https://doi.org/10.31891/2219-9365-2025-84-33
УДК 004.8:004.41:005.311.6

ДЕЙНИКОВСЬКИЙ Максим
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0009-0002-9519-0072
САМОРАЙ Олег
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0009-0000-3384-155X
БОЙКО Микола
Приватний вищий навчальний заклад «Європейський університет»
https://orcid.org/0009-0006-6086-3969

ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ЯК ІНСТРУМЕНТУ КРИТИЧНОГО АНАЛІЗУ ТА ПІДТРИМКИ ПРОЄКТУВАННЯ ПРОГРАМНИХ СИСТЕМ

У статті розглядається застосування методів штучного інтелекту (ШІ) як інструменту критичного аналізу вимог, архітектурних рішень та підтримки процесу проєктування програмних систем. Особливу увагу приділено використанню моделей машинного навчання та великих мовних моделей (LLM) для аналізу технічної документації, виявлення суперечностей у вимогах, прогнозування архітектурних ризиків і оптимізації дизайнерських рішень. Представлено комплексне дослідження, що охоплює аналіз сучасних AI-інструментів, інтеграцію їх у процеси системного аналізу та Software Architecture Design, а також оцінку ефективності таких підходів у реальних проєктах. Наведено приклади застосування ШІ для аналізу UML-діаграм, user stories, нефункціональних вимог і технічного боргу. Отримані результати можуть бути корисними для системних аналітиків, архітекторів ПЗ та команд розробки.

Ключові слова: штучний інтелект, проєктування програмних систем, системний аналіз, архітектура ПЗ, машинне навчання, LLM, технічні вимоги, критичний аналіз.

DEINYKOVSKYI Maksym, SAMORAI Oleh, BOIKO Mykola
Private Higher Educational Institution "European University"

USING ARTIFICIAL INTELLIGENCE AS A TOOL FOR CRITICAL ANALYSIS AND SUPPORT OF SOFTWARE SYSTEM DESIGN

The article investigates the use of artificial intelligence (AI) as an instrument for critical analysis and decision support in the process of software system design. Modern software systems are characterized by high complexity, scalability requirements, and intensive integration with external services, which significantly increases the risks associated with architectural and design decisions. Traditional approaches to system analysis and software architecture design largely rely on expert experience, which may lead to subjectivity, overlooked dependencies, and inconsistencies in requirements.

The study focuses on the application of machine learning techniques and large language models (LLMs) for analyzing technical documentation, user stories, functional and non-functional requirements, UML diagrams, and architectural descriptions. AI-based tools are examined as a means of identifying ambiguities, conflicts, and incompleteness in requirements, as well as predicting architectural risks and potential technical debt at early stages of the software life cycle.

A comprehensive case study is presented based on the design of a medium-complexity web-oriented CRM system. The research covers the full initial design cycle, including requirements analysis, architectural modeling, and evaluation of alternative architectural patterns such as microservices, CQRS, and event-driven architecture. Natural language processing models were used to analyze textual requirements and classify them by priority, business value, and risk level. The results demonstrate that automated AI-based analysis enables the detection of a significant proportion of potential defects and inconsistencies before implementation begins.

The findings confirm that artificial intelligence can effectively support system analysts and software architects by reducing subjectivity, improving the quality of design decisions, and mitigating long-term architectural risks. The article concludes that AI should be considered a complementary analytical tool that enhances expert judgment rather than replacing it. The proposed approach contributes to improving the reliability, maintainability, and scalability of software systems and outlines directions for further research in integrating AI tools into early stages of software engineering processes.

Keywords: artificial intelligence, software system design, system analysis, software architecture, machine learning, LLM, technical requirements, critical analysis.

Стаття надійшла до редакції / Received 06.11.2025
Прийнята до друку / Accepted 10.12.2025

ПОСТАНОВКА ПРОБЛЕМИ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

Сучасні програмні системи характеризуються високою складністю, масштабованістю та інтеграцією з численними зовнішніми сервісами. Процес їх проєктування включає аналіз великої кількості функціональних і нефункціональних вимог, вибір архітектурних стилів, технологічного стеку, а також прогнозування довгострокових наслідків прийнятих рішень. Помилки на етапі проєктування можуть призвести до значних витрат на підтримку, зниження продуктивності системи або неможливості її

подальшого масштабування. Традиційні підходи до системного аналізу значною мірою залежать від досвіду аналітиків і архітекторів, що створює ризики суб'єктивності, пропуску прихованих залежностей або суперечностей у вимогах. В умовах Agile та DevOps, де проєктування є ітеративним і тісно інтегрованим із розробкою, зростає потреба у швидких та обґрунтованих аналітичних рішеннях.

