

DOI: <https://doi.org/10.15276/ict.03.2026.41>

УДК 004.41:004.85

Методологія перевірки програмного коду з підтримкою штучного інтелекту на основі моделі процесу

Аратовський Олександр Сергійович¹⁾

ORCID: <https://orcid.org/0009-0006-0854-8582>; alex.aratovskiy@stud.op.edu.ua. Scopus Author ID: 60503533100

Любченко Віра Вікторівна^{1), 2)}

ORCID: <https://orcid.org/0000-0002-4611-7832>; lvv@op.edu.ua. Scopus Author ID: 56667638800

¹⁾ Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

²⁾ Гамбурзький університет прикладних наук, Ульменліт 20. Гамбург, 21033, Німеччина

АНОТАЦІЯ

Перевірка програмного коду залишається центральною практикою забезпечення якості в проєктах програмного забезпечення, і віднедавна в цьому процесі діє новий учасник: асистент на основі великої мовної моделі, який генерує зауваження до змін, підсумки та чернетку виправлень. Наявні описи перевірки коду трактують такого асистента як зовнішній інструмент, тому питання про те, на яких кроках, за яких умов і з якими повноваженнями він має діяти, залишається без відповіді. У роботі розвинуто методологію перевірки програмного коду з підтримкою штучного інтелекту, побудовану на явній моделі процесу, у якій асистент є повноправним учасником без права на вердикт. Методологія охоплює шість принципів, шість точок інтеграції, прив'язаних до елементів моделі процесу, правила маршрутизації за прогнозованими характеристиками перевірки, контур накопичення зворотного зв'язку на вихід асистента та процедуру впровадження. Емпіричне обґрунтування виконано на дванадцяти репозиторіях з відкритим кодом з близько шістдесят п'яти тисяч злитих запитів. У перевірці беруть участь дванадцять таких асистентів. Принаймні одну їхню перевірку має приблизно кожен шостий злитий запит. Серед усіх зібраних запитів частка з такою участю зросла з десятих часток відсотка до понад третини за два роки, а між репозиторіями різнився на три порядки. Описове порівняння всередині репозиторіїв показує, що запити, перевірені асистентом, систематично більші за обсягом і проходять більше ітерацій, і ця відмінність зберігається всередині квартилів розміру зміни. Емпірично обґрунтовано принципи вимірюваності та прозорості конфігурації; принцип керування за прогнозом спирається на результати попередньої роботи авторів; для решти принципів описово перевірено передумови: асистенти майже не подають вердиктів, діють на тригері моделі процесу, а реакції людей на їхній вихід фіксує сама платформа.

Ключові слова: інженерія програмного забезпечення; перевірка програмного коду; запит на злиття; модель процесу; методологія; штучний інтелект; великі мовні моделі; асистент перевірки коду; зворотний зв'язок; аналіз репозиторіїв програмного забезпечення

Для цитування: Аратовський О. С., Любченко В. В. «Методологія перевірки програмного коду з підтримкою штучного інтелекту на основі моделі процесу». *Інформатика. Культура. Техніка.* 2026; Том 3 № 2 (4): 553–566. DOI: <https://doi.org/10.15276/ict.03.2026.41>

Актуальність. Перевірка програмного коду (code review) є однією з найпоширеніших практик забезпечення якості в проєктах програмного забезпечення: зміну, оформлену як запит на злиття, оглядають інші розробники, залишають зауваження та ухвалюють рішення щодо її прийняття [1], [2]. Результати перевірки важко передбачити на момент подання [3], [4]. З початку 2025 року в цьому процесі з'явився ще один учасник. Асистенти на основі великих мовних моделей, вбудовані в платформи спільної розробки, автоматично генерують зауваження до кожної поданої зміни, складають її підсумок і пропонують чернетку виправлень.

