розроблюваної системи. Кожна з них містить набір текстових вимог, що до неї належать, але також можуть бути пов'язані або залежні від інших функціональних одиниць.

Чорні зв'язки позначають вертикальну ієрархічну структуру, червоні – горизонтальні залежності повторного використання даних або вимог між одиницями.

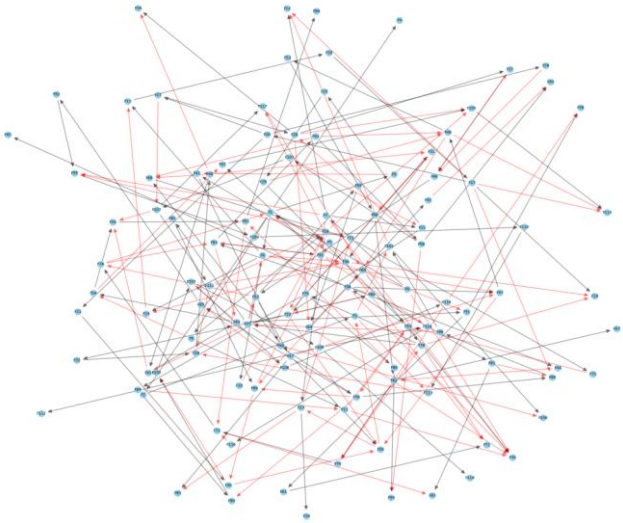

Рис. 4. *Візуалізація випадково обраного підграфа функціональних одиниць вимог до ПЗ з вертикальними ієрархічними (чорними) та горизонтальними зв'язками залежності (червоними)*

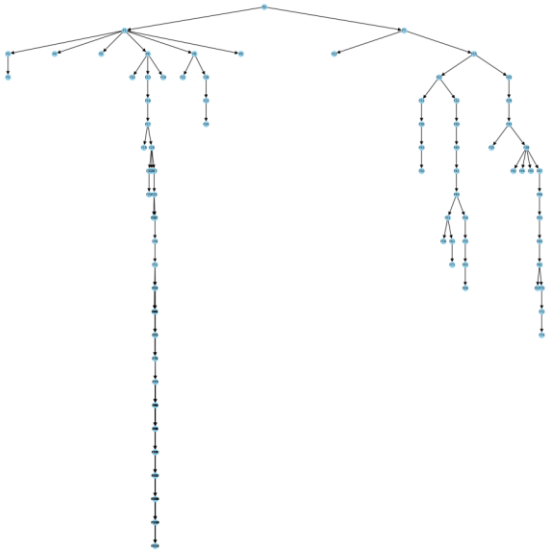

Рис. 5. *Візуалізація випадково обраного підграфа функціональних одиниць вимог до ПЗ з виключно ієрархічними зв'язками, що підкреслюють його деревовидну структуру*

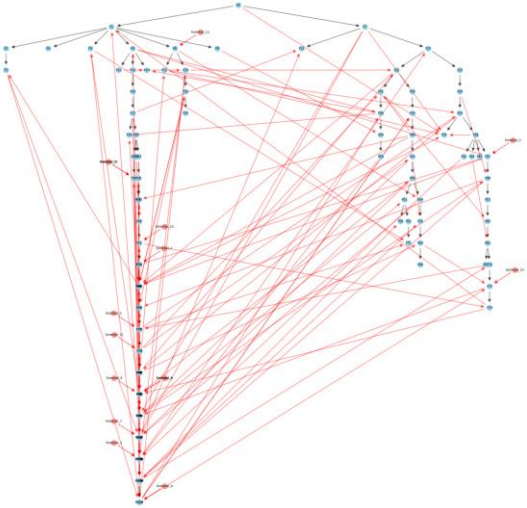

Рис. 6. *Візуалізація випадково обраного підграфа функціональних одиниць вимог до ПЗ з вертикальними ієрархічними (чорними) та горизонтальними зв'язками залежності (червоними) у розрізі, що підкреслюють його деревовидну структуру*

Обговорення.

