УДК 004.4:004.8 https://doi.org/10.15407/pp2026.02.004

О.В. Нестеренко, В.В. Федоров, П.П. Яцук

AI-ЛАНДШАФТ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Штучний інтелект стає невід'ємною частиною сучасної програмної інженерії, розширюючи можливості автоматизації, оптимізації й підтримки технічних рішень протягом усього життєвого циклу програмного забезпечення. У статті запропоновано методологію визначення інструментарію штучного інтелекту для вирішення задач програмної інженерії. Базою цієї методології є системна структуризація на основі галузей знань (Knowledge Area) нової версії міжнародного посібника SWEBOK V4 та сукупності певних типів процесів або діяльності (Topics), з яких складаються галузі знань. Для структуризації вибрано головні області (шість областей для розроблення ПЗ) та допоміжні організаційні сфери (шість областей, що забезпечують інженерію та керування розробкою ПЗ). Розглянуто застосування в програмній інженерії таких моделей і алгоритмів штучного інтелекту як машинне навчання (ML), глибинне навчання (DL), великі мовні моделі (LLM), обробка природної мови (NLP), генеративний ШІ (Generative AI), графові алгоритми, метаевристики та оптимізаційні алгоритми, експерті системи та аналітика процесів. У результаті побудовано загальну онтологію AI-ландшафту програмної інженерії. Детально розглянуто карту застосування ШІ в програмній інженерії, яка відображає відповідність між галузями знань SWEBOK, процесами, ШІ-технологіями та інструментами. Запропонована картографія демонструє три важливі принципи інтеграції AI в програмну інженерію: технологічний рівень; процесний та інструментальний рівні. Такий підхід дозволяє систематизувати використання ШІ, уникнути фрагментарного впровадження технологій, а також оцінювати рівень інтелектуалізації процесів розробки. Окремо в статті розглянуто модель зрілості для впровадження штучного інтелекту в галузі програмної інженерії (AI-Driven Software Engineering CMM), яка створена для допомоги підприємствам та організаціям оцінити й вдосконалити свої зусилля з інтеграції ШІ та програмної інженерії. Результати дослідження можуть бути використані для перетворення цього симбіозу з ідеї на робочу парадигму, зокрема, з урахуванням надійності, пояснюваності та етичних аспектів використання ШІ.

Ключові слова: штучний інтелект, SWEBOK, моделі і алгоритми, модель зрілості, програмне забезпечення, людино-машинна взаємодія.

O.V. Nesterenko, V.V. Fedorov, P. P. Yatsuk

AI LANDSCAPE OF SOFTWARE ENGINEERING

Artificial intelligence is becoming an integral part of modern software engineering, expanding the possibilities of automation, optimization and support of technical solutions throughout the software life cycle. The article proposes a methodology for defining artificial intelligence tools for solving software engineering problems. The basis of this methodology is a system structuring based on the knowledge areas of the new version of the international manual SWEBOK V4 and a set of certain types of processes or activities (Topics), which consist of knowledge areas. The main areas (six areas for software development) and auxiliary organizational areas (six areas that provide engineering and management of software development) were selected for structuring. The application of such artificial intelligence models and algorithms in software engineering as machine learning (ML), deep learning (DL), large language models (LLM), natural language processing (NLP), Generative AI, graph algorithms, metaheuristics and optimization algorithms, expert systems and process analytics is considered. As a result, a general ontology of the AI landscape of software engineering is built. A map of the application of AI in software engineering is considered in detail, which reflects the correspondence between SWEBOK knowledge areas, processes, AI technologies and tools. The proposed cartography demonstrates three important principles of integrating AI into software engineering: technological level; process level and instrumental level. This approach allows you to systematize the use of AI, avoid fragmented implementation of technologies, and also assess the level of intellectualization of development processes. Separately, the article discusses the AI-Driven Software Engineering Maturity Model, which was created to help enterprises and organizations assess and improve their efforts to integrate AI and SE. The results of the study can be used to transform this symbiosis from an idea into a working paradigm, in particular, taking into account the reliability, explainability and ethical aspects of using AI.

Keywords: artificial intelligence, SWEBOK, models and algorithms, maturity model, software, human-machine interaction.

Вступ

Використання у різних сферах діяльності засобів на основі технологій штучного інтелекту (ШІ) стрімко зростає. Не оминули нові підходи й вирішення задач програмної інженерії (ПІ). ШІ стає невід'ємною частиною сучасної програмної інженерії, розширюючи можливості автоматизації, оптимізації й підтримки технічних рішень протягом усього життєвого циклу програмного забезпечення (ПЗ). Застосування таких технологій, як машинне навчання (ML), глибинне навчання (DL), великі мовні моделі (LLM) та обробка природної мови (NLP) вже змінило традиційні підходи до розробки ПЗ. Результатом стало визначення перспективних напрямків застосування ШІ для покращення якості програмного забезпечення як от управління життєвим циклом розробки ПЗ, прогнозування помилок/дефектів та оцінки зусиль, інтелектуалізація програмної інженерії в управлінні кодом, обробка природної мови в інженерії вимог, а також пошук даних з репозиторіїв програмного забезпечення [1]. Найпоширеніше використання ШІ охоплює такі фази ПІ, як вимоги, проєктування, розробка, тестування, випуск та обслуговування [2].

Застосування технологій штучного інтелекту для вирішення проблем програмної інженерії не є виключно новою тенденцією, а радше результатом багаторічних зусиль спільноти програмної інженерії з розробки автоматизованих та інтелектуальних інструментів та фреймворків для забезпечення ефективного досвіду розробки ПЗ [3]. У програмній інженерії відбувається зміна парадигм, і саме системи штучного інтелекту відіграють дедалі важливішу роль у підвищенні продуктивності цих процесів. Основне питання, яке повинні вирішувати як дослідницька спільнота ШІ, так і програмної інженерії полягає у тому, як синергетично інтегрувати новий «обчислювальний інтелект» та людський інтелект [4].

