УДК 004.8:378.147; 378.147:004.42; 004.41
DOI https://doi.org/10.36994/2788-5518-2025-01-09-09

ІНТЕГРОВАНІ СЕРЕДОВИЩА РОЗРОБКИ З ПІДТРИМКОЮ ШТУЧНОГО ІНТЕЛЕКТУ: НОВІ МОЖЛИВОСТІ ДЛЯ ВИВЧЕННЯ ПРОГРАМУВАННЯ СТУДЕНТАМИ

Богун Р. І., *к.ф.-м.н., Відкритий міжнародний університет розвитку людини «Україна», Київ, Україна. https://orcid.org/0009-0002-8519-2234, bohun.roma@gmail.com*

Одрібець Н. В., *к.ф.-м.н., доц., Відкритий міжнародний університет розвитку людини «Україна», Київ, Україна. https://orcid.org/0009-0008-0176-1211, sosn@ukr.net*

Веденєєва О. А., *Відкритий міжнародний університет розвитку людини, Київ «Україна», Україна. https://orcid.org/0009-0006-0941-165X, vedenieieva@cyber.uu.edu.ua*

Анотація. *У статті проведено комплексний аналіз інтеграції середовищ розробки з підтримкою штучного інтелекту (ШІ) в процес вивчення програмування студентами технічних спеціальностей. Дослідження фокусується на тому, як інструменти GitHub Copilot, Cursor, Windsurf та інші трансформують традиційні підходи до навчання, пропонуючи нові можливості для персоналізації й підвищення ефективності. Наведено детальний порівняльний аналіз ключових платформ за такими критеріями, як базові великі мовні моделі (LLM), підтримка мов програмування, інтеграція з різними IDE й, що особливо важливо для освітнього середовища, умови надання освітніх ліцензій. При цьому окремо наголошується на практичних аспектах отримання пільгового доступу: складнощах валідації статусу студента, можливих регіональних обмеженнях і змінності умов. Розглянуто весь спектр режимів взаємодії із ШІ: від базового автодоповнення коду (Code Completion) і діалогового режиму (Chat), що перетворює IDE на інтерактивного тьютора, до просунутої генерації (Generate) й агентного режиму (Agent), здатного виконувати комплексні багатоетапні завдання. Окрему увагу приділено розширенню функціонала ШІ за межі IDE – в інструменти командного рядка (CLI), що відкриває нові горизонти для автоматизації та вивчення системного адміністрування. На основі цього аналізу запропоновано низку інноваційних навчальних завдань, спрямованих не на просте копіювання коду, а на розвиток критичного мислення, навичок пошуку й виправлення помилок та аналізу якості згенерованих рішень. Фінальна частина статті присвячена глибокому аналізу ризиків, зокрема надмірній залежності від інструментів, що може призвести до атрофії фундаментальних навичок, і викликам для академічної доброчесності. Запропоновано конкретні педагогічні стратегії для мінімізації цих загроз, що включають зміну парадигми оцінювання, упровадження усних захистів проєктів і формування в студентів культури відповідального й етичного використання ШІ.*

Ключові слова: *генеративний штучний інтелект, інтегроване середовище розробки, навчання програмування, програмування, GitHub Copilot, академічна доброчесність, генерація коду, командний рядок, великі мовні моделі, освітній процес.*

AI-POWERED INTEGRATED DEVELOPMENT ENVIRONMENTS: NEW OPPORTUNITIES FOR STUDENT PROGRAMMING EDUCATION

Roman Bohun, *Ph.D., Open International University of Human Development "Ukraine", Kyiv, Ukraine. https://orcid.org/0009-0002-8519-2234, bohun.roma@gmail.com*

Nataliia Odribets, *Ph.D., Ass. Prof., Open International University of Human Development "Ukraine", Kyiv, Ukraine. https://orcid.org/0009-0008-0176-1211, sosn@ukr.net*