Штучний інтелект відкриває нові можливості для підтримки проєктування програмних систем. Завдяки здатності аналізувати великі обсяги текстових і структурованих даних, AI-інструменти можуть виявляти логічні конфлікти у вимогах, оцінювати відповідність архітектури бізнес-цілям, а також пропонувати альтернативні рішення на основі накопиченого досвіду з аналогічних проєктів. Таким чином, актуальною науково-практичною задачею є дослідження ефективності використання штучного інтелекту як інструменту критичного аналізу та підтримки прийняття рішень у процесі проєктування програмних систем.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

Аналіз сучасних наукових досліджень і публікацій засвідчує стійку тенденцію до зростання інтересу наукової спільноти та індустрії до застосування методів штучного інтелекту в різних галузях програмної інженерії. У працях останніх років активно розглядаються питання автоматизованого аналізу та управління вимогами, інтелектуальної підтримки тестування, генерації програмного коду, його оптимізації та рефакторингу, а також виявлення дефектів і технічного боргу на етапах розроблення й супроводу програмних продуктів. Водночас значно менше уваги приділяється використанню AI саме на етапі архітектурного проєктування, який є критично важливим для забезпечення якості, надійності та довгострокової еволюції програмних систем.

Окремі дослідження демонструють високу ефективність методів обробки природної мови (NLP) для аналізу текстових артефактів проєктування, зокрема user stories, вимог, специфікацій і технічної документації. Такі підходи дають змогу автоматично виявляти неоднозначності, суперечності, дублювання або неповноту вимог, що позитивно впливає на якість початкових етапів розроблення. Разом із тим застосування великих мовних моделей і гібридних AI-підходів для аналізу архітектурних рішень, UML-діаграм, архітектурних стилів і патернів, а також для оцінювання нефункціональних характеристик систем — продуктивності, безпеки, масштабованості, надійності — перебуває на стадії активного становлення та ще не має усталених методологій.

Більшість наявних інструментальних рішень орієнтована на підтримку окремих, ізольованих аспектів життєвого циклу програмного забезпечення, таких як статичний або динамічний аналіз коду, автоматизоване тестування чи генерація документації. Комплексні підходи, що забезпечують інтегровану підтримку системного аналізу та архітектурного проєктування з урахуванням взаємозв'язку вимог, архітектурних рішень і потенційних ризиків, представлені обмежено. Це свідчить про наявність наукової прогалини та актуальність подальших досліджень, спрямованих на розроблення моделей і методів інтеграції AI-інструментів у ранні стадії життєвого циклу програмних систем, де закладаються ключові архітектурні рішення та визначається майбутня якість програмного продукту.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

Метою статті є комплексне та міждисциплінарне дослідження можливостей застосування штучного інтелекту в процесі проєктування програмних систем як інтелектуального засобу підтримки прийняття інженерних рішень. У роботі розглядається роль ШІ не лише як інструменту автоматизації окремих етапів системного аналізу, а як активного учасника процесу формування, оцінювання та вдосконалення архітектурних рішень на всіх рівнях — від аналізу предметної області та формалізації вимог до проєктування архітектури, прогнозування поведінки системи в умовах змін і супроводу її еволюції.

Особливу увагу приділено аналізу сучасних AI-орієнтованих підходів і платформ, що використовуються для роботи з функціональними та нефункціональними вимогами, виявлення прихованих залежностей, конфліктів і неоднозначностей, а також для раннього виявлення архітектурних ризиків, обмежень і технічного боргу. Окрім цього, стаття спрямована на обґрунтування концептуальної моделі інтеграції AI-інструментів у традиційні методології проєктування програмного забезпечення, з урахуванням питань масштабованості, якості, безпеки, етичних аспектів і впливу таких інструментів на роль і відповідальність системного аналітика та програмного архітектора.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

У межах даного дослідження було змодельовано процес проєктування веборієнтованої програмної системи середньої складності на прикладі CRM-системи, призначеної для управління клієнтськими даними, взаємодією з користувачами та підтримки основних бізнес-процесів організації. Основною метою дослідження є аналіз можливостей застосування інструментів штучного інтелекту на етапі архітектурного проєктування програмного забезпечення та оцінка їх впливу на якість проєктних рішень, рівень ризиків і потенційний технічний борг.

