УДК 004.8:004.05

ОЦІНКА ЯКОСТІ КОДУ В ЕПОХУ ГЕНЕРАТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ

Шабалтас В.А.
e-mail: vitalii.shabaltas@nure.ua
Науковий керівник – д. філос., ас. Дашенков Д.С.
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

The rapid integration of generative artificial intelligence into software development has significantly altered traditional coding workflows. However, the use of automated tools alone does not guarantee high code quality, nor can it completely replace human-driven architectural oversight. One promising approach to enhancing code maintenance is the balanced application of large language models for routine documentation. This technology enables an accelerated environment where teams automatically generate commit descriptions and docstrings. Yet, ensuring true quality requires a human-in-the-loop approach to mitigate contextual errors, proving that while AI simplifies technical routines, it cannot substitute the developer's contextual understanding.

У сучасному процесі розробки програмного забезпечення перш за все необхідно чітко окреслити, що саме слід розуміти під якістю коду. У контексті даного дослідження розглядаються процеси підтримування якості, а саме: читабельність кодової бази, її зрозумілість для інших розробників та наявність актуальної технічної документації. Зазвичай, якщо ці процеси виконуються класичним, конвенційним методом, розробники змушені витрачати значний час на рутинну роботу. Людина самостійно аналізує написаний нею алгоритм, формулює коментарі, описує вхідні та вихідні параметри функцій, а потім передає це на перевірку колегам. Цей підхід забезпечує високу точність, оскільки програміст володіє повним контекстом проєкту, але він є вкрай повільним.

Сучасні технології штучного інтелекту забезпечують принципово новий рівень автоматизації цих процесів [1]. Зокрема, на етапі написання коду розробнику достатньо зафіксувати зміни в системі контролю версій, після чого ШІ самостійно їх аналізує та формує розгорнутий опис разом з оновленою документацією до відповідних функцій [2, 3]. Хоча такий підхід видається оптимальним для прискорення розробки, він зумовлює низку викликів щодо певності автоматично створеної інформації.

З огляду на це, формується наступна гіпотеза: інструменти штучного інтелекту значно прискорюють етапи розробки, пов'язані з підтриманням якості кодової бази та її документуванням, але не виключають необхідності експертного контролю з боку розробника. Хоча ШІ ефективно автоматизує шаблонні процеси, складні інженерні завдання потребують критичного аналізу [4]. Практика показує, що іноді доцільніше

відмовитися від використання ШІ для розв'язання специфічних проблем: процес верифікації та усунення контекстних помилок вимагає від спеціаліста більше зусиль, ніж класичний ручний підхід.

Для підтвердження цієї гіпотези розглянемо реальний практичний кейс із генерації проєктної документації, а саме – формування списку функціональних вимог (SSR) до клієнтських застосунків програмної системи. Вибір цього етапу розробки зумовлений тим, що опис специфічної бізнес-логіки найчастіше виявляє типові недоліки генеративних моделей, зокрема схильність до галюцинацій та створення надлишкового функціоналу.