Olga Vedenieieva, *Open International University of Human Development "Ukraine", Kyiv, Ukraine. https://orcid.org/0009-0006-0941-165X, vedenieieva@cyber.uu.edu.ua*

Abstract. *The article provides a comprehensive analysis of the integration of AI-powered development environments into the process of learning programming for students in technical fields. The research focuses on how tools like GitHub Copilot, Cursor, Windsurf, and others are transforming traditional teaching approaches by offering new opportunities for personalization and efficiency. A detailed comparative analysis of key platforms is presented based on criteria such as their underlying large language models (LLMs), programming language support, integration with various IDEs, and, crucially for the educational context, the terms of educational licenses. Special emphasis is placed on the practical aspects of obtaining preferential access: the complexities of student status validation, potential regional restrictions, and the changing nature of the terms. The full spectrum of AI interaction modes is examined: from basic code*

completion and chat mode, which turns the IDE into an interactive tutor, to advanced generation and agent mode, capable of performing complex multi-step tasks. Special attention is given to the extension of AI functionality beyond the IDE into command-line interface (CLI) tools, opening new horizons for automation and learning system administration. Based on this analysis, a series of innovative educational assignments are proposed, aimed not at simple code copying but at developing critical thinking, debugging skills, and the ability to analyze the quality of generated solutions. The final part of the article is devoted to a deep analysis of the risks, particularly over-reliance on these tools, which can lead to the atrophy of fundamental skills, and the challenges to academic integrity. Specific pedagogical strategies are proposed to minimize these threats, including shifting assessment paradigms, implementing oral project defenses, and fostering a culture of responsible and ethical AI use among students.

Key words: *generative artificial intelligence; integrated development environment; learning to program; programming; GitHub Copilot; academic integrity; code generation; command-line interface; large language models; educational process.*

Вступ

Інтегровані середовища розробки (далі – IDEs) з підтримкою штучного інтелекту (далі – ШІ), зокрема GitHub Copilot, Cursor, Windsurf тощо, відкривають нові можливості для персоналізованого навчання програмування, що підтверджується результатами емпіричних досліджень у різних освітніх контекстах [1; 2; 3]. Практичне використання ШІ як тьютора або асистента демонструє зростання якості студентських проєктів, підвищення продуктивності та зменшення когнітивного навантаження [4; 5]. Водночас дослідники звертають увагу на необхідність критичного осмислення ролі ШІ в освітньому процесі, зокрема щодо формування довіри до результатів генерації коду й збереження мотивації до самостійного мислення [6; 1]. Це вимагає адаптації навчальних стратегій, які поєднують переваги автоматизації з розвитком алгоритмічного мислення й етичної відповідальності.

Згідно з аналітичними звітами, кількість користувачів ШІ-помічників зростає експоненційно, а компанії-розробники інвестують рекордні ресурси в їхній розвиток. За словами генерального директора Microsoft Сатьї Наделли, станом на 2025 рік до 30% коду в компанії вже генерується за допомогою ШІ [7]. GitHub Copilot, наприклад, уже має понад 15 мільйонів активних користувачів, а інші інструменти демонструють стрімке зростання популярності серед студентів і молодих фахівців. Цей тренд відображається й у практиці працевлаштування: усе більше роботодавців включають питання про досвід використання ШІ-середовищ у технічні співбесіди, оцінюючи здатність кандидатів ефективно працювати з інтелектуальними інструментами. IDE з підтримкою ШІ дедалі активніше використовуються не лише в професійній розробці [8], а й у навчальному процесі. Ці інструменти виступають як персоналізовані цифрові асистенти, що інтегруються в навчальний процес і суттєво полегшують засвоєння програмування. Особливо ефективними ці інструменти є на початкових етапах навчання, коли студенти стикаються з типовими труднощами – синтаксичними помилками, нерозумінням логіки алгоритмів, браком прикладів.