Варто зазначити, що через свою «чорну скриньку» ці перспективні моделі та технології на основі ШІ все ще далекі від практичного впровадження в програмній інженерії. Зокрема, ШІ викликає багато занепокоєнь щодо його відповідального використання. Результати багатьох досліджень свідчать не лише про зростаючу інтеграцію ШІ в різні галузі людської діяльності, а й вказують, у першу чергу, на збільшення уваги до етичних та правових аспектів використання ШІ [5]. Відсутність пояснень у рішеннях, що пропонує ШІ, створює небажані ризики для їхнього застосування в критично важливих завданнях ПІ, таких, наприклад, як виявлення вразливостей, де прозорість ухвалення рішень має першорядне значення [6]. У зв'язку із цим існує потреба зрозуміти та розробити принципи того, що передбачає відповідальна розробка ПЗ на практиці, а також ретельно оцінити результативність впровадження інструментів та методів ШІ в програмну інженерію [7].

Хоча чимало досліджень підкреслюють вагому роль автоматизації на базі ШІ у покращенні ухвалення рішень, оптимізації робочих процесів та скороченні часу розробки, однак звертається увага на те, шо саме традиційні методології все ще забезпечують структуру, відповідність та надійність. Тому вважається, що майбутнє за гібридною моделлю, яка поєднує традиційні підходи з автоматизацією на основі ШІ [8].

Отже, епоха Software Engineering 2.0, пов'язана з розвитком мікроелектроніки, поширенням Інтернету та впровадженням методологій командної роботи відходить у минуле. Інтеграція ШІ в програмну інженерію відкриває перспективи нової ери Software Engineering 3.0, що має величезний потенціал для підвищення ефективності та якості ПЗ. Технології ШІ несуть переосмислення традиційних практик розробки програмного забезпечення, революціонізують програмну інженерію, впроваджуючи інтелектуальну автоматизацію у життєвий цикл ПЗ [9].

Хоча за останні роки відбувся значний прогрес у перетині програмної інженерії та штучного інтелекту, щоб перетворити цей симбіоз з ідеї на робочу парадигму необхідні значні дослідницькі зусилля, зокрема щодо надійності, пояснюваності та етичних аспектів використання ШІ.

Методика визначення AI-інструментарію

Розглядаючи питання визначення AI-інструментарію для застосування в програмній інженерії, доцільно виходити з того, що ПІ як інженерна дисципліна – це сукупність прийомів виконання діяльності, пов'язаної з виготовленням програмних продуктів і систем (ППС) для різних видів цільових об'єктів із застосуванням різних методів, засобів та інструментів як складових програмної інженерії. З інженерної точки зору в ПІ розв'язуються задачі розробки ПЗ та виготовлення ППС, подані як технологічні процеси формування вимог, проєктування і супроводу продукту, а також перевірки операцій базового процесу щодо правильності виконання різних функціональних задач та виконання робіт за проєктом.

Основні необхідні компетентності в професійній практиці програмної інженерії як сфері діяльності з розробки ПЗ базуються на міжнародних рекомендацій-них документах з програмної інженерії. Основним з них є ядро знань ПІ – посіб-ник SWEBOK (Software Engineering Body of Knowledge). Це набір теоретичних концепцій і формальних визначень методів і засобів розробки та керування програмними проектами, які можуть застосовуватися в інженерії програмування. Починаючи з пробної версії, опублікованої 2001 року, версії 2004 року та широкого використання версії 2014 року (SWEBOK V3) цей посібник в межах спільноти розробників програмного забезпечення відіграє важливу роль як флагманський основоположний та структурний документ для створення програмних продуктів.

Метою розробки оновлення цього посібника 2025 року (SWEBOK V4) є покращення його актуальності, читабельності, узгодженості та зручності використання, формування єдиної концептуальної основи для організації, управління та вдосконалення процесів створення ПЗ. Зміст нової версії цього посібника зріс уже до 18 галузей знань (Knowledge Area), кожна з яких відповідає сукупності певних типів процесів або діяльності (Topics).

Понятійний апарат галузей знань (областей) SWEBOK можна умовно поділити на головні (шість областей для розроблення ПЗ), допоміжні організаційні області (шість областей, що забезпечують інженерію та керування розробкою ПЗ) та базові галузі знань.

Головні галузі – це Вимоги до ПЗ (Software Requirements), Архітектура ПЗ (Software Architecture), Проєктування ПЗ (Software Design), Конструювання ПЗ (Software Construction), Тестування ПЗ (Software Testing) та Супровід ПЗ (Software Maintenance). Допоміжні – це Операції програмної інженерї (Software Engineering Operations), Керування конфігурацією ПЗ (Software Configuration Management), Процес програмної інженерії (Software Engineering Process), Моделі та методи інженерії ПЗ (Software Engineering Models and Methods), Управління програмною інженерією (Software Engineering Management) та Якість ПЗ (Software Quality).

Враховуючи, що SWEBOK відіграє ключову роль у формуванні системного підходу до виробництва програмних продуктів, забезпечуючи структуровану модель знань, яка підтримує організацію, управління та вдосконалення процесів ПІ, вважається за доцільне спиратися на запропоновану в SWEBOK структуризацію для визначення інструментарію штучного інтелекту, що знаходить застосування в програмній інженерії. Такий підхід дозволяє пов'язати AI-технології з конкретними сферами діяльності в ПІ, системно визначити, де саме і які AI-інструменти можуть бути використані.

Таким чином методика визначення AI-інструментарію полягає у реалізації кількох кроків:

а) ідентифікація галузі знань за SWEBOK,

б) аналіз ключових процесів на основі топіків,

в) визначення задач автоматизації за допомогою ШІ,

г) відбір відповідних AI-технологій (моделей і алгоритмів),

д) визначення конкретних інструментів.