Архітектура дослідження охоплювала повний початковий цикл проєктування програмної системи. На першому етапі здійснювався аналіз бізнес-вимог, які були сформульовані у вигляді user stories. Вони описували ключові сценарії використання CRM-системи, зокрема управління контактами клієнтів, обробку угод, контроль взаємодії з клієнтами, формування аналітичних звітів та інтеграцію із зовнішніми сервісами. На рис. 1 схематично зображено загальну архітектуру дослідження та взаємозв'язок основних етапів проєктування.

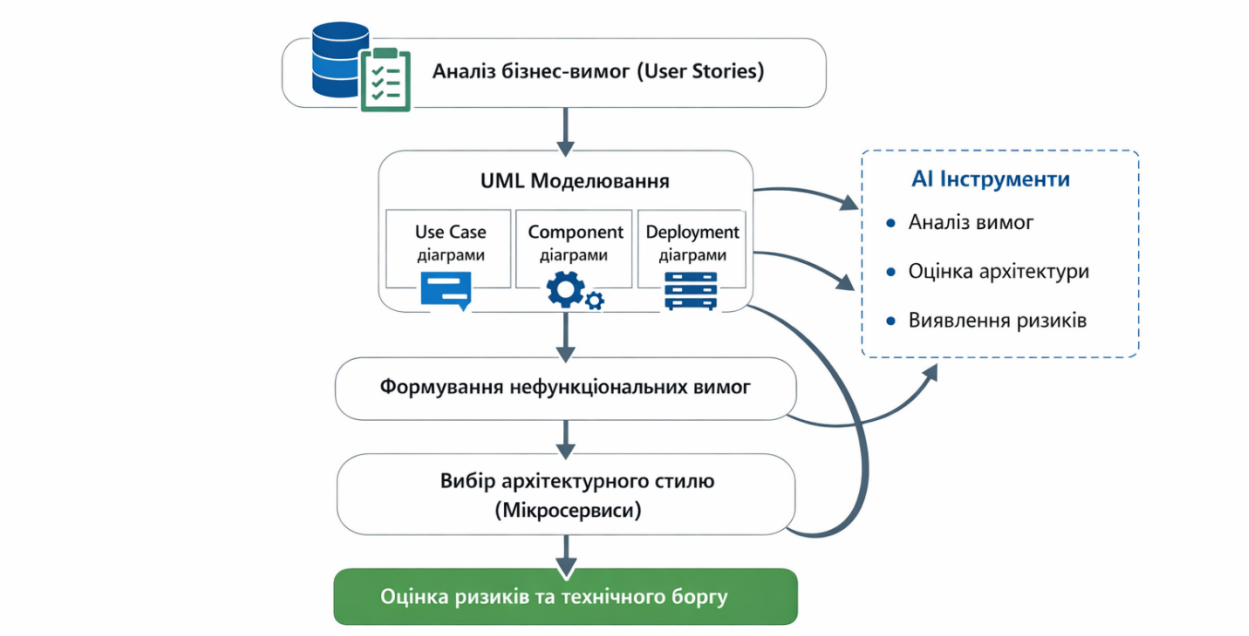

Рис. 1. Загальна архітектура дослідження

Наступним кроком стало формалізоване моделювання системи з використанням UML-діаграм. Було побудовано діаграми варіантів використання (Use Case), компонентів (Component) та розгортання (Deployment), що дозволило наочно представити функціональну структуру системи, взаємодію між її складовими та особливості розміщення компонентів у середовищі виконання. На рис. 2 зображено приклад діаграми компонентів CRM-системи, що відображає взаємодію між сервісами управління клієнтами, аналітики та автентифікації.