Виконання досліджень

IDEs з підтримкою ШІ забезпечують студентам доступ до інтелектуальної підтримки в реальному часі: автозаповнення коду, генерація функцій, пояснення логіки, пошук і виправлення помилок і рефакторинг. Завдяки цьому здобувачі освіти можуть швидше засвоювати нові мови програмування, експериментувати з кодом, учитися на помилках і водночас розвивати критичне мислення. За результатами досліджень, використання ШІ-помічників дає змогу зменшити час на виконання завдань, а також підвищити якість коду й упевненість студентів у власних силах [1; 2; 4].

Порівняльний аналіз ШІ-інструментів для розробки

Для об'єктивної оцінки можливостей сучасних ШІ-інструментів проведено порівняльний аналіз найпопулярніших рішень на ринку. Критеріями порівняння слугували доступність для закладів освіти, підтримка мов програмування й IDE, функціональні можливості й особливості використання різних великих мовних моделей (LLM).

Критерій	GitHub Copilot	Cursor	Windsurf (раніше Codeium)	Amazon CodeWhisperer	Tabnine
Основна LLM	OpenAI GPT-4.1/*	auto	OpenAI GPT-4.1/*, SWE-1*	Власна модель	Власна модель
Використання інших LLM	Так	Так	Так	Так (платф. Amazon Bedrock)	Так
Приблизна кількість користувачів	> 15 млн.	> 1 млн.	> 1 млн.	> 1 млн.	> 1 млн.
Підтримувані IDE	VS Code, JetBrains, ...	Власна IDE (форк VS Code)	VS Code, JetBrains, ...	VS Code, JetBrains, others	VS Code, JetBrains...
Підтримувані мови	Усі популярні мови (Python, JS, TS, Java, C++, Go, Ruby тощо)				

Безкоштовний план (з лімітами)	Так	Так	Так	Так	Ні (2025)
Ліцензія для студентів/ викладачів	Безкоштовно (Copilot Pro)/+	Безкоштовно 1 рік (Cursor Pro), знижки для студентів/-	Знижка 50% на Pro план/-	Безкоштовно (Individual tier)/+	Н/Д
Окрема IDE	Так	Ні	Так	Ні	Ні
Плагіни до IDE	Так	Ні	Так	Так	Ні
Основні можливості	Автодоповнення коду, чат, генерація функцій, пояснення, рефакторинг, CLI-помічник, агент	«AI-first» IDE, чат з кодовою базою, генерація «з нуля», авто-дебаг, міграція коду, агент	Швидке автодоповнення, чат в IDE, генерація коду за коментарями, агент	Автодоповнення, сканування безпеки, генерація коду, робота з AWS SDK	Контекстне автодоповнення, працює локально (підвищена приватність), інші
Переваги	Найкраща інтеграція, величезна спільнота, безкоштовно для освіти	Потужні функції для роботи з усім проєктом	Дуже широка підтримка IDE, швидкість	Глибока інтеграція з екосистемою AWS, акцент на безпеці	Висока швидкість, можливість роботи офлайн
Недоліки		Платні функції для повного доступу		Найкраще працює з AWS, менш універсальний	Відсутній безкоштовний план
Наявність підтримки	Документація, спільнота, підтримка				

Примітки. Варто зазначити, що ринок ШІ-інструментів для розробки активно розвивається, тому деякі дані, наведені в таблиці, можуть швидко змінюватися й потребують уточнення на офіційних ресурсах.

Варто зазначити, що отримання безкоштовного або пільгового доступу до цих інструментів за освітніми програмами зазвичай вимагає валідації статусу студента чи викладача. Цей процес може займати певний час і потребує надання відповідних документів. Крім того, умови таких програм лояльності можуть періодично змінюватися, мати часові обмеження або географічні рамки, розповсюджуючись, наприклад, лише на певні регіони, як-от США, відрізнятися для студентів і викладачів. Тому перед початком використання рекомендується ретельно ознайомитися з актуальними правилами на офіційних сайтах розробників.