Використання SWEBOK як методологічної основи визначення AI-інструментарію має низку переваг, як от забезпечення повноти аналізу завдяки охопленню всіх аспектів програмної інженерії, системність через чітку класифікацію застосувань AI, порівнюваність через можливість оцінювання різних інструментів, нарешті, узгодженість із міжнародними стандартами.

AI моделі і алгоритми

Серед моделей штучного інтелекту основним вважається машинне навчання (Machine Learning – ML), що вже тривалий час розвивається й успішно застосовується в різних галузях. В останні роки використання методів машинного навчання стало одним з основних і для вирішення багатьох завдань програмної інженерії (ML4SE). Типовими серед них є прогнозування дефектів, аналіз якості коду, оцінювання ризиків проєкту, оптимізація процесів розробки. Цього було досягнуто завдяки використанню найсучасніших моделей ML, які, як правило, є складнішими та нестандартними. Однак їх використання призводить до отримання менш з'ясованих рішень, що знижує довіру професіоналів галузі до використання рішень ML4SE. Одним з потенційних напрямків для зниження негативу відсутньої пояснивості є пропонування з'ясованих методів штучного інтелекту (explainable AI – XAI). На цей підхід (XAI4SE) на сьогодні значною мірою звертається увага у спільноті програмної інженерії [10].

Подальший розвиток машинного навчання, пов'язаний із навчанням з підкріпленням (Reinforcement Learning – RL) та глибинним навчанням (Deep Learning – DL), а також глибинним навчанням з підкріпленням (DRL) на сучасному етапі відіграє важливу роль у створенні просунутих засобів ШІ. Ці моделі успішно використовуються й для автоматизації складних завдань програмної інженерії, таких як тестування ігор, вирішення задач планування робіт, а також для навчання ефективної та економічно вигідної поведінки в різних середовищах спеціалізованих програмних агентів [11].

Застосування вищезазначених моделей забезпечило розвиток напряму обробки природної мови (Natural Language Processing – NLP), що відчутно спростило практику ПІ в автоматизації класифікації вимог, аналізі настроїв користувачів та управлінні проєктами, зокрема й щодо супроводу програмного забезпечення [12]. Типовими задачами, що вирішуються на основі NLP є аналіз текстових вимог, генерація документації, пошук невідповідностей у специфікаціях, аналіз технічної документації тощо.

Безумовно, на сьогодні найпопулярнішим є впровадження інструментів штучного інтелекту на основі великих мовних моделей (Large Language Models – LLM). Результати показують, що дослідники ПІ також визнають перевагу внеску LLM, однак лише коли ці моделі застосовується до невеликих, вузькоспеціалізованих та перевірюваних завдань, а не до складних, що не мають запланованого завершення і можуть розвиватися кількома способами (open-ended tasks). LLM використовується як додатковий інструмент до традиційних методів програмної інженерії щодо таких завдань з високими ставками, які потребують справжніх людських зусиль та емоційної відданості [13].

Також необхідно зазначити, що для моделей ШІ, в тому числі і LLM, головною потребою є дані. Ефективність цих моделей залежить від максимізації джерел високоякісних даних. Однак дані, особливо високої якості, часто мають комерційну або конфіденційну цінність, що робить їх менш доступними для застосування засобів ШІ в програмних проєктах, особливо в проєктах створення ПЗ з відкритим кодом. Ця реальність створює значну перешкоду для розробки та впровадження інструментів на основі ШІ в спільноті ПІ [14].

Серйозними задачами програмної інженерії є перевірки коду. У цьому за своєю суттю людиноцентричному процесі важливо зрозуміти, як інженерам-програмістам орієнтуватися у впровадженні ШІ у спільні з ним робочі процеси, адже діяльність із перевірки коду є багатовимірною, охоплюючи когнітивні, емоційні та поведінкові виміри. Впровадження

перевірок за допомогою LLM впливає на деякі з цих атрибутів. Наприклад, у роботі з LLM потреба в емоційній регуляції та механізмах подолання стає меншою, однак когнітивне навантаження іноді вище, зокрема, під час роботи зі зворотним зв'язком, згенерованим LLM, через його надмірну деталізацію. Водночас сприйняття зворотного зв'язку від LLM обмежене недовірою та відсутністю пояснювального контексту в перевірці [15].

Перспективним напрямком є застосування в програмній інженерії багатоагентних автономних систем (Multi-agent autonomous systems – MAS), які краще справляються із завданнями, що охоплюють кілька сфер, ніж окремі автономні агенти. Сучасні дослідження MAS в ПІ зосереджені на інтеграції LLM в ядро автономних агентів для створення багатоагентних систем. Однак впровадження таких систем створює безліч проблем, серед яких однією з основних є стратегічний розподіл завдань між людьми та MAS надійним чином [16].

Отже, LLM розширюють дослідницькі можливості в галузі ПІ завдяки прискореному генеруванню ідей та автоматизованим процесам, роблячи деякі традиційні практики застарілими. Однак людиноцентрична перспектива залишається важливою. Забезпечення людського нагляду та інтерпретації необхідне для підтримки наукової точності, сприяння етичній відповідальності та стимулювання прогресу в цій галузі [17].

В програмній інженерії як суттєвий чинник розглядається вплив поточного стану команди розробників на процеси розробки програмного забезпечення. Визначається, що управління проєктами має базуватися не лише на його параметрах та моделі предметної області як основи бази знань для підтримки управлінських рішень, а й на особистісних характеристиках програмістів відповідно до методології групової динаміки та комунікацій [18]. У нових умовах вже прослідкується командна робота людина-машина (Human-machine teaming - HMT), тобто взаємодія людей і машини як членів команди. Цей підхід може бути корисним у ШІ4SE, але питання впливу HMT на ефективність команди отримали ще мало уваги у спільноті програмної інженерії [19].