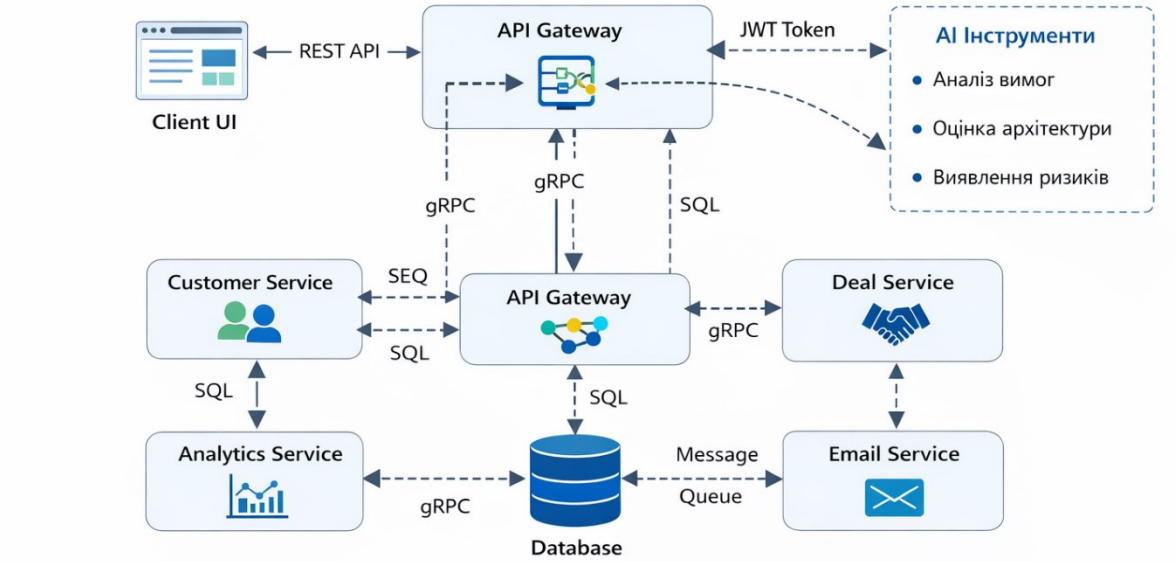

Рис. 2. Діаграма компонентів CRM-системи

Окрему увагу було приділено формуванню нефункціональних вимог, які мають критичне значення для систем середньої та високої складності. До таких вимог належали масштабованість, продуктивність, відмовостійкість, безпека та підтримуваність. З урахуванням зазначених вимог як базовий архітектурний стиль було обрано мікросервісну архітектуру. Вона забезпечує можливість незалежного розвитку окремих модулів, горизонтального масштабування та спрощення супроводу системи. На рис. 3 наведено узагальнену

схему мікросервісної архітектури CRM-системи.