Сучасні ШІ-помічники пропонують кілька фундаментальних режимів взаємодії, які можна адаптувати для освітніх цілей. Варто зазначити, що, хоча більшість сучасних інструментів підтримують перелічені режими, їхня реалізація, функціонал і рівень «агентивності» можуть суттєво відрізнятися, пропонуючи різну кількість і глибину можливостей. Загалом можна виділити такіі режими:

1. Режим асистента (Code Completion) – найпоширеніший режим, де ШІ пропонує варіанти завершення рядка або цілого блоку коду в реальному часі. Для студентів це миттєвий спосіб уникнути синтаксичних помилок і швидше писати шаблонний код.

2. Режим діалогу (Chat/Ask) – інтегрований чат, що дає змогу ставити запитання мовою, близькою до природної. Студент може запитати: «Поясни, як працює цей фрагмент коду», «Яка різниця між let і const у JavaScript?» або «Запропонуй кращий спосіб реалізації цього алгоритму». Це перетворює IDE на інтерактивного тьютора.

3. Режим генерації (Generate/Edit) – студент описує завдання (наприклад, «Створи функцію, що приймає URL і повертає JSON-дані»), а ШІ генерує повний код. Цей режим також використовується для рефакторингу (покращення наявного коду) або виправлення помилок (дебагінгу).

4. Режим агента (Agent) – найпросунутіший режим, де ШІ може виконувати складні, багатоетапні завдання. Наприклад, студент може поставити завдання: «Проаналізуй весь проєкт, знайди потенційні вразливості безпеки, запропонуй виправлення та напиши для них unit-тести». ШІ-агент самостійно розбиває завдання на кроки й виконує їх, взаємодіючи з файловою системою проєкту.

ШІ-помічники активно використовуються не лише в IDE, а і як інструменти командного рядка (CLI), що дає змогу отримувати інтелектуальну підтримку безпосередньо в терміналі для керування версіями, виконання скриптів та інших завдань. Основні можливості таких інструментів включають генерацію й пояснення системних команд, створення скриптів автоматизації та допомогу в роботі з Git. Серед ключових інструментів виділяються офіційні GitHub Copilot CLI й нещодавно представлений Google Gemini CLI, які перетворюють термінал на інтерактивне навчальне середовище. Anthropic

також надає офіційну документацію для використання Claude з командного рядка, а для моделей OpenAI існують численні інструменти від спільноти, що розвиває в студентів навички роботи з API й автоматизації.

Дослідження показують, що ефективність ШІ-інструментів залежить від складності завдання та LLM, що лежить в їх основі. Наприклад, у стандартних тестах на генерацію коду, як-от HumanEval, моделі GPT-4 (Copilot, Cursor) і Claude 3 Opus (Cursor) часто демонструють найвищі результати. Водночас Amazon CodeWhisperer може бути ефективнішим у завданнях, пов'язаних з інфраструктурою AWS.

На основі цих можливостей можна сформулювати такі види навчальних робіт для студентів:

1. Критик ШІ: студент дає ШІ завдання згенерувати код для вирішення певної проблеми, а потім має проаналізувати отриманий результат: знайти помилки, неефективні місця, потенційні вразливості й запропонувати власне, покращене рішення.

Приклад. Попроси CodeWhisperer написати функцію для обробки завантажених файлів. Проаналізуй її на наявність вразливостей типу «Path Traversal».

Переваги: формує критичне мислення, учить не довіряти сліпо технологіям і розуміти глибинні аспекти проблеми.

2. Рефакторинг-челендж: викладач надає застарілий або «погано написаний» код. Завдання студента – за допомогою ШІ-інструментів провести його повний рефакторинг: покращити читабельність, оптимізувати продуктивність, додати коментарі та документацію.

Приклад. Використовуючи функцію «Edit» у Cursor, перетвори цей код на основі колбеків на сучасний синтаксис з async/await.

Переваги: навчає сучасних стандартів кодування й важливості підтримки кодової бази.

3. Дебагінг-квест: викладач навмисно вносить у код неочевидну помилку. Завдання студента – використовуючи можливості ШІ для налагодження (пояснення коду, пропозиції виправлень, аналіз помилок), знайти й виправити її.