Емерджентні властивості LLM привносять новизну та креативність у застосування в усьому спектрі діяльності програмної інженерії, включаючи кодування, проєктування, вимоги, виправлення, рефакторинг, покращення продуктивності, документацію та аналітику. Однак ці ж емерджентні властивості створюють і значні технічні проблеми. Потрібні методи, які можуть надійно виявляти неправильні рішення, притаманні LLM, такі як галюцинації. Тому ключову роль у розробці та впровадженні надійних, ефективних та результативних методів ПІ на основі LLM мають відігравати гібридні методи (традиційна ПІ плюс LLM) [20].

Досягнення в сфері LLM привели до широкого поширення генеративного ШІ (Generative AI), або чат-ботів великих мовних моделей. В галузі програмної інженерії використання генеративного ШІ також може бути корисним інструментом, адже за його допомогою можливо підтримувати всі фази розробки програмного забезпечення, вирішуючи такі типові задачі як аналіз вимог, генерація програмного коду, створення тестів, автоматичне створення прототипів, генерація API та документації [21]. Завдяки використанню генеративних моделей, таких як попередньо натреновані трансформери (GPT), система здатна розуміти складні запити користувачів, підтримувати контекстну зв’язність у розмовах і автономно виконувати завдання з мінімальним втручанням людини [22]. Однак, можуть виникнути й різні проблеми, зокрема щодо правдивості відповідей, наприклад, згенерованих ChatGPT. Це потрібує додатковиї засобів для виявлення некоректностей у відповідях [23].

Оцінюючи інструменти генеративного ШІ на основі таких критеріїв, як зручність використання, ефективність та інтеграція в існуючі робочі процеси, дослідження підкреслюють важливість співпраці людини та ШІ. Припускається, що хоча генеративний ШІ може суттєво підтримувати завдання розробки програмного забезпечення, залишаються важливими люд-

ський нагляд та критична оцінка результатів, створених ШІ [24].

Для ефективного розв'язання реальних задач програмування перспективними методами є застосування за допомогою штучного інтелекту графових алгоритмів та структур даних. Типовими задачами, що вирішуються, є аналіз структури програмних систем, дослідження залежностей між компонентами, аналіз конфігурацій, impact-analysis змін. У конструюванні ефективних і оптимізованих рішень для конкретної проблеми ПІ важливим є акцент на практичну імплементацію алгоритмів, що особливо актуально для систем із великими обсягами даних (Big Data, фінансові або медичні системи).

Також ефективність у вдосконаленні таких завдань ПІ, як управління проєктами, прогнозування розташування функцій та дій з модифікації програмного забезпечення демонструють метаевристики та оптимізаційні алгоритми, підвищуючи точність та ефективність у цих завданнях [12]. Також типовими задачами можуть бути оптимізація архітектур, оптимізація тестових наборів, оптимізація планування розробки.

Не залишаються поза увагою й застосування ШІ в таких інструментах, як Process Mining, експерті системи та аналітика процесів. Типові задачі – аналіз процесів розробки ПЗ, оцінювання зрілості процесів, вдосконалення SDLC, управління змінами, зокрема в таких SWEBOK-областях як базовий процес програмної інженерії, управління проєктами, управління конфігурацією тощо.

Експертні методи, які певною мірою дають змогу вирішувати поставлені задачі, доцільні для вирішення багатокритеріальних задач, що зазвичай викликають значні когнітивні навантаження. Однією з таких, зокрема, є задача ранжування вимог. Вважається, що для ухвалення рішень у такому середовищі в сучасних умовах цифровізації перспективним є забезпечення підтримки рішень на основі вичерпного подання інформаційної моделі предметної області та опрацювання за допомогою інформаційних технологій кількісних оцінок альтернатив. Також важливою можливістю сучасних технологій є забезпечення візуалізації процесів, пов'язаних з ухваленням рішень [25]. Ці задачі якнайкраще розв'язуються із застосуванням засобів ШІ.

Отже, штучний інтелект більше не є футуристичною концепцією. Сьогодні це множина щоденних інструментів у наборі засобів сучасних розробників ПЗ. Починаючи від GitHub Copilot, OpenAI Codex до автоматизованих фреймворків ці інструменти призначені для підвищення продуктивності розробника, гарантування найвищої якості програмного забезпечення та скорочення часу виведення продукту на ринок. Тематичні дослідження відомих технологічних компаній демонструють суттєві покращення ефективності та надійності в результаті використання інструментів на базі ШІ, що підвищує впевненість у розробці інтелектуальних, масштабованих та адаптивних програмних систем [26].

Таким чином, загальну онтологію AI-ландшафту програмної інженерії можна представити рис. 1.

Картографія застосування AI в програмній інженерії

SWEBOK організовує знання програмної інженерії у вигляді галузей знань (Knowledge Areas), кожна з яких відповідає певному типу процесів або діяльності. Спираючись на цю структуру, можна застосувати картографічний підхід.

На основі структури SWEBOK можна сформувати карту застосування AI в програмній інженерії, яка відображає відповідність між галузями знань, процесами, AI-технологіями, інструментами. Такий підхід дозволяє систематизувати використання AI, уникнути фрагментарного впровадження технологій, а також оцінювати рівень інтелектуалізації процесів розробки.

Системно карту застосування штучного інтелекту в програмній інженерії доцільно будувати на основі структури галузей знань, визначених у SWEBOK Guide як багаторівневу модель, що складається з таких рівнів як сфери діяльності, типи задач, технології AI та практичні інструменти (табл. 1).