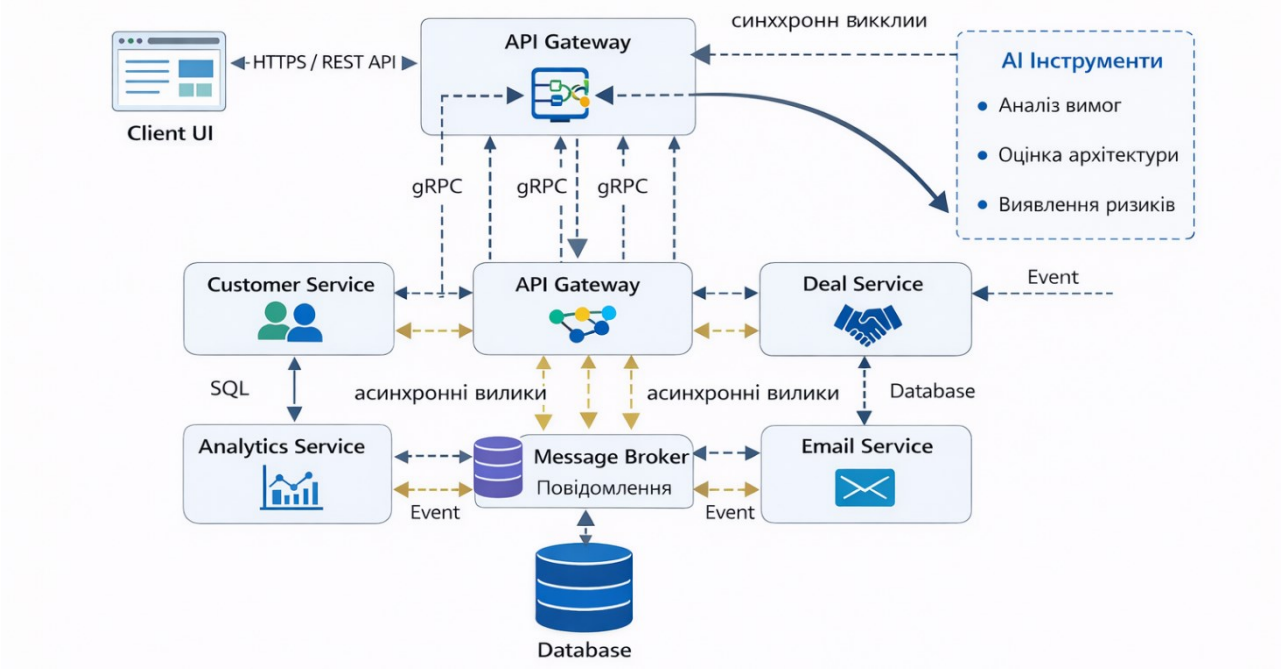

Рис. 3. Мікросервісна архітектура CRM-системи

Важливою складовою дослідження став аналіз вимог із використанням інструментів штучного інтелекту. Для цього застосовувалися моделі обробки природної мови (NLP), які аналізували текстові описи user stories. Основними завданнями AI-аналізу були виявлення суперечностей між вимогами, визначення неповних або нечітких формулювань, а також класифікація вимог за пріоритетами, бізнес-цінністю та рівнем ризику.

Результати аналізу показали, що автоматизований підхід дозволяє виявити значну кількість проблем ще до початку етапу реалізації. Зокрема, було встановлено, що близько 20% потенційних дефектів вимог можуть бути ідентифіковані на ранній стадії проєктування. У табл. 1 наведено приклад узагальнених результатів AI-аналізу вимог.

Таблиця 1

Результати автоматизованого аналізу вимог

Тип проблеми	Характеристика	Приклад
Нечіткі вимоги	Відсутність кількісних критеріїв	«Система повинна швидко відповідати»
Конфліктні вимоги	Логічні суперечності	Різні рівні доступу для однієї ролі
Неповні вимоги	Відсутні обмеження	Не визначено вимоги до SLA

Таким чином, застосування NLP-моделей дозволило підвищити якість вимог, зменшити ризик неправильного трактування та полегшити подальше архітектурне проєктування.

На етапі архітектурного проєктування великі мовні моделі використовувалися для аналізу прийнятих рішень з урахуванням нефункціональних вимог. AI-інструменти здійснювали оцінку відповідності мікросервісної архітектури вимогам масштабованості та підтримуваності, а також пропонували альтернативні архітектурні підходи. Зокрема, були розглянуті патерни CQRS та Event-Driven Architecture. На рис. 4 наведено порівняльну схему традиційної мікросервісної архітектури та архітектури з використанням CQRS.

