УДК 004.05:004.42:004.9 DOI: 10.31673/2786-8362.2026.014209

**Юрчик Д.Ю.**

# МЕТОДОЛОГІЧНІ АСПЕКТИ ВПРОВАДЖЕННЯ ГІБРИДНИХ МОДЕЛЕЙ ГЕНЕРАТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ В ПРОЦЕСИ АВТОМАТИЗАЦІЇ РЕГРЕСІЙНОГО ТЕСТУВАННЯ

**Yurchyk D.Yu. Methodological Aspects of Implementing Hybrid Generative AI Models in Regression Testing Automation Processes.** The article analyzes the transformation of software quality assurance practices under the growing adoption of Large Language Models (LLMs) in modern development workflows. It examines the integration of generative AI into the lifecycle of automated regression testing, including test design, page-object generation, implementation, and maintenance under UI changes. A comparative experiment contrasts traditional manual automation with an AI-assisted approach using time-based and quality-oriented metrics such as time-to-feedback, maintainability indicators, and test stability. The results demonstrate that a hybrid "human-in-the-loop" model significantly reduces effort in repetitive automation tasks and accelerates initial test development, while also limiting technical debt growth through faster adaptation to interface changes. At the same time, the study confirms the necessity of additional verification stages - including prompt validation, static code analysis, and human review – to mitigate hallucinated artifacts and manage the non-deterministic behavior inherent to probabilistic code generation.

**Keywords:** regression testing, Generative AI, LLM, test automation, Prompt Engineering, SDLC, QA metrics, technical debt

**Юрчик Д.Ю. Методологічні аспекти впровадження гібридних моделей генеративного штучного інтелекту в процеси автоматизації регресійного тестування.** У статті здійснено комплексний аналіз трансформації процесів забезпечення якості програмного забезпечення під впливом великих мовних моделей (LLM). Досліджено ефективність інтеграції генеративного штучного інтелекту в життєвий цикл створення автоматизованих регресійних тестів з точки зору продуктивності, підтримуваності та швидкості зворотного зв'язку. Проведено порівняльний експеримент між традиційною автоматизацією тестування та підходом, підсиленим використанням LLM. Отримані результати свідчать, що гібридна модель «human-in-the-loop» дозволяє суттєво скоротити час розробки тестових сценаріїв і зменшити темпи накопичення технічного боргу. Водночас обґрунтовано необхідність впровадження додаткових етапів валідації для зниження ризиків, пов'язаних із галюцинаціями та недетермінованістю генеративних моделей.

**Ключові слова:** регресійне тестування, генеративний ШІ, LLM, автоматизація тестування, Prompt Engineering, SDLC, метрики QA, технічний борг

## Вступ

Останні десятиліття характеризуються радикальним прискоренням циклів розробки програмного забезпечення. Парадигми Agile та DevOps, а також масове впровадження CI/CD конвеєрів перевели інженерні команди у режим безперервних релізів, де швидкість доставки функціоналу стала одним із ключових факторів конкурентоспроможності. Водночас саме тестування - і насамперед регресійне - часто перетворюється на «пляшкове горлечко» SDLC: зростає частота змін, збільшується поверхня ризику, а час на перевірку стабільності попередніх модулів скорочується. За цих умов особливої ваги набуває метрика Time-to-Feedback, що відображає швидкість отримання сигналу про деградацію якості після коміту або релізу.

Традиційна автоматизація регресії на UI-рівні здебільшого реалізується імперативно (Java/Python + Selenium/Playwright + архітектурні патерни на кшталт Page Object Model). Такий підхід забезпечує інженерний контроль, проте має структурний недолік: UI-тести крихкі та дорогі в підтримці через часті зміни DOM, локаторів, навігації та асинхронної поведінки клієнтської частини. Навіть за коректного застосування архітектурних патернів тестовий код схильний накопичувати технічний борг: зростає вартість змін, збільшується кількість тимчасових «обхідних» рішень, погіршується читабельність і передбачуваність прогонів.