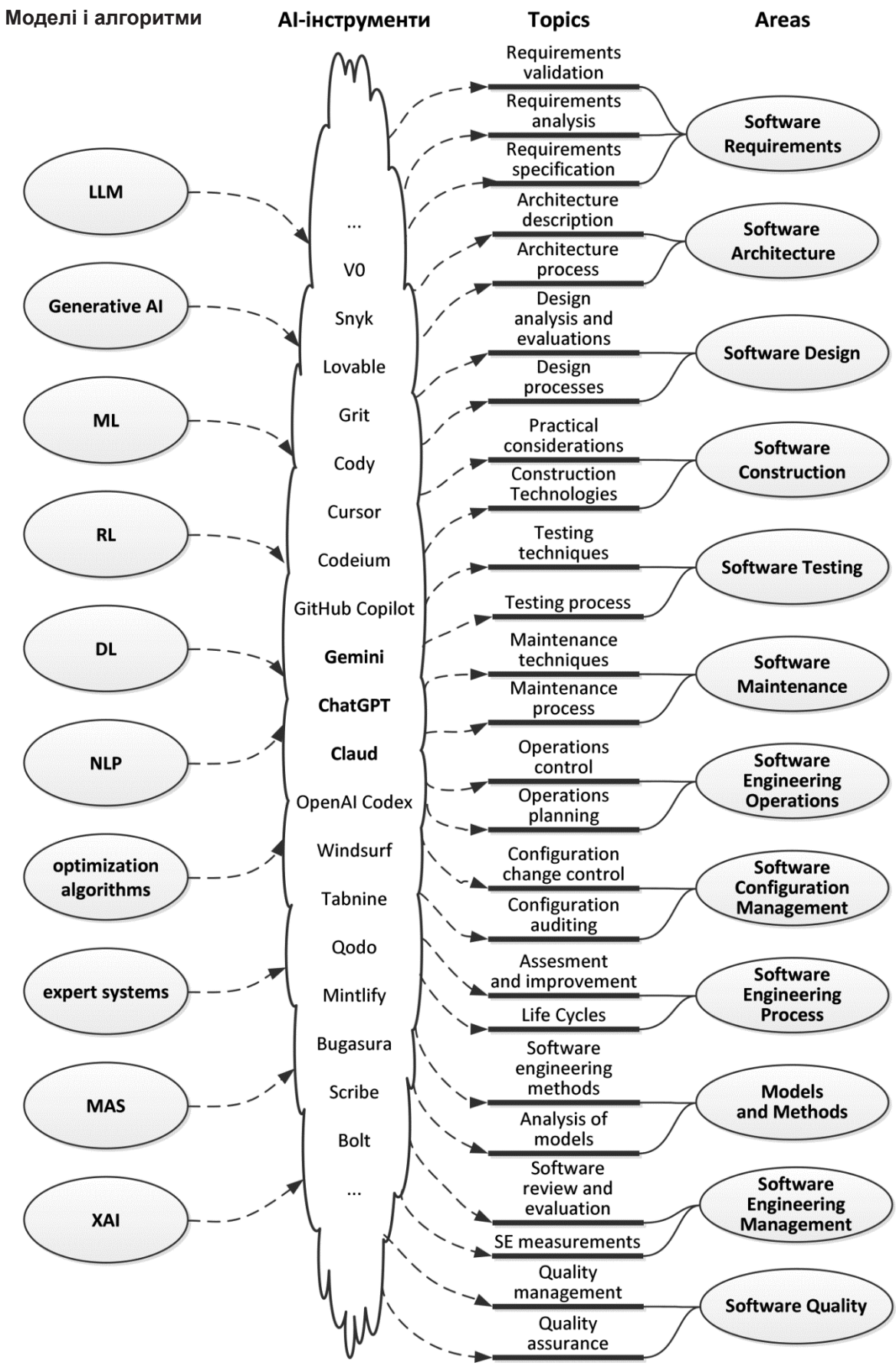

Рис. 1. Онтологія AI-ландшафту програмної інженерії

Таблиця 1.

Карта застосування AI в програмній інженерії

№ з/п	Область знань програмної інженерії	Основні задачі	AI-технології	Приклади інструментів
1.	Інженерія вимог	Аналіз вимог, виявлення неоднозначностей, трасування	NLP, класифікація тексту, knowledge graphs	ChatGPT-подібні системи, вимогові аналізатори
2.	Архітектура і проєктування ПЗ	Архітектурні рішення, моделювання, вибір патернів	Рекомендаційні системи, генеративні моделі	AI-архітектурні помічники
3.	Конструювання ПЗ	Програмування, рефакторинг, генерація коду	Великі мовні моделі, deep learning	GitHub Copilot
4.	Тестування ПЗ	Генерація тестів, виявлення дефектів	ML-класифікація, генеративні моделі	SonarQube
5.	Супровід ПЗ	Локалізація дефектів, модернізація систем	Defect prediction, code analysis	AI bug prediction systems
6.	Управління конфігурацією	контроль версій, merge-конфлікти	аналіз репозиторіїв, графові моделі	Git
7.	Базовий процес ПІ	оцінювання та вдосконалення процесів	process mining, ML-аналітика	AI-process analytics
8.	Методи інженерії	евристичні, формальні, прототипні методи	метаевристики, symbolic AI	AI-оптимізаційні системи
9.	Управління програмною інженерією та проєктами	прогнозування строків, ризик-аналіз	predictive analytics, ML-прогнозування	Jira

У цій карті AI виконує чотири ключові функції: асистент розробника (допомагає створювати код, моделі та документацію), аналітик системи (аналізує дані про проєкт і кодову базу), прогностичний модуль (прогнозує дефекти, ризики та строки), а також оркестратор процесів (інтегрує інструменти DevOps і оптимізує workflow). Запропонована карта демонструє три важливі принципи інтеграції AI в програмну інженерію: технологічний рівень – типи AI-алгоритмів; процесний рівень – галузі знань програмної інженерії; інструментальний рівень – конкретні програмні засоби. Це дозволяє формувати цілісну модель AI-орієнтованої програмної інженерії (AI-Driven Software Engineering).