1. Новизна дослідження та порівняння з подібними роботами. Детальний огляд пов'язаних робіт, наведений у розділі «пов'язані роботи», засвідчив, що наявні рішення зазвичай зосереджуються лише на одному конкретному завданні (наприклад, тільки на класифікації або лише на виявленні дублікатів) і часто не мають єдиного наскрізного механізму для роботи з динамічно змінюваними вимогами в Agile-проєктах. Крім того, багато існуючих підходів не інтегрують генеративні методи, тобто не використовують великі мовні моделі (LLM) або подібні інструменти для поєднання аналізу вимог та їх структуризації в єдину базу знань. Методологія, розглянута в цьому дослідженні (п'ятиетапна схема обробки й зберігання вимог через проміжний застосунок), поєднує відомі алгоритмічні та лінгвістичні підходи з великими мовними моделями та графовими базами даних для забезпечення безперервного оновлення знань.

Отже, новизна підходу полягає в поєднанні LLM та графових моделей для формування «живої» системи управління вимогами, яка безперервно оновлюється протягом ітерацій Agile, зокрема:

1) автоматичне виявлення та трасування залежностей не лише у вертикальному (функціональні одиниці), але й у горизонтальному вимірах (міжодиничні зв'язки, дублікати вимог, спільні дані);
2) включення механізму історичної реконструкції, що дозволяє відстежувати вимоги, дані та залежності повторного використання на різних часових зрізах у минулому;
3) фокус на інженерії вимог (SRE), на відміну від класичних систем на зразок Jira, які орієнтовані насамперед на управління проєктом.

2. Переваги запропонованого підходу. Завдяки своїй ітеративній природі (п'ять запропонованих кроків повторюються при кожному оновленні вимог) цей підхід природно інтегрується з Agile-робочими процесами.

Алгоритми на кшталт fuzzy matching і косинусної подібності дозволяють швидко виявляти повторювані або надмірні вимоги, тоді як зв'язність між функціональними одиницями підтримується в єдиній графовій базі даних.

Усі зв'язки між функціональними одиницями зберігаються в орієнтованому графі. Це дозволяє командам розуміти контекст змін у вимогах та швидко оцінювати вплив модифікації будь-якої окремої одиниці розроблюваної системи.

Завдяки використанню великих мовних моделей на зразок ChatGPT (LLM) (але за потреби може бути застосована й альтернатива) стає можливим не лише автоматизувати рутинні завдання, а й підтримувати рефакторинг (покращення) вимог. Такі можливості обме-

жені або повністю відсутні в традиційних системах на основі правил чи низькорівневих методах обробки природної мови (NLP).

3. Обмеження запропонованого підходу. Для забезпечення ефективного навчання та точних рекомендацій велика мовна модель (LLM) повинна мати доступ до різноманітних і репрезентативних даних про вимоги та тести. У реальних проєктах часто існують обмеження щодо доступності та якості даних (вимоги можуть бути частково задокументовані або конфіденційними).

Крім того, проміжна програма повинна бути правильно налаштована – з доступом до Jira API, захищеною автентифікацією та забезпеченням безпеки даних. Також необхідний механізм синхронного оновлення графової бази даних, що може ускладнити DevOps-процеси.

Генеративні моделі часто розглядаються як «чорні скриньки» у контексті прозорості прийняття рішень. Це може бути проблемою в галузях із жорсткими вимогами до валідації (наприклад, у фінансовій сфері чи охороні здоров'я). У цьому дослідженні не розглядається юридична сторона запропонованого рішення, зокрема питання NDA тощо.

Точність відповідей LLM суттєво залежить від якості сформульованих prompt'ів. Це вимагає досвіду та постійної адаптації до специфіки проєкту. Чітка структуризація вхідних даних стає критичним фактором успіху.