Це створює розрив між практикою та її описом. Описи перевірки коду [3], [4], засоби моделювання програмних процесів [5], [6] та емпіричні дослідження перевірки [1], [2] опубліковані до появи таких асистентів і знають лише автора, рецензента і службу безперервної інтеграції. Роботи про автоматизацію перевірки [7], [8], [9] оцінюють якість згенерованих зауважень і розглядають асистента як інструмент, що працює поруч із процесом. Де саме асистент має діяти, з якими повноваженнями і що робити з реакцією людей на його вихід, не визначено, а наслідки відомі з досліджень ботів: некерована автоматизація породжує шум, який рецензенти навчаються ігнорувати [10], [11]. Розв'язувати це питання треба на рівні методології сучасної перевірки коду [1], [2], тобто системи принципів і правил полегшеної асинхронної перевірки кожної зміни з інструментальною підтримкою та людським вердиктом, у компанії Google це полегшений процес здебільшого з

одним рецензентом [2], а систематичний огляд 244 досліджень [3] показує, що практики найбільше цінують дослідження властивостей самого процесу. Статистичні моделі параметрів перевірки [4], дослідження запитів на злиття як одиниці процесу [12] і участі рецензентів [13] описують процес словами, а не формальною моделлю, і без автоматизованих учасників. Моделювання програмних процесів має власну традицію [5], [6]. У попередній роботі авторів [14] її застосовано до перевірки коду і побудовано модель з асистентом як одним із чотирьох учасників, відправну точку для методології.

Прогнозування характеристик перевірки розвивалося окремо: час завершення запиту у хмарних сервісах [15] і служба нагадувань для прострочених запитів [16], час завершення перевірки у проєктах з відкритим кодом [17], затримка першої відповіді [18]. У роботі [14] показано, що прогнози кількості ітерацій переносяться між проєктами, а прогнози застарілості потребують калібрування на кожному проєкті; це визначає, які прогнози можуть керувати входом асистента.

Автоматизація перевірки за допомогою мовних моделей пройшла шлях від попередньо навчених моделей, що генерують зауваження і виправлення [7], [8], до моделей з ефективним донавчанням [9] та оцінювання розробниками зауважень від моделей загального призначення [19]; огляд [23] показує, що такі застосування оцінюють переважно за якістю виходу моделі, а не за зміною процесу. Впровадження в реальних проєктах показують і користь, і межі: автоматичне вирішення частини зауважень [20], контроль правил написання коду під час перевірки [21], автоматизована перевірка у компанії [22] з високою часткою вирішених зауважень, але з довшим часом закриття запитів. В агентах розробки під наглядом людини [24] інженер спрямовує план і код агента на кожному кроці; цей зразок контролю методологія доповнює накопиченням реакцій. Разом це дає модель процесу з асистентом [14], засоби генерації зауважень [7], [8], [9] і досвід їх застосування [20], [22], але не методологію, яка визначала б, де і за якими правилами діє асистент і як використати реакцію людей на його пропозиції. Саме її пропонує ця стаття.

Мета і завдання дослідження. Метою роботи є розвиток методології перевірки програмного коду з підтримкою штучного інтелекту на основі явної моделі процесу, яка дає змогу вбудувати асистента на основі великої мовної моделі у процес як керованого учасника та оцінювати його вплив за вимірюваними характеристиками процесу. Для досягнення мети розв'язано чотири завдання. Перше: сформулювати принципи, які відрізняють методологію від сучасної перевірки коду без автоматизованих учасників. Друге: визначити точки інтеграції асистента як елементи моделі процесу та задати для кожної з них тригер, вихід, особу, що ухвалює рішення, і сигнал зворотного зв'язку. Третє: сформулювати правила маршрутизації за прогнозованими характеристиками та контур накопичення зворотного зв'язку. Четверте: емпірично обґрунтувати принципи на корпусі репозиторіїв з відкритим кодом, де асистенти вже діють.

Постановка задачі. Дано: модель процесу [14] з чотирма учасниками, корпус з дванадцяти репозиторіїв і прогнози кількості ітерацій та застарілості на момент подання з якостями 0,89 і 0,71 (площа під характеристичною кривою). Знайти: систему принципів, точок інтеграції, правил маршрутизації та контур зворотного зв'язку, які роблять асистента керованим учасником процесу. Обмеження: вердикт лишається за людиною; використовуються лише прогнози, доступні на момент подання; реакції збираються з журналу платформи без спеціального інтерфейсу. Критерій: кожна точка має тригер, вихід, особу, що ухвалює рішення, і сигнал зворотного зв'язку; правила здійсненні за наявними прогнозами; принципи не суперечать даним корпусу; зміну трьох характеристик процесу перевіряє пілотне розгортання асистента на частині запитів за процедурою впровадження.