Модель зрілості AI-Driven Softare Engineering

Парадигма програмної інженерії запровадила мислення на основі моделі зрілості, яка надає компаніям дорожню карту для покращення їхньої продуктивності з обраних точок зору, відомих як ключові можливості. Capability Maturity Model Integration (CMMI) – це комплексна модель продуктивності та зрілості, що представляє набір моделей (методологій) вдосконалення процесів в організаціях різних розмірів і видів діяльності. CMMI містить набір рекомендацій у вигляді практик, реалізація яких дозволяє досягти цілей, необхідних для повної реалізації сфер діяльності.

На основі цієї методології Інститутом програмної інженерії (Software Engineering Institute – SEI) запропоновано використовувати модель зрілості можливостей людей (People CMM). Це інструмент, який допомагає успішно вирішувати критичні проблеми з людьми в організації, яка, зокрема, працює над розробкою ПЗ. P-CMM використовує систему процесів надзвичайно успішної моделі зрілості можливостей для програмного забезпечення як основу для моделі найкращих практик для управління та розвитку робочої сили організації. Базовий підхід моделі P-CMM – це усвідомлення цінності кожного працівника як особистості та необхідності його подальшого розвитку.

На основі цих концепцій дослідники і фахівці пропонують модель зрілості для впровадження штучного інтелекту в галузі програмної інженерії (AI-Driven Software Engineering CMM), щоб допомогти організаціям оцінити та вдосконалити свої зусилля з інтеграції ШІ та ПІ. Ґрунтуючись на екосистемі багатьох зацікавлених сторін, що включає академічні кола, дослідників, гравців галузі та урядові установи, ця модель визначає ключові етапи впровадження ШІ та умови, необхідні для прогресу в цій сфері.

Переосмислюючи розрив між інноваціями та практикою через цю структуровану призму, модель пропонує практичні висновки для узгодження результатів досліджень з промисловою готовністю та прискорення ефективного впровадження штучного інтелекту в середовищах програмної інженерії [27]. Модель об'єднує наукові та практичні висновки і створює рекомендації для розробників програмного забезпечення та описує структуру цієї моделі зрілості для оцінки та покращення використання розробки із застосуванням ШІ, зокрема LLM [28].

Модель зрілості AI-DSE CMM передбачає п'ять рівнів зрілості (табл. 2). Таким чином ця модель не лише описує ступінь інтеграції технологій штучного інтелекту в процеси розробки програмного забезпечення, а й. дозволяє оцінити, наскільки організація використовує ШІ для підтримки життєвого циклу ПЗ, та визначити напрямки подальшого розвитку. Концептуально модель спирається на структуру галузей знань, визначених у SWEBOK Guide, а також на підходи до оцінювання зрілості процесів, подібні до CMMI.

Узагальнено структуру моделі зрілості AI-Driven Software Engineering можна представити рис. 2.

Оцінювання зрілості AI-Driven Software Engineering може проводитися за такими критеріями та параметрами:

1) інструментальна інтеграція – рівень використання AI-інструментів;

2) процесна інтеграція – ступінь використання AI у SDLC;

3) дані та аналітика – доступність історичних даних проєктів;

4) автоматизація – рівень автоматизації процесів розробки;

5) організаційна готовність – компетентності персоналу, наявність стратегії впровадження AI.

Практичне значення використання моделі зрілості AI-Driven Software Engineering полягає у тому, що вона дозволяє оцінювати рівень інтелектуалізації процесів розробки, планувати стратегію впровадження AI, порівнювати різні організації виробництва, а також визначати напрямки ефективного розвитку програмної інженерії.

Висновки

Для визначення та класифікації інструментів штучного інтелекту в програмній інженерії може використовуватися як концептуальна рамка структуризація знань, запропонована у SWEBOK. Вона дозволяє побудувати системну модель інтеграції AI в усі сфери діяльності, що формують життєвий цикл програмного забезпечення.

Проведено огляд можливостей і обмежень застосування основних моделей штучного інтелекту, таких як машинне навчання (ML), навчання з підкріпленням (RL) та глибинне навчання (DL), обробка природної мови (NLP), великі мовні моделі (LLM), генеративний ШІ (Generative AI) та ін.

На цій основі визначено картографію застосування AI в програмній інженерії та побудовано загальну онтологію AI-ландшафту програмної інженерії. Запропонована картографія демонструє три важливі принципи інтеграції AI в програмну інженерію: технологічний рівень, процесний рівень та інструментальний рівень. Це дозволяє сформувати цілісну модель AI-орієнтованої програмної інженерії.

Розглянуто структуру моделі зрілості AI-Driven Softare Engineering та її практичні переваги застосування, що дозволяють оцінити, наскільки організація використовує ШІ для підтримки життєвого циклу ПЗ.

Таблиця 2.