Поява генеративного штучного інтелекту (GenAI) та великих мовних моделей (LLM) відкрила можливість переходу до частково декларативної автоматизації: створення частини тестового коду на основі описів поведінки, user story та навіть фрагментів HTML/DOM. На

практиці LLM здатні швидко синтезувати каркас тесту, Page Object-компоненти, локатори, очікування і типові асерти. Однак неконтрольоване використання генерації породжує ризики: галюцинації (вигадані селектори, методи, структури), порушення безпекових політик (витік чутливих даних у промптах і логах), а також падіння відтворюваності через недетерміновану природу генерації. Це формує науково-прикладне завдання: обґрунтувати методологію впровадження ШІ, яка поєднає продуктивність генерації з надійністю інженерного контролю.

**Аналіз останніх досліджень.** Класична школа тестування значною мірою спирається на концепцію «піраміди тестування», популяризовану Л. Кріспін та Дж. Грегорі [1]. Основна теза цієї концепції полягає в тому, що UI-рівень слід обмежувати через високу крихкість і вартість підтримки. Попри це, в сучасних веб- і мобільних продуктах повне уникнення E2E/GUI регресії часто неможливе через бізнес-ризики: критичні шляхи користувача (оформлення замовлення, платежі, керування акаунтом) потребують саме наскрізної валідації.

Стандартизаційна база тестування визначається ISO/IEC/IEEE 29119, зокрема частиною 29119-1:2022, де формалізовано загальні поняття тестування та термінологію [2]. Водночас стандарт історично орієнтований на детерміновані процеси й не деталізує особливості стохастичних генеративних систем як учасника інженерного циклу, зокрема вимоги до валідації результатів генерації та керування недетермінізмом.

Оскільки практики GenAI розвиваються швидше, ніж академічні публікації, важливим стає методологічний підхід multivocal literature review (MLR), який передбачає включення «сірої літератури»: звітів, технічних доповідей, whitepaper і практичних кейсів. Garousi, Felderer та Mäntylä обґрунтували правила проведення MLR і довели доцільність такого підходу для галузей із високою швидкістю змін [3]. Це є релевантним для тестування з LLM, оскільки індустріальні практики впровадження часто випереджають їх теоретичне узагальнення.

Звіт World Quality Report 2023–24 (Capgemini, Sogeti) підтверджує масштабну тенденцію експериментів із ШІ в QE/QA та водночас демонструє недостатню зрілість стратегій впровадження, включно з питаннями ризиків, політик і вимірювання ефекту [4]. Така диспропорція формує потребу у формалізації методології: що саме інтегрується в процес, як контролюється результат, і якими метриками оцінюється ефективність.

У науковому контексті застосування LLM у тестуванні розглядається в роботах, що описують підходи до генерації тестів, аналізу логів, підтримки тест-даних та автоматизації рутинних інженерних задач. Feldt та Poulding, узагальнюючи відповідні тенденції, підкреслюють, що LLM можуть підсилювати тестову інженерію, але вимагають контрольних процедур через ризики помилкової генерації та когнітивні упередження даних [5]. Водночас практична складність E2E-регресії проявляється сильніше саме на рівні локаторів, синхронізації та поведінкової стабільності UI.

Паралельно в інженерному середовищі накопичуються емпіричні дані про вплив «AI pair programmer» підходів на продуктивність. Дослідження GitHub щодо Copilot вказують на суттєве скорочення часу виконання типових задач і покращення суб'єктивного відчуття продуктивності [6]. Утім перенесення цього ефекту в домен E2E-автоматизації потребує обережності: виграш у швидкості синтезу каркасу може нівелюватися витратами на валідацію локаторів і логіки перевірок.

Інженерною основою підтримуваності UI-тестів залишається застосування Page Object Model, який прямо рекомендується як спосіб зменшення дублювання і локалізації змін у разі еволюції інтерфейсу [7]. Для об'єктивної оцінки підтримуваності та технічного боргу доцільно використовувати статичний аналіз і метрики maintainability/technical debt, що інструментально підтримуються SonarQube [8]. Додатково, як перспективний напрям, розглядаються self-healing підходи до тестів, що дозволяють скорочувати час відновлення після змін UI завдяки автоматизованому підбору оновлених локаторів і патчів [9].

**Постановка завдання.** Проблема дослідження полягає у відсутності формалізованого підходу до створення, перевірки та підтримки коду автотестів, згенерованого LLM, у регресійних E2E-сценаріях. Метою є розробка та експериментальна перевірка методології

гібридної автоматизації («AI-Augmented Automation»), що інтегрує техніки Prompt Engineering у класичний процес розробки автотестів.