Модель процесу як основа методології. Методологія спирається на модель процесу перевірки коду з роботи [14]; Рис. 1 відтворює її з позначеними точками входу асистента T1–T6. Модель має процесну проєкцію у нотації моделювання бізнес-процесів і артефактну у вигляді автомата станів запиту на злиття. Процесна проєкція розрізняє чотирьох учасників. Рецензент оглядає зміну (окрема задача всередині перевірки) і виносить вердикт: схвалити,

запросити зміни або лише прокоментувати; служба безперервної інтеграції (CI на Рис. 1) виконує автоматичні перевірки кожної ревізії, і в разі невдачі автор виправляє зміну. Асистент на власний тригер, відкриття або оновлення запиту, генерує пропозиції, підсумки або чернетку виправлень; його вихід потрапляє до того самого потоку обговорення, що й зауваження людей, але не має права на вердикт.

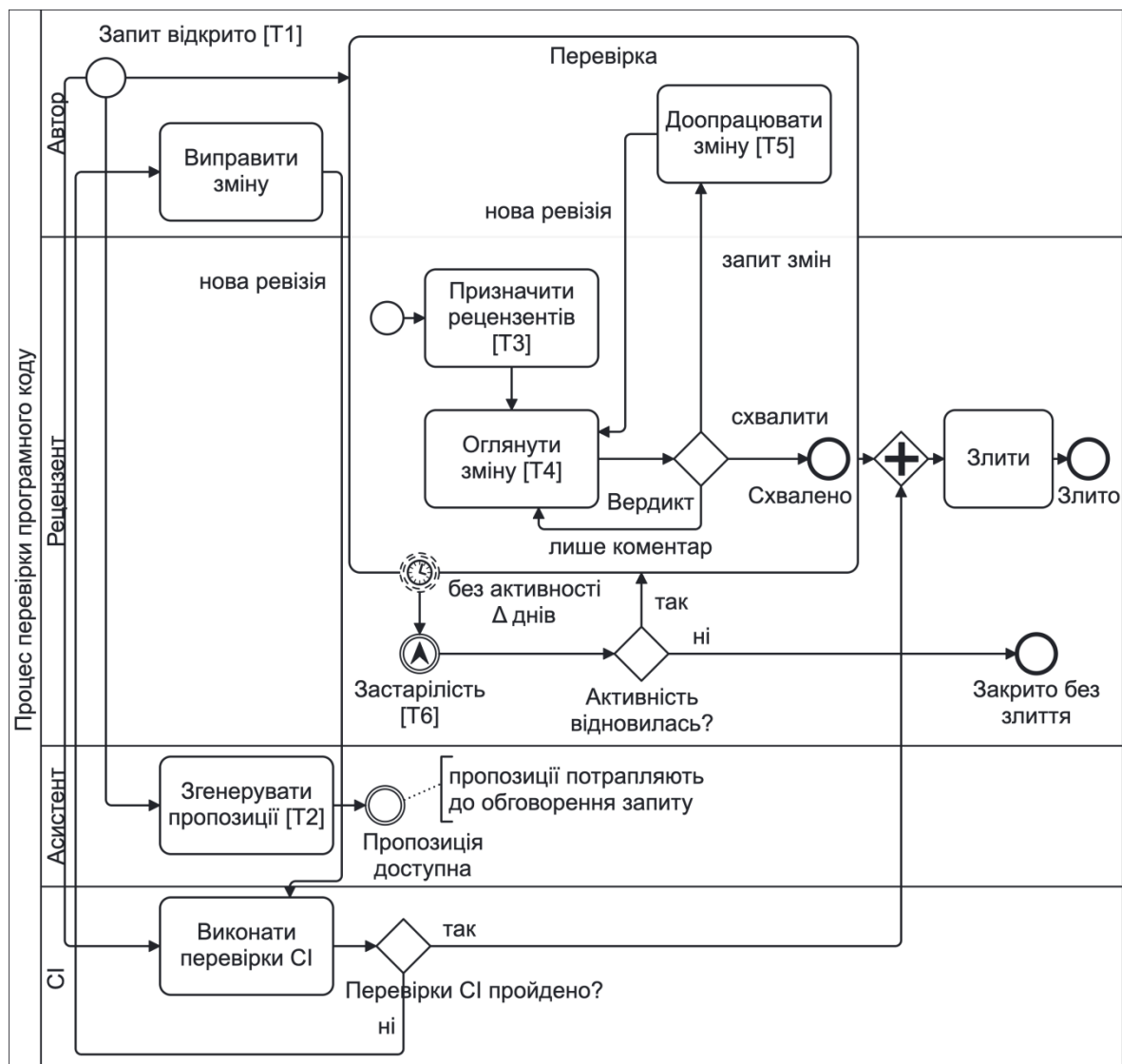


Рис. 1. Модель процесу перевірки програмного коду з позначеними точками інтеграції асистента T1-T6 (CI – служба безперервної інтеграції)

Джерело: адаптовано авторами за [14]

Шлях «запит змін» веде до доопрацювання, нова ревізія повторно запускає автоматичні перевірки і асистента і повертає процес до огляду; кожен прохід цього циклу є однією ітерацією перевірки. До підпроцесу перевірки приєднано граничну подію таймера: якщо впродовж порогу Δ немає жодної активності, запит позначається як застарілий; у [14] Δ дорівнює 30 дням, і якість прогнозу застарілості від порогу майже не залежить. Артефактна проєкція додає час перебування у стані; з неї виведено три вимірювані характеристики: тривалість перевірки, кількість ітерацій і застарілість.

Для методології ця модель важлива трьома властивостями: асистент у ній є учасником процесу з визначеним місцем у потоці; характеристики обчислюються з журналу подій, тому вплив асистента вимірний; кількість ітерацій і застарілість прогнозовані на момент подання зміни [14], що й дає змогу керувати входом асистента за прогнозом.

Методологія перевірки коду з підтримкою штучного інтелекту. Пропонована методологія розширює методологію сучасної перевірки коду [1], [2] чотирма складниками:

принципами, точками інтеграції, правилами маршрутизації за прогнозом і контуром зворотного зв'язку.

Принципи. Шість принципів методології, їхнє обґрунтування і місце реалізації зведені у Табл. 2.