Рівні моделі зрілості AI-Driven Software Engineering

Рівень	Характеристика рівня	Типові приклади	Результат
1. Початковий (Ad-hoc AI Use)	AI застосовується епізодично. Інструменти використовуються окремими розробниками.. Відсутня загальна стратегія використання AI	Використання AI-асистентів програмування, окремі експерименти з генерацією коду, застосування AI для аналізу окремих задач	AI використовується як індивідуальний інструмент, але не інтегрований у процес розробки ПЗ
2. Інструментальна інтеграція (AI-Assisted Development)	AI інтегрується в інструментальне середовище розробки. З'являється підтримка окремих процесів SDLC. Використовуються AI-DevOps інструменти.	Автоматична генерація коду, AI-аналіз якості коду, генерація тестів, інтелектуальний аналіз вимог	AI стає частиною інструментальної екосистеми розробки ПЗ
3. Процесна інтеграція (AI-Enabled Processes)	AI використовується в ключових процесах життєвого циклу ПЗ. Дані проєктів системно аналізуються. Ухвалення рішень підтримується аналітичними моделями	Прогнозування дефектів, аналіз продуктивності, команд, автоматизоване тестування, оптимізація CI/CD процесів.	AI стає невід'ємною частиною процесів програмної інженерії
4. Інтелектуальна організація виробництва (AI-Optimized Engineering)	AI використовується для оптимізації процесів розробки. Впроваджено аналітику процесів і process mining. Використовується прогнозна аналітика в управлінні проєктами	AI-аналіз ризиків проєкту, оптимізація архітектурних рішень, інтелектуальна підтримка управління конфігураціями, автоматичний аналіз технічного боргу	Виробництво ПЗ переходить до data-driven управління програмною інженерією
5. Автономна програмна інженерія (Autonomous AI-Driven Engineering)	AI виконує значну частину інженерних задач. Системи здатні адаптувати процеси розробки. Використовується самонавчальна інфраструктура	Автоматичне створення архітектури систем, генерація та оптимізація коду, автономне тестування та верифікація, self-adaptive software systems	AI виступає активним учасником інженерної діяльності, а не лише інструментом

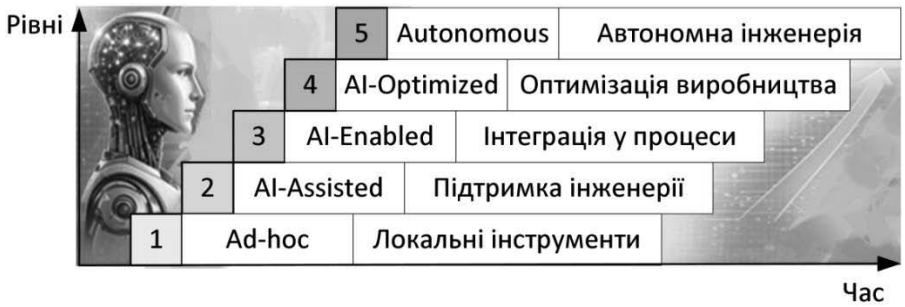

Рис. 2. Модель зрілості AI-Driven Software Engineering

References

1. P. Kokol, The Use of AI in Software Engineering: A Synthetic Knowledge Synthesis of the Recent Research Literature, in: Information, 2024, 15, 354, doi: 10.3390/info15060354
2. H. Aditya Pai, K. R. Sharath, K. Babalad, et all, Integrating AI into Software Engineering: A Critical Review and Future Directions, 2025 International Conference on Intelligent Control, Computing and Communications (IC3 2025), in: Proceedings of the Conference, 2025, pp. 20–25. doi: 10.1109/IC363308.2025.10957449
3. M. J. Hossain Faruk, H. Pournaghshband, H. Shahriar, AI-Oriented Software Engineering (AIOSE): Challenges, Opportunities, and New Directions, in: Lecture Notes in Networks and Systems, 2023, 576 LNNS, pp. 3–19. doi: 10.1007/978-3-031-20322-0_1
4. V. Terragni, A. Vella, P. Roop, K. Blincoe, The Future of AI-Driven Software Engineering, in: ACM Transactions on Software Engineering and Methodology, 2025, 34, 5, art. no. 120. doi: 10.1145/3715003
5. O. Nesterenko, P. Yatsuk, AI (not) against AI, in: Environmental Safety and Natural Resources, 2025, 56, 4, pp. 134–153. doi: 10.32347/2411-4049.2025.4.134-153. [in Ukrainian]
6. S. Cao, X. Sun, R. Widyasari, et all, A Systematic Literature Review on Explainability for ML/DL-based Software Engineering, in: ACM Computing Surveys, 2025, 58, 4, art. no. 95, pp. 1–34, doi: 10.1145/3763230.
7. R. Ulfsnes, N. B. Moe, J. Emmerhoff, et all, Responsible AI in Agile Software Engineering – An Industry Perspective. in: Lecture Notes in Business Information Processing, 2025, 524, pp. 33–41. doi: 10.1007/978-3-031-72781-8_4.
8. Q. U. Ain, M. Haq, A. A. A. Jilani, K. Sohail, A comparison between traditional software engineering practices and AI-driven methodologies, in: Generative AI in Software Engineering, 2025, pp. 57–92. doi: 10.4018/979-8-3373-0370-3.ch002.
9. M. Alenezi, M. Akour, AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions, in: Applied Sciences, 2025, 15, 3, art. no. 1344, doi: 10.3390/app15031344.
10. A. Mohammadkhani, N. Bommi, M. Daboussi, et all, A Systematic Literature Review of Explainable AI for Software Engineering, 2023, arXiv.2302.06065, doi: 10.48550/arXiv.2302.06065.
11. P. S. N. Mindom, A. Nikanjam, F. Khomh, Harnessing pre-trained generalist agents for software engineering tasks, in: Empirical Software Engineering, 2024, 30, 1, art. no. 39. doi: 10.1007/s10664-024-10597-8.
12. U. K. Durrani, M. Akpinar, M. F. Adak, et all, A Decade of Progress: A Systematic Literature Review on the Integration of AI in Software Engineering Phases and Activities (2013-2023), in: IEEE ACCESS, 2024, 12, pp. 171185–171204. doi: 10.1109/ACCESS.2024.3488904.
13. V. T. Wivestad, A. M. Barbala, Attitudes Toward LLM Use Among Software Engineering Researchers: Results from a Two-Phase Survey Study, in: Companion proceedings of the 33RD ACM International conference on the foundations of software engineering (FSE), 2025, pp. 1531–1535. doi: 10.1145/3696630.3731671
14. Z. H. Lin, W. Ma, T. Lin, et all, Open Source AI-based SE Tools: Opportunities and Challenges of Collaborative Software Learning, in: ACM transactions on software engineering and methodology, 2025, 34, 5, pp. 1535–1560. doi: 10.1145/3708529.
15. A. Alami, N. Ernst, Human and Machine: How Software Engineers Perceive and Engage with AI-Assisted Code Reviews Compared to Their Peers, 2025 IEEE/ACM 18th International conference on cooperative and human aspects of software engineering (CHASE), in: Proceedings of the Conference, 2025, pp. 63–74. doi: 10.1109/CHASE66643.2025.00016.
16. K. Ronanki, Facilitating Trustworthy Human-Agent Collaboration in LLM-based Multi-Agent System oriented Software Engineering, in: Companion proceedings of the 33rd ACM International conference on the foundations of software engineering (FSE 2025), arXiv:2505.0425. doi: 10.48550/arXiv.2505.04251.
17. B. Trinkenreich, F. Calefato, G. Hanssen, et all, Get on the Train or be Left on the Station: Using LLMs for Software Engineering Research, in: Companion proceedings of the 33rd ACM international conference on the foundations of software engineering (FSE COMPANION 2025), 2025, pp. 1503–1507. doi: 10.1145/3696630.3731666.
18. O. Nesterenko, Y. Selin, The Teams Information Model for Software Engineering Management, in: Proceedings of 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT), 2021, 1, pp. 341–344, doi: 10.1109/CSIT52700.2021.9648737.