Завданнями дослідження є: 1) провести порівняльний аналіз часових витрат на створення тестів вручну та за допомогою LLM; 2) оцінити якість і підтримуваність згенерованого коду за метриками статичного аналізу та індикаторами технічного боргу; 3) розробити алгоритм верифікації згенерованих сценаріїв і локаторів, що знижує ризики «галюцинацій» і підвищує відтворюваність результатів.

**Метою роботи** є підвищення ефективності процесів регресійного тестування шляхом розробки адаптивної методики застосування генеративних нейромереж для автоматичного створення та підтримки тестових скриптів, що дозволяє зменшити Time-to-Market і уповільнити приріст технічного боргу в тестовому репозиторії.

**Виклад основного матеріалу дослідження**

Дослідження ґрунтується на емпіричному аналізі практичного застосування генеративних мовних моделей у процесах автоматизації регресійного тестування. З огляду на прикладний характер поставлених завдань, дослідження орієнтоване не лише на теоретичне осмислення потенціалу LLM, а й на кількісну оцінку їх впливу на ключові інженерні показники якості автотестів. Особливу увагу приділено порівнянню традиційного підходу до автоматизації та гібридної моделі, у якій генерація тестового коду здійснюється за участю ШІ з подальшою валідацією результатів людиною.

Запропонований підхід розглядає генеративний ШІ як допоміжний інструмент у межах контрольованого життєвого циклу тесту, де швидкість створення коду поєднується з вимогами до підтримуваності, стабільності та відтворюваності регресійного набору. Для цього в дослідженні використано уніфіковане експериментальне середовище, формалізовані процедури генерації та верифікації тестів, а також набір метрик, що дозволяють комплексно оцінити ефективність обох підходів.

**Методологічні основи дослідження.** Емпіричне дослідження виконано на базі веб-додатку домену e-commerce. Для мінімізації впливу відмінностей інструментарію застосовано однаковий стек: Java + Selenium та Page Object Model як архітектурна рамка для обох варіантів.

Дизайн експерименту реалізовано як A/B порівняння:

1) Варіант А (Traditional): ручне написання коду (Java + Selenium + POM).
2) Варіант Б (AI-Assisted): той самий стек, але генерацію каркасу коду та частину рефакторингу делеговано LLM через стандартизовані промпт-шаблони.

В двох варіантах працювали з одним беклогом із 50 тестових сценаріїв різної складності, що охоплювали модулі «Кошик», «Оформлення замовлення», «Фільтрація товарів». Для зменшення впливу «ефекту навчання» сценарії розподілено так, щоб у кожному варіанті зберігався подібний профіль складності за кількістю кроків та UI-взаємодій.

Prompt Engineering було формалізовано у вигляді трьох типів промптів:

1) Промпт 1: генерація Page Object (вхід: HTML/DOM + правила локаторів; вихід: POM-клас).
2) Промпт 2: генерація тест-кейсу (вхід: user story + кроки; вихід: тест-метод із викликом POM).
3) Промпт 3: repair/refactor (вхід: stacktrace + DOM після зміни UI; вихід: патч локаторів і очікувань).

Застосування шаблонів промптів розглядалося як спосіб зниження ентропії генерації та підвищення повторюваності, що відповідає логіці методологічно контрольованих індустріальних практик (MLR).

**Метрики та інструментарій.** Для порівняння обрано метрики, що охоплюють швидкість, якість і стабільність:

1) Script Development Time (SDT) - час від отримання вимог до першого успішного прогону тесту.
2) Code Quality Metric (CQM) - складність, дублювання, code smells та індикатори технічного боргу на основі статичного аналізу (SonarQube).

3) Flakiness Rate / Reliability - частка помилкових падінь протягом 100 запусків.
4) Maintenance Time under UI Change (MTUC) - час адаптації тесту після контрольованої зміни UI.

Окремо фіксувалися типові дефекти генерації (вигадані селектори/методи), відповідність архітектурним домовленостям (POM, рівні абстракції), а також повторюваність результатів при повторній генерації за подібних промптів.