Таблиця 2. Принципи методології перевірки коду з підтримкою штучного інтелекту

Принцип	Формулювання	Обґрунтування	Де реалізовано
П1 – Розділення повноважень	Асистент пропонує, людина вирішує: асистент не може ані схвалити, ані заблокувати зміну, а серед автоматизованих учасників блокувальні повноваження зберігає лише служба безперервної інтеграції	Передумова: асистенти майже не подають вердиктів (Табл. 7); нормативно [10], [11]	Рішення ухвалює людина в усіх точках Т1-Т6 (Табл. 1)
П2 – Прив'язка до процесу	Підтримка асистента вводиться лише у визначених точках моделі із заданими тригером, вхідними артефактами, виходом і адресатом; підтримка «поруч із процесом» виключається, бо саме вона породжує шум	Передумова: зауваження з'являються на тригері моделі (Табл. 7); нормативно [10], [11]	Т1-Т6 як елементи моделі (Рис. 1, Табл. 1)
П3 – Керування за прогнозом	Чи входить асистент у процес у конкретній точці і в якому обсязі, визначають прогнозовані на момент подання характеристики перевірки	Прогнози ітерацій і застарілості з [14]; участь асистента не є предиктором (розділ «Прогнозованість входу»)	Правила маршрутизації (Табл. 3) у точках Т1, Т3, Т6
П4 – Накопичення зворотного зв'язку	Кожна реакція людини на вихід асистента (прийняття, відхилення, редагування, закриття обговорення) фіксується як позначений приклад для ранжування подальших пропозицій та адаптації асистента	Передумова: реакції фіксує сама платформа (Табл. 7); нормативно [20], [24]	Мигнали зворотного зв'язку Т1-Т6 (Табл. 1), контур (Рис. 2)
П5 – Вимірюваність	Вплив асистента оцінюється лише за характеристиками процесу, обчисленими з журналу подій, окремо для кожного проєкту, з контролем розміру зміни та періоду	Емпірично: участь різниться між репозиторіями на три порядки, відмінності зберігаються у квартилях розміру (Табл. 4, Табл.5, Табл. 6)	Процедура впровадження, етапи 1 і 4
П6 – Прозорість	Вихід асистента марковано як згенерований, відповідальність лишається за людьми, а конфігурація асистента є явним артефактом проєкту	Емпірично: участь є переважно властивістю проєкту (Табл. 4)	Іаркування виходу у Т2; політика входу як артефакт проєкту (Рис. 2)

Джерело: розроблено авторами

Точки інтеграції. Точки Т1-Т6 позначені на Рис. 1 і зведені у Таблицю 1.