Нарешті, орієнтовна вартість повного циклу обробки із використанням GPT-4o становила близько ~$1031, тоді як альтернативна реалізація на базі Gemini 1.5 Pro дала подібні результати зі зменшенням витрат на 28% (згідно з розрахунками продуктивності). Розцінки базувалися на публічно доступних тарифах станом на квітень 2025 року. Варто зазначити, що основна частина витрат LLM припадає на початкову фазу налаштування, коли модель повинна опрацювати великий обсяг історичних даних про вимоги для генерації первинного графа знань. У режимі безперервної роботи очікується суттєве зменшення приросту витрат, оскільки обробки потребуватимуть лише нові або змінені квитки.

Водночас поточний етап дослідження не передбачає оцінювання економічної доцільності запропонованої системи, оскільки для цього потрібні розширене тестування та ґрунтовна оцінка економії часу та ергономічної вигоди для реальних користувачів у середовищі Agile-команд.

Висновки. У цьому дослідженні було запропоновано п'ятиетапний фреймворк для трансформації артефактів вимог в Agile-проєктах у структуровану, придатну до запитів базу знань із використанням великих мовних моделей (LLM) та графових моделей. Запропоновану проміжну систему було протестовано на понад 18 000 Jira-квитках, у результаті чого побудовано граф ієрархічних та залежних зв'язків і

досягнуто понад 90% точності у відповіді на користувацькі запити під час пілотного тестування. Результати підтверджують, що LLM можуть ефективно підтримувати розуміння вимог, трасування та відстеження змін у динамічному середовищі Agile.

Фреймворк демонструє технічну здійсненність та потенціал для економії часу для таких ролей, як аналітики, розробники та тестувальники. Попри те, що початкове налаштування системи вимагає значних обчислювальних ресурсів через обробку історичних даних, у звичайному режимі експлуатації приріст витрат очікувано знижується. Порівняльний аналіз свідчить, що бюджетніші LLM, такі як Gemini 1.5 Pro, можуть істотно зменшити витрати на розгортання.

У подальших дослідженнях передбачено розширення системи на інші проєкти та предметні області, інтеграцію адаптивних стратегій формування prompt'ів, а також збір довгострокових даних про зручність користування. Для повної оцінки економічного та ергономічного впливу управління вимогами за допомогою ШІ в реальних умовах Agile буде потрібно всеосяжне дослідження співвідношення витрат і вигод.

Апробація. Апробація результатів дослідження здійснювалася шляхом участі у міжнародних науково-практичних конференціях, а також через публікацію в рецензованих фахових виданнях. Зокрема:

1. Під час доповіді на VI Міжнародній науково-практичній конференції «Моделювання, керування та інформаційні технології» (9-11 листопада 2023 року, Національний університет водного господарства та природокористування, м. Рівне) [13].
2. У науковій статті «Оглядове дослідження проблем та викликів інженерії вимог в Agile-проєктах», опублікованій у журналі категорії Б «Вісник Національного університету водного господарства та природокористування» (розділ «Технічні науки», 2024 рік) [12].
3. Під час доповіді на VII Міжнародній науково-практичній конференції «Моделювання, керування та інформаційні технології» (7-9 листопада 2024 року, м. Рівне) [14].
4. Під час доповіді на XII Міжнародній науково-практичній конференції (10-12 грудня 2024 року, Національний університет «Запорізька політехніка») [15].
5. У науковій статті «Analysis of bottleneck points based on the software requirements data flow model in Agile projects», прийнятій до друку в журналі категорії А «Mathematical Modelling and Computing» (прийнято до публікації у 2025 році) [16].
6. Під час доповіді на Міжнародній науково-практичній конференції молодих науковців, аспірантів і здобувачів вищої освіти «Проблеми та перспективи розвитку сучасної науки» (8-9 травня 2025 року, м. Рівне) [19].