**Отримані результати дослідження.** Дані демонструють суттєве скорочення SDT у варіанті Б. Методологічно це пояснюється тим, що LLM швидко генерує каркас POM і тестів, включаючи повторювані елементи: структуру класів, шаблони очікувань, типові перевірки та допоміжні методи. Це детально представлено в таблиці 1.

Зниження LOC і невелике зменшення складності свідчать, що за наявності правил і архітектурних обмежень модель не лише прискорює створення, а й частково зменшує надмірність.

Найбільш виразним є показник MTUC. У варіанті Б часто відбувався сценарій: «прикріпити оновлений HTML/DOM → попросити оновити селектори → внести мінімальний патч». Це наближено до логіки self-healing, де система зменшує час відновлення тестів після зміни UI

Таблиця 1

Порівняльний аналіз ефективності створення тестових сценаріїв

| **Показник** | **Варіант А (Manual Coding)** | **Варіант Б (AI-Assisted)** | **Відхилення (%)** |
|---|---|---|---|
| Середній час розробки 1 сценарію (хв) | 65.4 | 28.2 | -56.8% |
| Цикломатична складність коду | 4.2 | 3.8 | -9.5% |
| Кількість рядків коду (LOC) | 120 | 95 | -20.8% |
| Час на рефакторинг при зміні UI (хв) | 25 | 8.5 | -66% |

Однак швидкість не є єдиним критерієм для регресії. Стабільність прогонів наведено у Таблиці 2.

Таблиця 2 показує, що AI-згенеровані тести на ранніх ітераціях можуть бути менш стабільними через помилки локаторів та семантики перевірок. Отже, LLM підвищує швидкість старту, але збільшує потребу у валідації. Відповідно роль QA-інженера зміщується від «написання» до архітектури, рев'ю та управління правилами генерації.

Таблиця 2

Аналіз стабільності тестів (Reliability Analysis)

| **Тип помилки** | **Варіант А (випадки)** | **Варіант Б (випадки)** | **Аналіз причин** |
|---|---|---|---|
| NoSuchElementException | 5 | 12 | LLM частіше пропонує крихкі селектори без стабільних атрибутів |
| Логічні помилки в Assertions | 2 | 8 | За нечіткого контексту модель може перевіряти не ті бізнес-дані. |
| StaleElementReferenceException | 4 | 3 | Частіше застосовано explicit waits та повторне отримання елементів. |

**Порівняння з іншими науковими дослідженнями.** Отримані результати щодо продуктивності узгоджуються з емпіричними спостереженнями щодо Copilot: генеративні інструменти прискорюють синтез шаблонного коду та зменшують час на «boilerplate».

Водночас у домені E2E-тестування виграш частково нівелюється необхідністю перевіряти локатори й логіку асертів. Це узгоджується з позицією, що LLM у тестуванні потребує контрольованих процедур та не може трактуватися як повноцінна заміна інженерного процесу.

Порівняно з нормативним підходом ISO/IEC/IEEE 29119-1, запропонований метод вимагає додаткового етапу «Prompt Verification» як частини життєвого циклу тесту: перевірка якості інструкції до генератора, перевірка результатів генерації та контроль відтворюваності перед інтеграцією у репозиторій. Це дозволяє поєднати стохастичний характер генерації з детермінованими вимогами до регресійного набору.

**Пропозиції автора.** На основі проведеного аналізу пропонується впровадити архітектуру «Smart Testing Pipeline», де LLM виконує роль «парного програміста», а кінцеве рішення про прийняття коду залишається за інженером.

Згідно з Рисунком 1, процес складається з трьох шарів:

1) Input Layer: тестувальник надає user story/acceptance criteria, тест-дизайн (Gherkin або покроковий опис) та фрагмент HTML/DOM, а також policy pack (локаторна стратегія, стиль коду, заборона *sleep*, вимоги до explicit waits).
2) Processing Layer: LLM генерує Page Object та тестові методи відповідно до шаблонів промптів і встановлених best practices (POM, іменування, структурування).
3) Validation Layer: артефакти проходять статичний аналіз (SonarQube), мінімальний smoke-прогін у CI та обов'язковий людський code review.

Ключовим елементом запропонованої архітектури є контур зворотного зв'язку: у разі падіння тесту система формує пакет артефактів (stacktrace, лог, DOM/HTML) і надсилає його для генерації патчу. Це відповідає концепції self-healing, але в межах «human-in-the-loop», де зміни не застосовуються автоматично без інженерного підтвердження.