Приклад. У цьому коді є помилка, пов'язана з асинхронними операціями, яка призводить до «race condition». Використовуй чат GitHub Copilot, щоб проаналізувати логіку та знайти причину проблеми.

Переваги: розвиває навички системного налагодження, учить використовувати ШІ як діагностичний інструмент.

4. TDD-спринт (Test-Driven Development): студент за допомогою ШІ спочатку генерує набір unit-тестів для ще не написаної функціональності, а потім пише код, який має задовольнити ці тести.

Приклад. Сформулюй для Tabnine запит на створення unit-тестів для функції калькулятора, що покривають додавання, віднімання, ділення на нуль і роботу з нечисловими даними.

Переваги: прищеплює культуру тестування та розробки через тестування (TDD).

5. API-дослідник: студентам дається завдання інтегрувати в проєкт новий, незнайомий сторонній API. Вони повинні використовувати ШІ-асистент для вивчення документації, генерації прикладів запитів і написання коду інтеграції.

Приклад. Використовуючи Cursor, інтегруй API для прогнозу погоди в цей додаток. Згенеруй код для запиту поточної погоди за назвою міста й відобрази результат.

Переваги: імітує поширене реальне завдання, навчає швидко освоювати й застосовувати нові технології.

6. Навігатор по коду: студенти отримують великий, незнайомий проєкт (наприклад, open-source бібліотеку) і за допомогою ШІ-чату повинні пояснити архітектуру, призначення ключових модулів і логіку роботи основних функцій.

Приклад. Використовуючи Copilot Chat, опиши, як реалізовано механізм маршрутизації в цьому веб-фреймворку.

Переваги: розвиває навички аналізу чужого коду, що є ключовим для роботи в команді.

7. Архітектор рішень: студенти отримують опис проблеми високого рівня й за допомогою ШІ повинні розробити архітектуру застосунку: обрати технології, спроєктувати схему бази даних, структурувати проєкт.

Приклад. Спроєктуй архітектуру для простої блог-платформи. Запитай у Claude поради щодо вибору між монолітною та мікросервісною архітектурою. Згенеруй базову структуру папок і файлів для обраного підходу.

Переваги: навчає системного проєктування високого рівня, спонукає думати про компроміси й архітектурні патерни.

Попри значні переваги, інтеграція ШІ в навчання несе суттєві ризики, які потребують детального аналізу та пропрацьованих стратегій мінімізації:

1. Надмірна залежність та «атрофія» навичок: студенти можуть утратити здатність самостійно вирішувати проблеми, оскільки звикають отримувати готові рішення від ШІ. Це може призвести до браку фундаментального розуміння алгоритмів і структур даних.

2. Галюцинації та неякісний код: ШІ-моделі іноді генерують код, який виглядає правдоподібно, але містить логічні помилки, вразливості безпеки або є неоптимальним. Студент без належного досвіду може не помітити цих недоліків.

3. Плагіат та академічна недоброчесність: межа між допомогою інструмента й плагіатом стає розмитою. Визначити, студент самостійно дійшов до рішення чи просто скопіював відповідь ШІ, стає надзвичайно складно для викладача.

4. Зниження креативності: постійне використання готових шаблонів може пригнічувати здатність студентів до пошуку нестандартних, інноваційних рішень.

Способи контролю й вирішення проблеми бездумного використання

Проблема «скопіював-вставив» є центральним викликом. Для її вирішення потрібен комплексний підхід, що поєднує технологічні, педагогічні й адміністративні заходи.

Запропоновані варіанти вирішення:

1. Зміна парадигми завдань: замість завдань, що мають одну правильну відповідь (наприклад, «напишіть функцію сортування»), варто пропонувати завдання відкритого типу, що вимагають аналізу й обґрунтування.

Приклад. Дослідіть три різні алгоритми сортування, згенеруйте їх реалізації за допомогою ШІ. Порівняйте їхню продуктивність на різних наборах даних. Обґрунтуйте, який алгоритм є найкращим для кожного випадку, і поясніть, чому згенерований ШІ код може бути неоптимальним.