Додатково, основні положення роботи пройшли апробацію через прийняття тез доповіді на 15th International Conference of Advanced Computer Information Technologies, що запланована на вересень 2025 року [20]. Частина матеріалу цієї статті попередньо представлена у зазначених тезах доповіді.

Подяка. Автори щиро дякують компанії SoftServe за надане професійне середовище, доступ до інфраструктури реальних Agile-проєктів, а також за необхідні інструменти та підтримку, які зробили можливим проведення цього дослідження. Можливість здійснювати дослідження в умовах активного середовища розробки значною мірою сприяла практичній релевантності та глибині отриманих результатів.

Список використаних джерел:

1. Beck, K. et al. Manifesto for Agile Software Development. Manifesto for Agile Software Development. Retrieved March 19, 2025. URL: https://agilemanifesto.org/
2. Salim Inayat S. S., Marczak S., Daneva M., Shamshirband S. A systematic literature review on agile requirements engineering practices and challenges. *Computers in human behavior*. 2015. Vol. 51. P. 915-929. URL: https://doi.org/10.1016/j.chb.2014.10.046.
3. Hoy Z., Xu M. Agile software requirements engineering challenges-solutions – a conceptual framework from systematic literature review. *Information.* 2023. Vol. 14 (6). P. 322. URL: https://doi.org/10.3390/info14060322.
4. IBM (2023, May 5). IBM Engineering Requirements Quality Assistant. IBM Engineering Requirements Quality Assistant – IBM Documentation. Retrieved March 19, 2025. URL: https://www.ibm.com/docs/en/erqa?topic=engineering-requirements-quality-assistant.
5. Gillain J., Jureta I., Faulkner, S. Planning Optimal Agile Releases via Requirements Optimization. *2016 IEEE 24th International Requirements Engineering Conference Workshops*. 2016. URL: https://doi.org/10.1109/REW.2016.36
6. Umoh E., Sampaio P. R. F., Theodoulidis B. REFINTO: An Ontology-Based Requirements Engineering Framework for Business-IT Alignment in Financial Services Organizations. *2011 IEEE International Conference on Services Computing*. 2011. URL: https://doi.org/10.1109/SCC.2011.104.
7. AlDhafer O., Ahmad I., Mahmood S. An end-to-end deep learning system for requirements classification using recurrent neural networks. *Information and Software Technology*. 2022. Vol. 147. URL: https://doi.org/10.1016/-j.infsof.2022.106877.
8. Alhoshan W., Ferrari A., Zhao L. Zero-shot learning for requirements classification: An exploratory study. *Information and Software Technology*. 2023. Vol. 159. URL: https://doi.org/10.1016/j.infsof.2023.107202.
9. Cheng H. et al. Generative AI for Requirements Engineering: A Systematic Literature Review. *Information and Software Technology*. 2024. URL: https://doi.org/10.48550/arXiv.2409.06741, unpublished.
10. Sami M. A. et al. AI based Multiagent Approach for Requirements Elicitation and Analysis. 2024. URL: https://doi.org/10.48550/arXiv.2409.00038, unpublished.