Таблиця 1. Точки інтеграції асистента у процес перевірки програмного коду

Точка	Елемент моделі; тригер	Вихід асистента	Рішення ухвалює	Сигнал зворотного зв'язку
T1 – Подання зміни	Подія «запит відкрито»; до початку огляду, за прогнозом ітерацій	Опис зміни, самоперевірка, порада розбити зміну	Автор	Прийняття поради, фактична кількість ітерацій
T2 – Попередня перевірка	Задача «згенерувати пропозиції»; відкриття або оновлення запиту	Ранжовані зауваження до зміни, підсумок	Автор, рецензент	Прийняття, відхилення, редагування зауважень
T3 – Призначення рецензентів	Задача «призначити рецензентів»; завершення попередньої перевірки	Ранжований перелік рецензентів	Рецензент із правами на злиття	Фактичний рецензент, час до першої реакції
T4 – Огляд зміни	Задача «оглянути зміну»; початок огляду	Підсумок зміни, пояснення ризикових ділянок, зіставлення зауважень	Рецензент	Вердикт, використані пояснення
T5 – Доопрацювання	Задача «доопрацювати зміну»; вердикт «запит змін»	Чернетки виправлень для прийнятих зауважень	Автор	Прийняття чернетки, результат автоматичних перевірок
T6 – Нагляд за застарілістю	Подія таймера «застарілість»; прогноз застарілості або таймер	Нагадування, пропозиція ескалації, підсумок обговорення	Рецензент із правами на злиття	Відновлення активності, закриття

Джерело: розроблено авторами

Точка T1 діє на момент подання: оскільки домінантним предиктором кількості ітерацій є структура поданої зміни [14], автору повертається високий прогноз ітерацій як порада розбити або консолідувати зміну. Порада необов'язкова і має ціну: за порогу 0,5 приблизно кожна третя порада зайва [14], тому поріг для T1 обирається за прийнятною для проєкту часткою зайвих порад. Точка T2 відповідає задачі асистента у моделі, де сьогодні діють наявні інструменти; методологія додає ранжування зауважень за прогнозованою корисністю і маркування виходу, а поки модель корисності не навчена, T2 працює з евристичним ранжуванням, наприклад, за рівнем серйозності від самого асистента і належністю зауваження до змінених рядків. Точки T3-T6 дають ранжований перелік рецензентів, підсумок зміни із зіставленням зауважень, чернетки виправлень для прийнятих зауважень [20] і нагадування або пропозицію ескалації за таймером застарілості [16]; рішення в кожній з них лишається за людиною (Табл. 1).

Правила маршрутизації за прогнозом. Три рішення, які підтримують прогнози, сформульовано в [14]; тут їх прив'язано до точок інтеграції та принципу ПЗ і зведено разом із компромісами у Таблиці 3. Прогноз ітерацій застосовний у новому проєкті без донавчання, а прогноз застарілості потребує калібрування порогів на історії проєкту [14], тому ескалація

вводиться після етапу калібрування, а через компроміси правил методологія оцінює три характеристики разом.

Таблиця 3. Правила маршрутизації за прогнозом

Умова	Дія	Точка	Компромід
Прогнозована ймовірність двох і більше ітерацій перевищує поріг проєкту	Зміна спрямовується до рецензента, який має час на тривалий обмін; зміни з прогнозом одного проходу групуються у короткі сесии перевірки (черговість)	T3	Пришвидшує прості зміни коштом складних
Той самий прогноз ітерацій	Прогноз повертається авторові як порада розбити або консолідувати зміну (попередній зворотний зв'язок)	T1	Збільшує кількість запитів і чергу; частина порад зайва
Прогнозована ймовірність застарілості перевищує поріг проєкту	Явне призначення рецензента замість очікування таймера (ескалація)	T6	Перерозподіляє увагу рецензентів; потребує калібрування порогу на історії проєкту