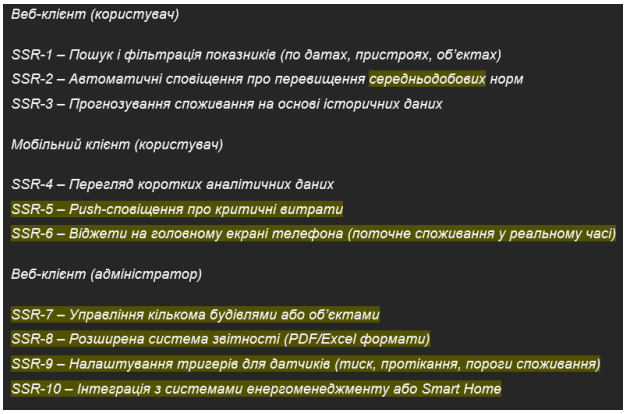

Рисунок 1 –
Текст, згенерований моделлю ШІ

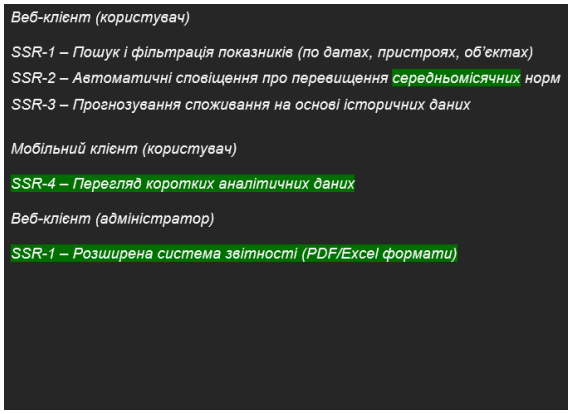

Рисунок 2 –
Відредагований текст
документації

На рисунку 1 представлено "сирий" результат, виданий генеративною моделлю до етапу верифікації. Натомість рисунок 2 демонструє затверджений текст, який пройшов експертну перевірку, стилістичну адаптацію та коректне форматування.

Алгоритм дій у цьому кейсі складався з кількох етапів. Крок перший: копіювання фрагмента коду та надсилання його із текстовим запитом для генерації документації. Крок другий: отримання результату та його візуальний поділ на правильні та хибні фрагменти. Крок третій: збереження коректної інформації – цей текст було залишено, оскільки він був технічно точним. Крок четвертий: виявлення змістових прогалин. Як виділено жовтим кольором на рисунку 1, ШІ згенерував загальний опис системи, але повністю проігнорував технічну специфіку та архітектуру проєкту. Крок п'ятий: експертне доопрацювання. На основі реального контексту системи розробник розширив документацію, додавши необхідний блок про взаємодію серверної частини (Back-end) та веб інтерфейсу. На рисунку 2 наведено фінальний, експертно адаптований варіант цього фрагмента.

Порівняння результатів на рисунках 1 та 2 наочно ілюструє, що відсутність безпосереднього експертного аналізу при використанні ШІ призвела б до втрати фінальної якості та точності документації.

Підбиваючи підсумки, результати використання генеративних моделей доцільно розділити на функціональні переваги та критичні обмеження. До основних переваг належить здатність ШІ ефективно автоматизувати початкові етапи структурування даних та виконувати великі обсяги шаблонної роботи, що суттєво знижує когнітивне навантаження на розробника. Водночас суттєвим обмеженням є неспроможність моделей враховувати унікальний контекст конкретного проєкту, що призводить до формування помилкових, проте стилістично переконливих суджень. Попри очікуване вдосконалення інструментів та появу нових механізмів інтеграції в майбутньому, у даному дослідженні оцінюється актуальний стан технологій та їхня практична придатність на сучасному етапі розробки.

Оптимізація описаного процесу можлива за допомогою кількох методів. По-перше, через застосування інженерії запитів, що передбачає включення до контексту моделі розширеної інформації про архітектуру системи. По-друге, перспективним напрямком є мультимодальна стратегія, яка полягає у паралельному використанні кількох великих мовних моделей (LLM) з подальшим порівняльним аналізом результатів [5]. Таким чином, збалансований симбіоз машинного генерування та експертного контролю забезпечує максимальну оптимізацію витрат часу без компромісів щодо архітектурної цілісності та надійності програмного продукту. Це дозволяє нівелювати специфічні недоліки окремих систем, забезпечуючи при цьому вирішальну роль фахівця у верифікації фінальної якості коду та документації.

Список використаних джерел:

1. Hou X., Zhao Y., Liu Y et al. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv.org. URL: https://arxiv.org/abs/2308.10620 (дата звернення: 10.03.2026).

2. Khan J. Y., Uddin G. Automatic Code Documentation Generation Using GPT-3. arXiv.org. 2022. URL: https://arxiv.org/abs/2209.02235 (дата звернення: 06.03.2026).

3. Chen M., Tworek J., Jun H. et al. Evaluating Large Language Models Trained on Code. arXiv.org. 2021. URL: https://arxiv.org/abs/2107.03374 (дата звернення: 06.03.2026).

4. Fan A., Gokkaya B., Harman M. et al. Large Language Models for Software Engineering: Survey and Open Problems. arXiv.org. URL: https://arxiv.org/abs/2310.03533 (дата звернення: 10.03.2026).

5. Huynh T., Thung F., Cheng L., Lo D. Models for Code Generation: A Comprehensive Survey. arXiv.org. 2023. URL: https://arxiv.org/abs/2311.02534 (дата звернення: 06.03.2026).