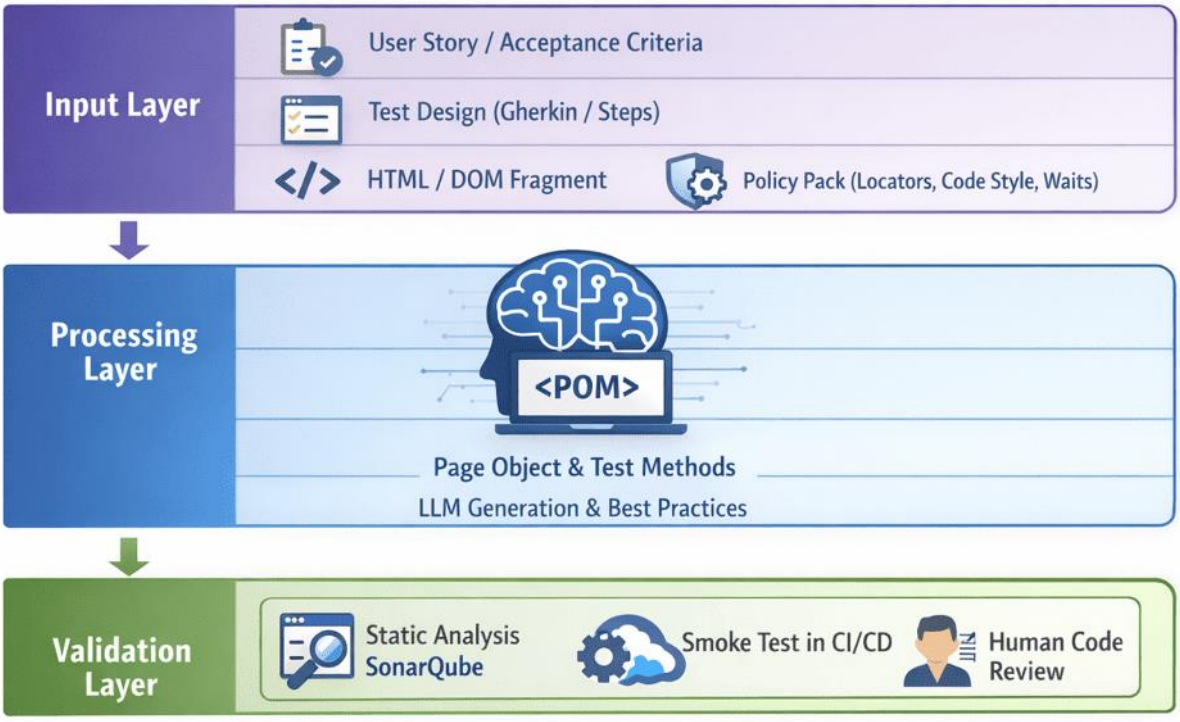


Рис. 1. Концептуальна модель «AI-in-the-Loop» для автоматизації тестування

**Обмеження та подальші дослідження.** Основним обмеженням є конфіденційність. Передача бізнес-вимог, коду, логів або DOM у публічні моделі може порушувати NDA чи внутрішні політики. У зв'язку з цим подальші дослідження мають включати порівняння публічних і локально розгорнутих LLM (private deployment), а також оцінку ефективності retrieval-підходів у периметрі безпеки.

Друге обмеження – недетермінованість: однакові промпти здатні продукувати різний код, що ускладнює відтворюваність. Частково проблема знижується шаблонізацією промптів і quality gates, проте потребує додаткового вивчення.

Третє обмеження – деградація якості локаторів за відсутності стабільних атрибутів у продукті. Тому перспективним напрямом є поєднання LLM-підходів із testability engineering,

зокрема узгодження з Dev-командами політики data-атрибутів та стабільних UI-хуків у межах POM-практик.

Крім того, ефективність AI-підсиленої автоматизації залежить від зрілості процесів тест-дизайну, якості вхідних вимог і наявності формалізованих правил інтеграції згенерованого коду в існуючий тестовий фреймворк.