19. I. Rauf, H. Sharp, T. Lopez, M. Wermelinger, Human-Machine Teaming and Team Effectiveness in AI tools for Software Engineering, 2025 Conference on Cooperative and Human Aspects of Software Engineering (CHASE), in: Proceedings of the Conference, 2025, pp. 75–80. doi: 10.1109/CHASE66643.2025.00017/
20. A. Fan, B. Gokkaya, M. Harman, et all, Large Language Models for Software Engineering: Survey and Open Problems, 2023 IEEE/ACM International conference on software engineering: future of software engineering (ICSE-FOSE), in: Proceedings of the Conference, 2023, pp. 31–53. doi: 10.1109/ICSE-FoSE59343.2023.00008.
21. A. Striapunin., V. Kharchenko, Using AI tools in requirements engineering: opportunity analysis and chatbot for validation, in: Aerospace Technic and Technology, 2024, 2, pp. 91–101. doi: 10.32620/aktt.2024.2.10. [in Ukrainian]
22. D. Nikitin, V. Golian, Methods for integrating artificial intelligence and knowledge engineering into automaton-based real-time software systems, in: Herald of Khmelnytsskyi National University, 2025, 2, pp. 285–292. doi: 10.31891/2307-5732-2025-349-42.
23. M. H. Tanzil, J. Y. Khan, G. Uddin, ChatGPT Incorrectness Detection in Software Reviews, in: Proceedings of the IEEE/ACM 46th International conference on software engineering (ICSE 2024), arXiv:2403.16347. doi: 10.1145/3597503.3639194.
24. M. Fischer, C. Lanquillon, Evaluation of Generative AI-Assisted Software Design and Engineering: A User-Centered Approach, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2024, 14734 LNAI, pp. 31–47. doi: 10.1007/978-3-031-60606-9_3.
25. O. Nesterenko, O. Trofymchuk, Decision Support for Requirements Prioritization in Software Engineering, in: CEUR Workshop Proceedings, 2024, 3806, 50–61. URL: https://ceur-ws.org/Vol-3806/S_31_Nesterenko_Trofymchuk.pdf.
26. K. Shantha Kumari, B. Sundaravadivazhagan, V. Indumathy, D. Maladhy, Detailing AI techniques and tools for software engineering acceleration and automation, in: Advances in Computers, 2026, 141, pp. 183– 209. doi: 10.1016/bs.adcom.2025.07.007.
27. H. Lhazmir, S. Samhale, K. Louzaoui, K. Benlhachmi, The Uneven Journey of AI in Software Testing: A Maturity Model for Industry Adoption, 8th IEEE International Congress on Information Science and Technology (CIST 2025), in: Proceedings of the Conference, 2025, pp. 132–137. doi: 10.1109/CiSt65886.2025.11224099.
28. S. L. France, Navigating software development in the ChatGPT and GitHub Copilot era, in: Business Horizons, 2024, 67, 5, pp. 649–661. doi: 10.1016/j.bushor.2024.05.009

Дата першого надходження до видання: 27.03.2026
Внутрішня рецензія отримана: 13.04.2026
Зовнішня рецензія отримана: 22.04.2026
Дата прийняття статті до друку: 05.06.2026
Дата публікації: 29.06.2026

Про авторів:

Нестеренко Олександр Васильович,
доктор технічних наук, професор,
завідувач кафедри
Nesterenko Olexandr,
Ph.D. (doctor, technical sciences),
professor, head of department
http://orcid.org/0000-0001-5329-889X .

Федоров Володимир Володимирович,
кандидат фізико-математичних наук,
доцент
Fedorov Volodymyr,
Ph.D. (physical and mathematical sciences),
associate professor
https://orcid.org/0009-0004-2901-3646.

Яцук Петро Петрович,
кандидат технічних наук, доцент
Yatsuk Petro,
Ph.D. (technical sciences), associate professor
https://orcid.org/ 0009-0002-7124-4849.

Місце роботи авторів:

Міжнародний європейський університет,
Кафедра інформаційних технологій
International European University,
Department of Informational Technologies
Tel.: +380 97 757 27 96
E-mail: oleksandr_nesterenko@ieu.edu.ua
https://business.ieu.edu.ua/kafedry/kafedra-informatsiinykh-tekhnolohii