Список використаних джерел:

1. Шелл Дж. Мистецтво геймдизайну : книга лінз. Київ : ArtHuss, 2021. 600 с.
2. Nystrom R. Game Programming Patterns. USA : Genever Benning, 2014. 354 p.
3. Unity Documentation. URL: https://docs.unity3d.com (дата звернення: 19.03.2026).
4. Godot Engine Documentation. URL: https://docs.godotengine.org (дата звернення: 19.03.2026).
5. React Native Documentation. URL: https://reactnative.dev (дата звернення: 19.03.2026).

УДК 004.8:004.415.2

ВИКОРИСТАННЯ AI-АСИСТЕНТІВ У ПРОЦЕСІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: ПЕРЕВАГИ ТА РИЗИКИ

Мартиненко М. С.
martynenkoms2005@gmail.com
Черкаський державний фаховий бізнес-коледж
Бреус Р.В.
м. Черкаси, Україна

Стрімкий розвиток генеративного штучного інтелекту суттєво трансформує сучасний процес розробки програмного забезпечення. Якщо раніше автоматизація охоплювала переважно компіляцію, тестування та розгортання програмних продуктів, то сьогодні AI-асистенти беруть участь безпосередньо у створенні програмного коду, поясненні логіки алгоритмів, генерації тестів, документуванні, пошуку помилок і рефакторингу. До таких інструментів належать системи на основі великих мовних моделей, які інтегруються у середовища програмування та виконують роль інтелектуального помічника розробника.

Актуальність теми зумовлена тим, що використання AI-асистентів у програмній інженерії вже стало масовою практикою: за даними GitHub, понад 97% опитаних учасників команд розробки вказали, що хоча б раз

використовували AI coding tools у роботі, а за даними Stack Overflow, 76% респондентів уже використовують або планують використовувати AI-інструменти у процесі розробки [1; 6].

Однією з головних причин швидкого поширення AI-асистентів є їхній позитивний вплив на продуктивність праці програмістів. Такі системи дозволяють швидше створювати шаблонний код, пропонують готові фрагменти функцій, допомагають адаптуватися до нових мов програмування та спрощують розуміння великих кодових баз. У контрольованому експерименті Microsoft Research встановлено, що група розробників, яка використовувала GitHub Copilot, виконала поставлене завдання в середньому на 55,8% швидше порівняно з контрольною групою. Додатково GitHub зазначає, що AI-інструменти допомагають розробникам вивільняти час для більш складних видів діяльності, зокрема проєктування систем, співпраці в команді та глибшого аналізу вимог [1; 2].

Практична цінність AI-асистентів проявляється на різних етапах життєвого циклу програмного забезпечення. На етапі проєктування вони можуть узагальнювати функціональні вимоги, пропонувати структуру модулів і допомагати формувати архітектурні рішення. На етапі програмування AI-асистенти застосовуються для автодоповнення коду, генерації окремих класів, методів і SQL-запитів. У процесі тестування вони здатні формувати модульні тести та підказувати граничні сценарії перевірки. GitHub у своєму опитуванні також вказує, що понад 98% респондентів повідомили про експерименти їхніх організацій із використанням AI для генерації тестових випадків. Це свідчить про поступову інтеграцію AI-асистентів не лише в написання коду, а й у забезпечення якості програмних продуктів [1].

На рис.1 показано основні етапи життєвого циклу програмного забезпечення, на яких AI-асистенти можуть застосовуватися для автоматизації та інтелектуальної підтримки роботи розробника.

Рисунок 1 – Використання AI-асистентів на етапах життєвого циклу програмного забезпечення

Водночас ефективність використання таких інструментів не означає повної безпечності або безумовної якості результату. Одним із ключових ризиків є генерація синтаксично правильного, але логічно некоректного коду. Великі мовні моделі не “розуміють” програму у класичному інженерному сенсі, а статистично прогнозують найімовірніші послідовності токенів. Через це можливі помилки в алгоритмах, неправильне використання бібліотек, порушення бізнес-логіки, а також так звані галюцинації, коли система впевнено пропонує неіснуючі методи, API або параметри. Небезпека підсилюється тим, що згенерований код часто виглядає переконливо та може бути сприйнятий розробником як коректний без належної перевірки. Саме тому зростає значення експертної валідації результатів, статичного аналізу та обов’язкового код-рев’ю [5].