**Висновки**

У результаті проведеного дослідження встановлено, що впровадження великих мовних моделей у процеси автоматизації регресійного тестування є перспективним напрямом оптимізації QA-практик, ефективність якого залежить від методологічної зрілості команди та наявності формалізованих процедур контролю якості. Порівняльний експеримент у межах двох варіантів підтвердив, що гібридний підхід «human-in-the-loop» забезпечує істотний виграш у швидкості розробки тестів і уповільнює накопичення технічного боргу за умови обов'язкової валідації згенерованих артефактів.

По-перше, встановлено, що застосування LLM для генерації каркасу тестів і Page Object-компонентів суттєво скорочує час первинної розробки сценаріїв та прискорює time-to-feedback. Емпіричні дані підтвердили зменшення середнього часу розробки одного сценарію на 56,8% у варіанті Б (AI-Assisted), що відповідає першому завданню дослідження.

По-друге, показано, що підвищення продуктивності супроводжується зростанням ризиків нестабільності тестів на ранніх ітераціях, зокрема через помилки локаторів і логічні неточності в assertions. Це обґрунтовує необхідність формалізованих етапів валідації та підтверджує виконання другого завдання дослідження -оцінки якості й підтримуваності згенерованого коду.

По-третє, встановлено, що ключовим фактором зниження технічного боргу є не сама генерація коду, а організація процесу його прийняття. LLM доцільно розглядати як інструмент створення чернетки, результати якої проходять «Prompt Verification», статичний аналіз, регресійний прогін і людський code review, що реалізує третє завдання дослідження.

По-четверте, сформульовано рекомендації щодо впровадження підходу «AI-in-the-Loop» у доменах із високими вимогами до надійності. Оптимальним є гібридний сценарій, у якому ШІ генерує каркас і підтримує рефакторинг, а людина відповідає за складні негативні та інтеграційні сценарії.

Наукова новизна роботи полягає у формалізації гібридної моделі застосування LLM у регресійній E2E-автоматизації з урахуванням стохастичної природи генерації. Запропонований підхід переносить фокус QA-інженера з написання коду на архітектуру, рев'ю та управління якістю вхідних інструкцій.

Отримані результати можуть бути застосовані в інших критичних доменах цифрових сервісів. Перспективи подальших досліджень пов'язані з розширенням емпіричної бази, порівнянням публічних і локальних LLM та розробкою формальних метрик якості промптів і прогнозування flakiness AI-згенерованих тестів.

**Список використаної літератури:**

1. Gregory J., Crispin L. Agile Testing: A Practical Guide for Testers and Agile Teams. Pearson Education, Limited, 2021.

2. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering — Software testing — Part 1: Concepts and definitions. URL: https://www.iso.org/standard/81291.html.

3. Garousi V., Felderer M., Mäntylä M. V. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*. 2019. Vol. 106. P. 101–121. URL: https://doi.org/10.1016/j.infsof.2018.09.006.

4. Capgemini, Sogeti. World Quality Report 2023–2024**.** URL: https://www.capgemini.com/insights/research-library/world-quality-report-2023-24/.

5. Feldt R., Poulding S. Test Automation with Large Language Models. *IEEE Software*. 2023. Vol. 40, no. 2. P. 18–26. URL: https://ieeexplore.ieee.org/document/10053992.

6. GitHub. Research: Quantifying GitHub Copilot's Impact on Developer Productivity and Happiness. 2023. URL: https://github.blog/research/publications/quantifying-github-copilots-impact-on-developer-productivity-and-happiness/.

7. Selenium Documentation. The Page Object Model (POM) Design Pattern. URL: https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/.

8. SonarSource. SonarQube Documentation: Maintainability and Technical Debt Metrics. URL: https://docs.sonarsource.com/sonarqube/latest/user-guide/metric-definitions/.

9. Autify. Self-healing Test Automation: Concept and Workflow. URL: https://autify.com/blog/self-healing-test-automation/.

***Автор статті***

**Юрчик Дмитро** – аспірант, Державний університет інформаційно-комунікаційних технологій, Київ, Україна.

ORCID: 0009-0000-9295-5354

***Author of the article***

**Yurchyk Dmytro** - postgraduate, State University of Information and Communication Technologies, Kyiv, Ukraine.

ORCID: 0009-0000-9295-5354

Надійшла до редакції: 20.04.2026
Прийнята до друку: 27.04.2026
Опубліковано: 25.05.2026