2. Упровадження усної захисту робіт: студент повинен бути готовим пояснити кожний рядок свого коду, логіку прийнятих рішень та альтернативні шляхи вирішення проблеми. Це унеможливлює бездумне копіювання.

3. Використання інструментів для детекції ШІ-згенерованого коду: хоча такі інструменти не є ідеальними, вони можуть слугувати допоміжним сигналом для викладача, указуючи на роботи, що потребують ретельнішої перевірки.

4. Навчання «ШІ-грамотності»: включення в навчальні програми модулів, присвячених етичному й ефективному використанню ШІ. Студентів варто навчати, як правильно формулювати запити (prompt engineering), як критично оцінювати результати та як посилатися на використання ШІ-інструментів у своїх роботах, аналогічно до цитування наукових джерел.

5. Розроблення чітких інституційних політик: заклад вищої освіти має розробити й донести до студентів і викладачів чіткий кодекс академічної доброчесності в епоху ШІ, де буде визначено, що є допустимим використанням, а що вважається плагіатом.

Висновок

Отже, інтегровані середовища розробки з підтримкою ШІ є не лише інструментами автоматизації, а й потужними засобами розвитку цифрової грамотності, аналітичного мислення та професійної готовності студентів до викликів сучасної ІТ-індустрії.

Інтегровані середовища розробки з підтримкою ШІ створюють нові умови для ефективного, адаптивного й мотиваційного навчання програмування, сприяючи розвитку продуктивності, критичного мислення та цифрової грамотності студентів. На особливу увагу заслуговує GitHub Copilot завдяки своєму широкому функціоналу, підтримці багатьох мов програмування, інтеграції з популярними IDE та наявності безкоштовного доступу для викладачів і студентів через програму GitHub Education [9]. Подальші дослідження мають бути спрямовані на розроблення методик оцінювання навчального прогресу в умовах ШІ-підтримки, адаптації матеріалів навчальних курсів і вивчення впливу таких технологій на професійну готовність здобувачів освіти.

Література

1. Alanazi M., Soh B., Samra H., Li A.L.C. The Influence of Artificial Intelligence Tools on Learning Outcomes in Computer Programming: A Systematic Review and Meta-Analysis. *Computers*. 2025. Vol. 14, № 5. Article 185. DOI: https://doi.org/10.3390/computers14050185.

2. Gardella N., Pettit R., Riggs S.L. Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation. *Proceedings of the 2024 Conference on Innovation and Technology in Computer Science Education (ITiCSE 2024).* Vol. 1. Milan, Italy, 2024. P. 290–296. DOI: https://doi.org/10.1145/3649217.3653615.

3. Avramovic S., Avramovic I., Wojtusiak J. Exploring the Impact of GitHub Copilot on Health Informatics Education. *Applied Clinical Informatics*. 2024. Vol. 15, № 5. P. 1121–1129. DOI: https://doi.org/10.1055/a-2414-7790.

4. Timcenko O. Case Study: Using Artificial Intelligence as a Tutor for a Programming Course. *2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET)*. Paris, France, 2024. P. 1–4. DOI: https://doi.org/10.1109/ITHET61869.2024.10837624.

5. Mesaros G.O. Learning Web Development Using GitHub Copilot in and Outside Academia: A Blessing or a Curse? *Interdisciplinary Description of Complex Systems*, 2024. Vol. 22, № 3. P. 355–359. DOI: https://doi.org/10.7906/indecs.22.3.10.

6. Shah A., Chernova A., Tomson E., Porter L., Griswold W.G., Raj A.G.S. Students' Use of GitHub Copilot for Working with Large Code Bases. *Proceedings of the 56th ACM Technical Symposium on Computer Science Education (SIGCSE TS 2025).* Vol. 1. Pittsburgh, PA, 2025. P. 1050–1056. DOI: https://doi.org/10.1145/3641554.3701800.