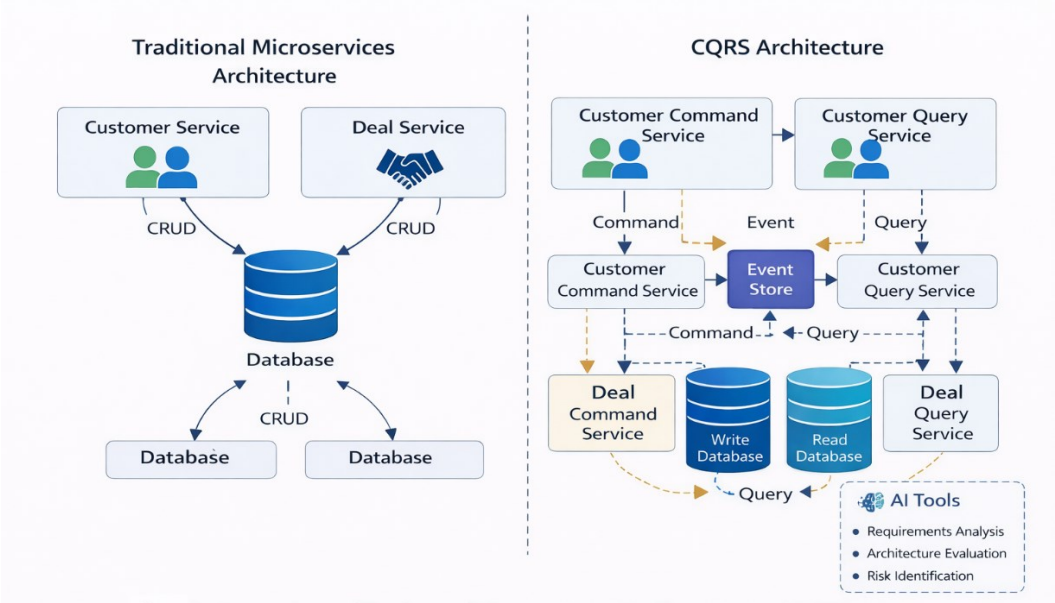

Рис. 4. Comparison of Traditional Microservices Architecture and CQRS

AI-аналіз показав, що застосування CQRS може суттєво підвищити продуктивність операцій читання та спростити масштабування, проте водночас ускладнює реалізацію та синхронізацію даних. Event-Driven Architecture, у свою чергу, зменшує зв'язаність компонентів та підвищує гнучкість системи, але потребує додаткових механізмів обробки подій і моніторингу. У лістингу 1 наведено приклад псевдокоду обробки події у подієво-орієнтованій архітектурі.

```
/**
 * Event published when a new customer is created in the system
 */
public class CustomerCreatedEvent {

    private final UUID customerId;
    private final String email;
    private final LocalDateTime createdAt;

    public CustomerCreatedEvent(UUID customerId, String email) {
        this.customerId = customerId;
        this.email = email;
        this.createdAt = LocalDateTime.now();
    }

    public UUID getCustomerId() {
        return customerId;
    }

    public String getEmail() {
        return email;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }
}
```

```
/**
 * Event handler responsible for customer-related side effects
 */
@Component
public class CustomerEventListener {

    private final NotificationService notificationService;
    private final AnalyticsService analyticsService;
    private final AuditService auditService;

    public CustomerEventListener(NotificationService notificationService,
                                 AnalyticsService analyticsService,
                                 AuditService auditService) {
        this.notificationService = notificationService;
        this.analyticsService = analyticsService;
        this.auditService = auditService;
    }

    /**
                                                                usly

        // Send welcome email to the new customer
        notificationService.sendWelcomeEmail(
                event.getCustomerId(),
                event.getEmail()
        );                                                      atedEvent event) {

        // Register customer in analytics subsystem
        analyticsService.registerNewCustomer(
                event.getCustomerId(),
                event.getCreatedAt()
        );

        // Persist audit log entry
        auditService.logEvent(
                "CUSTOMER_CREATED",
                event.getCustomerId().toString(),
                event.getCreatedAt()
        );
    }
```

Лістинг 1. Приклад обробки події в Event-Driven архітектурі

Ще одним важливим аспектом дослідження стало виявлення потенційного технічного боргу та архітектурних ризиків. На основі історичних даних аналогічних проєктів AI-інструменти прогнозували проблемні зони, які можуть негативно вплинути на подальший розвиток системи. До таких зон належали надмірна зв'язаність між сервісами, ризики масштабування бази

даних та складність тестування окремих модулів. На рис. 5 зображено приклад схеми виявлення архітектурних ризиків у мікросервісній системі.