Окрему групу становлять ризики інформаційної безпеки. Якщо розробник передає AI-асистенту фрагменти закритого коду, конфігураційні файли, ключі доступу, логіни або внутрішню документацію, це може призвести до витоку чутливої інформації. OWASP серед критичних ризиків для LLM-застосунків виокремлює prompt injection, insecure output handling, supply chain vulnerabilities та sensitive information disclosure. У контексті розробки програмного

забезпечення це означає, що AI-асистент може стати додатковою поверхнею атаки: зловмисно сформований ввід здатний вплинути на відповіді моделі, а небезпечний або неперевірений вихідний код може бути вбудований у систему без достатнього контролю. Крім того, використання сторонніх моделей або плагінів створює ризики, пов’язані з ланцюгом постачання програмного забезпечення, коли небезпека виникає не лише в самому коді, а й у зовнішніх сервісах, від яких залежить команда розробки [4].

Проблемним залишається і питання довіри до AI-асистентів. Хоча значна частина розробників позитивно оцінює потенціал таких засобів, реальні результати їх упровадження є неоднозначними. Згідно зі звітом DORA 2024, більше 75% респондентів використовують AI щонайменше для одного щоденного професійного завдання, а понад третина повідомила про помітне зростання продуктивності. Водночас зростання рівня використання AI асоціювалося не лише з покращенням якості документації, коду та швидкості рев’ю, але й із погіршенням окремих показників software delivery: дослідники зафіксували оцінюване зниження пропускної здатності доставки на 1,5% та стабільності доставки на 7,2%. Це свідчить, що AI-асистенти не є універсальним рішенням і не можуть замінити зрілих інженерних практик, таких як автоматизоване тестування, невеликі батчі змін, контроль версій, безпечний CI/CD та архітектурний нагляд [3].

Для узагальнення основних переваг і ризиків використання AI-асистентів у процесі розробки програмного забезпечення доцільно подати їх у вигляді таблиці.

Дані, наведені в табл.1, свідчать, що AI-асистенти поєднують значний потенціал для підвищення продуктивності з низкою технічних і безпекових ризиків, що вимагає контрольованого використання таких інструментів.

Таблиця 1 – Основні переваги та ризики використання AI-асистентів у процесі розробки програмного забезпечення

№	Переваги	Ризики
1	Прискорення написання типового коду	Генерація некоректного коду
2	Автоматизація створення тестів	Поява прихованих вразливостей
3	Спрощення роботи з новими технологіями	Витік конфіденційних даних
4	Допомога в документуванні	Надмірна залежність від підказок
5	Підтримка рефакторингу	Зниження рівня критичного аналізу коду

У зв’язку з цим ефективне впровадження AI-асистентів має базуватися на принципі «людина в контурі прийняття рішень». AI повинен розглядатися не як автономний розробник, а як інструмент інтелектуальної підтримки, результати якого підлягають перевірці. NIST у профілі AI RMF для генеративного ШІ наголошує на потребі системного управління ризиками протягом усього життєвого циклу використання таких технологій. Для команд розробки це означає необхідність формування політик безпечного використання AI, заборони на передавання чутливих даних у зовнішні моделі, журналювання взаємодії з асистентами, обов’язкової ревізії згенерованого коду та навчання персоналу правилам безпечної роботи з генеративним ШІ [5].

Отже, AI-асистенти вже стали важливим елементом сучасної програмної інженерії та мають значний потенціал для підвищення продуктивності, якості документації, швидкості тестування та підтримки розробників у роботі з новими технологіями. Водночас їх використання супроводжується технічними, безпековими, організаційними та правовими ризиками. Тому найбільш раціональним підходом є не безумовне захоплення можливостями AI, а його контрольоване впровадження в межах інженерних стандартів, де автоматизовані рекомендації поєднуються з професійною відповідальністю фахівця. Подальші дослідження варто спрямувати на розробку методів оцінювання якості AI-

згенерованого коду, моделей довіри до AI-асистентів та практик їх безпечної інтеграції у життєвий цикл програмного забезпечення.

Список використаних джерел:

1. Survey: The AI wave continues to grow on software development teams // GitHub Blog. 2024. URL: https://github.blog/news-insights/research/survey-ai-wave-grows/ (дата звернення: 14.03.2026).
2. Peng S., Kalliamvakou E., Cihon P., Demirer M. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot // Microsoft Research. 2023. URL: https://www.microsoft.com/en-us/research/publication/the-impact-of-ai-on-developer-productivity-evidence-from-github-copilot/ (дата звернення: 15.03.2026).
3. Accelerate State of DevOps Report 2024 // DORA. 2024. URL: https://cloud.google.com/blog/products/devops-sre/announcing-the-2024-dora-report (дата звернення: 16.03.2026).
4. OWASP Top 10 for Large Language Model Applications // OWASP Foundation. 2025. URL: https://owasp.org/www-project-top-10-for-large-language-model-applications/ (дата звернення: 17.03.2026).
5. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1. 2024. URL: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf (дата звернення: 18.03.2026).
6. AI. 2024 Stack Overflow Developer Survey // Stack Overflow. 2024. URL: https://survey.stackoverflow.co/2024/ai (дата звернення: 18.03.2026).