11. Li T., Wang S., Lillis D., Yang Z. Combining Machine Learning and Logical Reasoning to Improve Requirements Traceability Recovery. *Appl. Sci.* 2020. Vol. 10 (20). URL: https://doi.org/10.3390/app10207253.
12. Крук Р., Жуковська Н. Оглядове дослідження проблем та викликів інженерії вимог в Agile-проєктах. *Вісник Національного університету водного господарства та природокористування*. 2024. Вип. 3 (103). С. 195-212. URL: https://ep3.nuwm.edu.ua/29840/. https://doi.org/10.31713/vt3202318.
13. Крук Р., Жуковська Н. Requirements engineering issues and challenges in Ukrainian Agile-projects. Modeling. *Control and Information Technologies: Proceedings of International Scientific and Practical Conference*. 2023. Vol. 6. P. 177-180. URL: https://doi.org/10.31713/MCIT.2023.054.
14. Крук Р., Жуковська Н. Дотримання RMS інструментами вимог якості SRS та дизайн комплементарної системи. *Modeling, Control and Information Technologies: Proceedings of International Scientific and Practical Conference.* 2025. Vol. 7. P. 144-147. URL: https://doi.org/10.31713/MCIT.2024.041.
15. Крук Р., Жуковська Н. Виявлення критичних точок робочих процесів системи управління вимогами Agile проєкту в контексті проблем якості вимог до ПЗ. *Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій*: тези доповідей XII Міжнародної науково-практичної конференції (10-12 грудня 2024 р., м. Запоріжжя). 2025. С. 354-358. ISBN 978-617-529-487-1
16. Крук Р., Жуковська Н. Analysis of bottleneck points based on the software requirements data flow model in Agile projects. Mathematical Modelling and Computing (прийнято до публікації). 2025. (англ.)
17. Atlassian. Jira REST API v3. The Jira Cloud Platform REST API. 2025 Retrieved March 4, 2025. URL: https://developer.atlassian.com/cloud/-jira/platform/rest/v3/intro/#about.
18. Wiegers K., Beatty J. Software Requirements (3rd ed.). *Microsoft Press*. 2013. ISBN: 978-0-7356-7966-5
19. Крук Р., Жуковська Н. Напівавтоматична LLM-базована система управління вимогами до програмного забезпечення. *Міжнародна науково-практична конференція молодих науковців, аспірантів і здобувачів вищої освіти «Проблеми та перспективи розвитку сучасної науки», м. Рівне, 8-9 трав. 2025 р.* Рівне, 2025. URL: https://ep3.nuwm.edu.ua/ (дата звернення: 31.05.2025).
20. Крук Р., Жуковська Н. Toward AI-assisted Framework for Agile Requirement Knowledge Management: Five-Stage Generative LLM Approach. 15th International Conference on Advanced Computer Information Technologies, м. Sibenic, 17-19 верес. 2025 р. (прийнято до публікації) (англ.)

FIVE-STEP SEMANTIC ANALYSIS MIDDLEWARE FOR ATLASSIAN JIRA SOFTWARE REQUIREMENTS TRANSFORMATION USING GENERATIVE LARGE LANGUAGE MODEL

The article presents a framework for transforming requirement artifacts in Agile development environments into a structured knowledge base using large language models (LLMs) and graph-based methods. The study focus-

es on addressing key limitations of contemporary requirements engineering, including tool fragmentation, weak traceability, and insufficient adaptability to semi-structured data. The proposed system operates as an intermediary layer between Atlassian Jira and a graph-based knowledge repository, implementing a five-step methodology: requirement clustering, test linkage, dependency identification, deduplication, and historical reconstruction. The core component is the LLM, which the system interacts with via a JSON-oriented protocol that includes instructions for data interpretation, the expected structure of the response, and permitted actions (e.g., creation, updating, traceability). The modular system architecture includes a user interaction layer, an orchestration agent, a semantic core, a graph database (Neo4j), and an access control subsystem. Both the theoretical and experimental phases of the study were carried out with the support of SoftServe Inc. As part of the experimental deployment, the system processed over 18,000 Jira issues. Pilot testing demonstrated high response accuracy (90.4%) and the potential for significant time savings for analysts and testers. The system also proved capable of reconstructing historical requirement states, detecting duplicates, and identifying logical inconsistencies in dependencies – features particularly valuable for complex products with multi-layered structures. However, the system remains at the prototype stage and has a number of limitations, including dependency on the quality of prompt formulation, challenges in interpreting LLM decisions, and the need for carefully configured secure data access. The results should be considered preliminary; further testing across different projects, domains, and larger query volumes is needed to assess the scalability, robustness, and practical applicability of the proposed approach.

Key words: *requirements engineering, Agile, LLM, requirements analysis, Jira, semantic analysis.*

Отримано: 9.06.2025