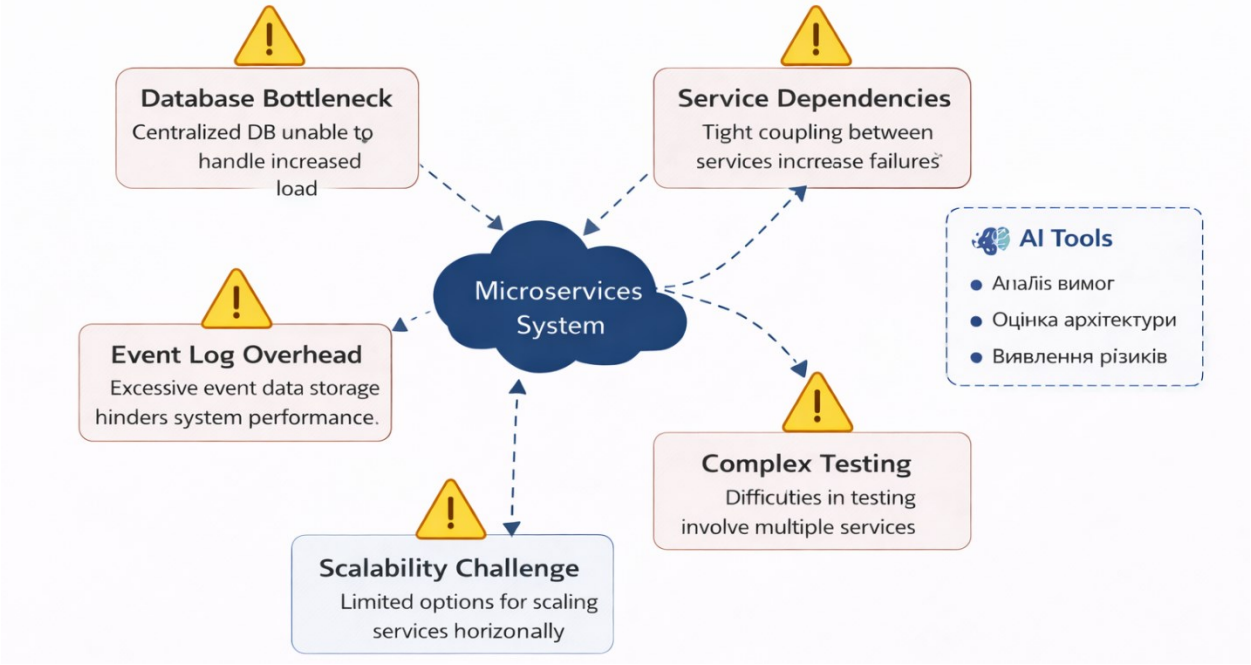

Рис. 5. Схема виявлення архітектурних ризиків у мікросервісній системі

Отримані результати дозволили сформувати рекомендації щодо зменшення технічного боргу, зокрема шляхом ізоляції даних окремих сервісів, впровадження контрактного тестування та чіткого визначення API-інтерфейсів.

У підсумку було запропоновано модель інтеграції AI у процес архітектурного проєктування, за якої штучний інтелект виступає як допоміжний аналітичний інструмент, що підтримує архітектора у прийнятті рішень, але не замінює експертну оцінку. Такий підхід дозволяє зменшити ризики, підвищити обґрунтованість архітектурних рішень та покращити якість програмної системи ще на ранніх етапах її життєвого циклу.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У статті досліджено можливості застосування штучного інтелекту як інструменту критичного аналізу на етапі архітектурного проєктування програмних систем. На прикладі веборієнтованої CRM-системи середньої складності продемонстровано, що інтеграція AI-інструментів у процес аналізу вимог, вибору архітектурних рішень та оцінки ризиків дозволяє суттєво підвищити ефективність і обґрунтованість проєктних рішень.

Зокрема, показано, що використання моделей обробки природної мови для аналізу бізнес-вимог у вигляді user stories дає змогу виявляти суперечності, нечіткі формулювання та пропуски ще на ранніх етапах життєвого циклу програмного забезпечення. Це зменшує ймовірність виникнення критичних помилок на етапах реалізації та тестування, а також сприяє зниженню витрат на доопрацювання системи. Результати дослідження підтверджують, що автоматизований аналіз вимог може виявляти значну частину потенційних проблем до початку розробки.

У межах архітектурного проєктування штучний інтелект продемонстрував ефективність як інструмент підтримки прийняття рішень. AI-інструменти дозволяють аналізувати відповідність обраного архітектурного стилю нефункціональним вимогам, порівнювати альтернативні архітектурні патерни, такі як мікросервісна архітектура, CQRS та Event-Driven Architecture, а також оцінювати їхній вплив на масштабованість, продуктивність і підтримуваність системи. Важливою перевагою такого підходу є зменшення суб'єктивності архітектурних рішень за рахунок використання аналітичних і прогнозних можливостей AI.

Окрему увагу приділено виявленню архітектурних ризиків і потенційного технічного боргу. На основі історичних даних та типових шаблонів проєктування AI-інструменти здатні ідентифікувати проблемні зони, зокрема надмірну зв'язаність компонентів, вузькі місця у масштабуванні бази даних та складність тестування мікросервісних систем. Це дозволяє архітекторам заздалегідь враховувати можливі негативні наслідки та приймати превентивні рішення.

У результаті дослідження запропоновано модель інтеграції штучного інтелекту в процес архітектурного проєктування, за якої AI виступає як допоміжний аналітичний інструмент, що підсилює експертну оцінку, але не замінює роль архітектора. Такий підхід забезпечує баланс між автоматизацією та

професійним досвідом, підвищує якість архітектурних рішень і сприяє створенню більш надійних та гнучких програмних систем.