7. Novet J., Vanian J. Satya Nadella says as much as 30% of Microsoft code is written by AI. URL: https://www.cnbc.com/2025/04/29/satya-nadella-says-as-much-as-30percent-of-microsoft-code-is-written-by-ai.html (date of access: 06.07.2025).

8. Подвишенна Д. Вайбкодинг: чому всі про нього говорять і коли він справді потрібен. URL: https://dou.ua/lenta/articles/what-is-vibecoding-developement/ (дата звернення: 06.07.2025).

9. Getting free access to Copilot Pro as a student, teacher, or maintainer. URL: https://docs.github.com/en/copilot/managing-copilot/managing-copilot-as-an-individual-subscriber/getting-started-with-copilot-on-your-personal-account/getting-free-access-to-copilot-pro-as-a-student-teacher-or-maintainer (date of access: 06.07.2025).

References

1. Alanazi, M., Soh, B., Samra, H., Li, A.L.C. (2025). The Influence of Artificial Intelligence Tools on Learning Outcomes in Computer Programming: A Systematic Review and Meta-Analysis. *Computers*, 14 (5), 185. https://doi.org/10.3390/computers14050185.

2. Gardella, N., Pettit, R., Riggs, S.L. (2024). Performance, Workload, Emotion, and Self-Efficacy of Novice Programmers Using AI Code Generation. *In Proceedings of the 2024 Conference on Innovation and Technology in Computer Science Education (ITiCSE 2024), Vol. 1* (pp. 290–296). Milan, Italy. https://doi.org/10.1145/3649217.3653615.

3. Avramovic, S., Avramovic, I., Wojtusiak, J. (2024). Exploring the Impact of GitHub Copilot on Health Informatics Education. *Applied Clinical Informatics,* 15 (5), 1121–1129. https://doi.org/10.1055/a-2414-7790.

4. Timcenko, O. (2024). Case Study: Using Artificial Intelligence as a Tutor for a Programming Course // *2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET)* (pp. 1–4). Paris, France. https://doi.org/10.1109/ITHET61869.2024.10837624.

5. Mesaros, G.O. (2024). Learning Web Development Using GitHub Copilot in and Outside Academia: A Blessing or a Curse? *Interdisciplinary Description of Complex Systems*, 22 (3), 355–359. https://doi.org/10.7906/indecs.22.3.10.

6. Shah, A., Chernova, A., Tomson, E., Porter, L., Griswold, W.G., Raj, A.G.S. (2025). Students' Use of GitHub Copilot for Working with Large Code Bases. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education, (SIGCSE TS 2025), Vol. 1* (pp. 1050–1056). Pittsburgh, PA. https://doi.org/10.1145/3641554.3701800.

7. Novet, J., Vanian, J. (2025). Satya Nadella says as much as 30% of Microsoft code is written by AI. *cnbc.com*. Retrieved from https://www.cnbc.com/2025/04/29/satya-nadella-says-as-much-as-30percent-of-microsoft-code-is-written-by-ai.html (date of access: 06.07.2025).

8. Podvyshenna, D. (2025). Vaibkodynh: chomu vsi pro noho hovoriat i koly vin spravdi potriben [Vibecoding: why everyone's talking about it and when it actually matters]. *dou.ua*. Retrieved from https://dou.ua/lenta/articles/what-is-vibecoding-developement/ (date of access: 06.07.2025) [in Ukrainian].

9. Getting free access to Copilot Pro as a student, teacher, or maintainer. *docs.github.com*. Retrieved from https://docs.github.com/en/copilot/managing-copilot/managing-copilot-as-an-individual-subscriber/getting-started-with-copilot-on-your-personal-account/getting-free-access-to-copilot-pro-as-a-student-teacher-or-maintainer (date of access: 06.07.2025).

Дата першого надходження рукопису до видання: 14.05.2025
Дата прийнятого до друку рукопису після рецензування: 12.06.2025
Дата публікації: 25.07.2025