Киселевич В. В.,
асистент кафедри комп’ютерних
наук та інформаційних технологій,
Житомирський державний університет
імені Івана Франка, м. Житомир

УДК 378.147:004.415.2:004.8

ПІДГОТОВКА ФАХІВЦІВ З ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В УМОВАХ АГЕНТНИХ ІНСТРУМЕНТІВ

Постановка проблеми. Інструменти на основі великих мовних моделей перейшли від доповнення окремих рядків до виконання цілісних фрагментів роботи: створення функцій, тестів і виправлень за текстовим описом задачі. Для підготовки фахівців з інженерії програмного забезпечення це змінює сам

предмет навчання. Уміння написати реалізацію перестає бути дефіцитним ресурсом, тоді як уміння сформулювати перевірні вимоги, оцінити чужий результат і локалізувати дефект у коді, якого студент не писав, лишається дефіцитним і слабко забезпеченим. Потребу переглянути цілі, зміст і етичні складники підготовки обґрунтовано в літературі [1]. Проблема полягає в тому, що наявні настанови сформульовано на рівні загальних орієнтирів, тоді як викладачеві потрібне зіставлення конкретних умінь із конкретними дисциплінами та формами контролю.

Аналіз останніх досліджень. Емпіричні оцінки впливу генеративних інструментів на продуктивність розходяться. Польовий експеримент із 1974 розробниками зафіксував зростання кількості запитів на злиття на 12,92–21,83 відсотка [2], рандомізоване дослідження з 96 інженерами показало скорочення часу складного завдання приблизно на 21 відсоток [3], натомість дослідження з 16 досвідченими розробниками на 246 завданнях у зрілих проєктах зафіксувало протилежний ефект: виконання з доступом до інструментів тривало на 19 відсотків довше [4]. Якість згенерованого коду обмежує його безпосереднє використання: аналіз 4066 програм для 2033 задач показав 1082 неправильні результати й 1930 фрагментів із проблемами підтримуваності [5], близько 40 відсотків згенерованих фрагментів містили уразливості [6], а учасники з доступом до асистента писали менш безпечний код і водночас частіше вважали його безпечним [7]. Чинники, що визначають коректність і безпеку такого коду, узагальнено в огляді 24 первинних досліджень [8]. Для освітнього контексту істотно, що моделі значно перевершують студентів-початківців у виявленні логічних помилок (n = 964) [9], а таксономія станів взаємодії програміста з асистентом показує, що перевірка та відкидання пропозицій становлять помітну частку витрат часу [10]. Числові оцінки праць [4; 8] походять із препринтів, тому їх належить трактувати як попередні.

Мета. Мета роботи полягає в тому, щоб визначити перелік умінь, які стають критичними для фахівця з інженерії програмного забезпечення в умовах доступності агентних інструментів, зіставити ці уміння з наявною структурою дисциплін бакалаврської підготовки та сформулювати наслідки для оцінювання.

Виклад основного матеріалу. Наведені результати дають підставу для проміжного твердження: цінність праці фахівця зміщується від породження реалізації до формулювання перевірних умов і перевірки результату. Породження тексту програми частково автоматизоване, тоді як відповідальність за коректність, безпеку та підтримуваність лишається на людині [5; 6; 8]. Розбіжність оцінок продуктивності [2; 3; 4] свідчить, що перевага не є

автоматичним наслідком доступу до інструмента, а залежить від уміння розпізнати випадки, у яких перевірка чужого результату дорожча за самостійну реалізацію [10].

Запропоновано перелік восьми умінь, які виокремлено як критичні, і для кожного вказано місце формування та потрібну зміну (табл. 1). Уміння згруповано у три блоки: специфікація, верифікація та професійна рефлексія. Перелік і зіставлення є власним результатом автора, а емпіричні дані опрацьованих праць [1–10] використано як обґрунтування, а не як джерело готової класифікації.

Таблиця 1

Зіставлення критичних умінь із наявною структурою дисциплін

Уміння	Де формується зараз	Що потрібно змінити
Формулювання перевірних вимог і критеріїв приймання	Аналіз вимог, фрагментарно, без вимоги перевірності	Критерії приймання як обов’язковий артефакт задачі
Побудова тестів як засобу специфікації	Тестування програм, після готової реалізації	Приймальні тести до отримання коду
Читання й розуміння чужого коду	Побічно, на курсовому проєктуванні	Окремий модуль читання та рецензування коду
Локалізація дефекту в незнайомій кодовій базі	Системно не формується	Практикум налагодження на сторонніх репозиторіях
Критична перевірка правдоподібного результату	Системно не формується	Завдання з навмисно внесеними дефектами
Аудит безпеки згенерованих фрагментів	Основи безпеки, без зв’язку з генерацією	Перевірка згенерованого коду на типові уразливості
Оцінювання вартості та обмежень інструмента	Не передбачено освітньою програмою	Врахування часу перевірки й обчислювальних витрат
Фіксація власного внеску та походження коду	Загальні норми академічної доброчесності	Журнал запитів і декларація авторства фрагментів

Головним підсумком зіставлення є те, що три уміння з восьми не мають системного місця формування: локалізація дефекту в незнайомій кодовій базі, критична перевірка правдоподібного, але хибного результату та врахування вартості й обмежень інструмента. Перші два підтримуються даними про обмежену здатність студентів виявляти логічні помилки [9] і про надмірну впевненість у безпеці власного коду [7]. Ще два уміння формуються в порядку, що не відповідає новим умовам: тести вивчаються як засіб перевірки готової реалізації, а не як спосіб задати специфікацію до її появи.