Перспективами подальших досліджень є розробка спеціалізованих AI-моделей для аналізу архітектурних артефактів, зокрема UML-діаграм, архітектурних описів та нефункціональних вимог. Важливим напрямом також є інтеграція AI-інструментів з CASE-засобами проєктування та системами управління життєвим циклом програмного забезпечення. Окремого вивчення потребує оцінка довгострокового впливу використання штучного інтелекту на підтримуваність, еволюцію та загальну вартість володіння програмними системами.

Література

1. Bucaioni A., Weyssow M., He J., Lyu Y., Lo D. Штучний інтелект для архітектури програмного забезпечення: огляд літератури та подальші напрями досліджень. *Proceedings of the ACM Foundations of Software Engineering (FSE)*, 2025.
2. Esposito M., Li X., Moreschini S., Ahmad N., Cerny T., Lenarduzzi V., Taibi D. Генеративний штучний інтелект для архітектури програмного забезпечення: застосування, тенденції, виклики та перспективи. *arXiv preprint arXiv:2503.13310*, 2025.
3. Ebrahim M. Інженерія програмних вимог із використанням мовних моделей. *Proceedings of the ACL Student Research Workshop*, 2025.
4. Using Large Language Models in Software Requirements Analysis. Використання великих мовних моделей в аналізі програмних вимог. *Edelweiss Applied Science and Technology*, т. 9, № 3, 2025.
5. Radliński Ł. Великі мовні моделі для оцінювання програмних проєктів на ранніх етапах. *Applied Sciences*, 2025.
6. Nyaga F. AI-орієнтована інженерія програмного забезпечення: систематичний огляд впливу машинного навчання та перспектив розвитку. *Preprints.org*, 2025.
7. Ahmed I. Штучний інтелект в інженерії програмного забезпечення: еволюція та інтеграція великих мовних моделей. *ACM Computing Surveys*, 2025.
8. AI-Driven Decision Support Systems for Software Architecture. Системи підтримки прийняття рішень для архітектури програмного забезпечення на основі штучного інтелекту. *Journal of Computational Artificial Intelligence*, 2025.
9. Zhang Q., Fang C., Xie Y., Zhang Y., Yang Y., Chen Z. Огляд використання великих мовних моделей в інженерії програмного забезпечення. *arXiv preprint arXiv:2312.15223*, 2023.
10. Helmi A. ARLO: трансформація вимог природною мовою в архітектуру програмних систем із використанням великих мовних моделей. *arXiv preprint arXiv:2504.06143*, 2025.

References

I. Bucaioni, M. Weyssow, J. He, Y. Lyu and D. Lo, *Artificial Intelligence for Software Architecture: Literature Review and the Road Ahead*, ACM/ACM FSE, 2025.
2. M. Esposito, X. Li, S. Moreschini, N. Ahmad, T. Cerny, K. Vaidhyanathan, V. Lenarduzzi and D. Taibi, *Generative AI for Software Architecture: Applications, Trends, Challenges, and Future Directions*, arXiv:2503.13310, 2025. arXiv
3. M. Ebrahim, *Software Requirements Engineering with Language Models*, in Proc. ACL SRW, 2025. ACL Anthology
4. "Using Large Language Models in Software Requirements Analysis", *Edelweiss Applied Science and Technology*, vol. 9, no. 3, 2025 (LLM-based requirement analysis research). Learning Gate
5. Ł. Radliński, *Large Language Models for Early-Stage Software Project Estimation*, Appl. Sci., 2025. MDPI
6. F. Nyaga, *AI-Driven Software Engineering: A Systematic Review of Machine Learning's Impact and Future Directions*, Preprints.org, 2025. Preprints
7. Ahmed, *Artificial Intelligence for Software Engineering: The Journey and LLM Integration*, *ACM Comput. Surv.*, 2025. ACM Digital Library
8. "AI-Driven Decision Support Systems for Software Architecture", *J. Comput. Artif. Intel.,* vol. –, 2025 (review of AI methods including ML & LLM for architecture). Brain
9. Q. Zhang, C. Fang, Y. Xie, Y. Zhang, Y. Yang and Z. Chen, *A Survey on Large Language Models for Software Engineering*, arXiv:2312.15223, 2023. GitHub
10. Helmi, *ARLO: Transforming Natural Language Requirements Into Architecture Using LLMs*, arXiv:2504.06143, 2025. arXiv