Наслідком для оцінювання є втрата артефактом роздільної здатності як об’єкта контролю: якщо працездатну реалізацію типового завдання можна

отримати за кілька хвилин, оцінка готового коду майже не розрізняє студентів за рівнем розуміння. Як пропозицію автора сформульовано перенесення ваги оцінювання на процес і виявлення дефектів, для чого запропоновано три формати завдань. Перший полягає в локалізації навмисно внесеного дефекту в наданій реалізації обсягом 200–400 рядків, причому оцінюється протокол пошуку, а не саме виправлення. Другий передбачає написання приймальних тестів до чужої реалізації ще до її запуску. Третій є усним захистом з поясненням відкинутих альтернатив і декларацією походження кожного фрагмента [1]. Окремою умовою є посильність завдання: пошук має бути можливий за час аудиторного заняття.

Обмеження. Робота не містить власного педагогічного експерименту, тому перелік умінь і формати завдань не перевірено на навчальній вибірці. Зіставлення виконано за узагальненою структурою бакалаврської програми, а не за суцільним аналізом робочих програм. Усі числові показники запозичено з чужих вимірювань, здійснених у різних умовах, причому в дослідженнях продуктивності брали участь професійні розробники, а не студенти [2; 3; 4]. Оцінки праць [4; 8] не проходили рецензування, а частина джерел описує інструменти доповнення коду, а не повноцінні агентні системи [6; 10]. Вибірка обмежена десятьма працями 2023–2026 років, тому вона не є систематичним оглядом.

Висновки. Поява інструментів, здатних самостійно породжувати й виправляти код, зміщує предмет фахової підготовки з написання реалізації на формулювання перевірних вимог, перевірку чужого результату та локалізацію дефекту. Виокремлено вісім умінь, які стають критичними, і зіставлено їх зі структурою дисциплін: три з них не мають системного місця формування, а два формуються в порядку, що не відповідає новим умовам. Сформульовано умову, за якої оцінювання зберігає роздільну здатність: об’єктом контролю має бути процес і виявлення дефекту, а не лише поданий артефакт. Запропоновано три формати завдань, що реалізують цю умову. Перевірка їх на навчальній вибірці становить напрям подальшої роботи.

Список використаних джерел

1. Kirova V., Ku C. S., Laracy J. R. [та ін.] Software Engineering Education Must Adapt and Evolve for an LLM Environment. Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1. 2024. P. 666–672. DOI: 10.1145/3626252.3630927.

2. Cui K. Z., Demirer M., Jaffe S. [та ін.] The Productivity Effects of Generative AI: Evidence from a Field Experiment with GitHub Copilot. An MIT Exploration of Generative AI. 2024. DOI: 10.21428/e4baedd9.3ad85f1c.

3. Paradis E., Grey K., Madison Q. [та ін.] How Much Does AI Impact Development Speed? an Enterprise-Based Randomized Controlled Trial. 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). 2025. P. 618–629. DOI: 10.1109/icse-seip66354.2025.00060.

4. Becker J., Rush N., Barnes E. [та ін.] Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity : препринт. arXiv:2507.09089. 2025. URL: https://arxiv.org/abs/2507.09089.

5. Liu Y., Le-Cong T., Widyasari R. [та ін.] Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues. ACM Transactions on Software Engineering and Methodology. Vol. 33, No. 5. 2024. P. 1–26. DOI: 10.1145/3643674.

6. Pearce H., Ahmad B., Tan B. [та ін.] Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions. Communications of the ACM. Vol. 68, No. 2. 2025. P. 96–105. DOI: 10.1145/3610721.

7. Perry N., Srivastava M., Kumar D. [та ін.] Do Users Write More Insecure Code with AI Assistants? Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. 2023. P. 2785–2799. DOI: 10.1145/3576915.3623157.

8. Geruslu V., Aliyeva Z., Tüzün E. Factors Influencing the Quality of AI-Generated Code: A Synthesis of Empirical Evidence : препринт. arXiv:2603.25146. 2026. URL: https://arxiv.org/abs/2603.25146.

9. MacNeil S., Denny P., Tran A. [та ін.] Decoding Logic Errors: A Comparative Study on Bug Detection by Students and Large Language Models. Proceedings of the 26th Australasian Computing Education Conference. 2024. P. 11–18. DOI: 10.1145/3636243.3636245.

10. Mozannar H., Bansal G., Fourney A. [та ін.] Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. Proceedings of the CHI Conference on Human Factors in Computing Systems. 2024. P. 1–16. DOI: 10.1145/3613904.3641936.