Джерело: розроблено авторами

Контур зворотного зв'язку. Контур накопичення зворотного зв'язку показано на рис. 2. Політика входу асистента, тобто набір активних точок T1-T6 і правил маршрутизації для проєкту, визначає, які дії асистента запускаються. Реакції людей на кожну пропозицію разом із вердиктом утворюють корпус позначених прикладів, на якому періодично виконується адаптація: переранжування пропозицій, налаштування підказок асистента і порогів політики проєкту. Контур не потребує спеціального інтерфейсу, бо платформи вже фіксують вирішення обговорень, прийняття змін і вердикти; це показано далі в розділі «Реакції на вихід асистентів».

Процедура впровадження. Методологія задає п'ять етапів: вимірювання базового стану процесу за трьома характеристиками на історії проєкту; вибір активних точок інтеграції та правил маршрутизації за профілем проєкту (частки багатоітераційних і застарілих запитів, медіанний час до першої реакції); пілотне розгортання асистента в обраних точках на частині запитів із маркуванням його виходу (П6) і фіксацією реакцій людей (П4); оцінювання зміни характеристик для кожного репозиторію з контролем розміру зміни і періоду; масштабування, коригування або відкат політики входу. Критерій відкату явний: якщо з урахуванням розміру змін погіршилася будь-яка з трьох характеристик або частка прийнятих пропозицій асистента опустилася нижче порогу проєкту, політику входу повертають до попереднього стану.

Емпіричне обґрунтування методології. Емпіричне обґрунтування відповідає на чотири питання: чи діє асистент у процесі, чи відрізняються характеристики процесу за його участі, чи можна керувати його входом за прогнозом і чи фіксує платформа реакції людей на його вихід. Усі чотири перевірено на корпусі роботи [14], побудованому за методикою [25]: дванадцять репозиторіїв з відкритим кодом, шість мов програмування, разом 64 820 злитих і 7 156 відкритих запитів на злиття.

Участь асистентів. Чи є асистент дійсним учасником процесу? Учасників перевірки класифіковано за явним переліком облікових записів асистентів на основі великих мовних моделей, кожен з яких верифіковано шляхом прочитання вибірки його зауважень; детерміновану автоматизацію (звіти безперервної інтеграції, оновлення залежностей, лінтери) обліковано окремо. Перевіркою асистентом вважається принаймні одна подія перевірки або коментар в обговоренні від такого запису, який не є автором запиту.



Рис. 2. Контур накопичення зворотного зв'язку на вихід асистента

Джерело: розроблено авторами

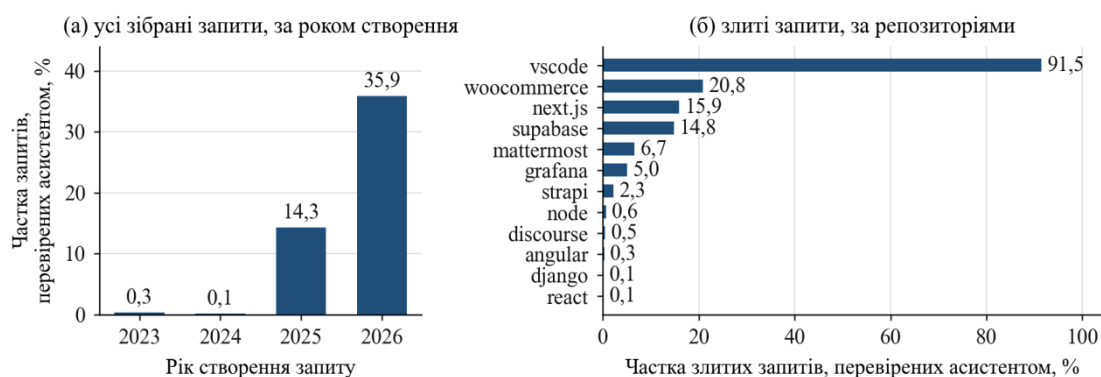


Рис. 3. Частка запитів на злиття з принаймні однією перевіркою асистентом: а – усі зібрані запити за роком створення; б – злиті запити за репозиторіями

Джерело: розроблено авторами

У корпусі діють дванадцять асистентів. Найпоширеніший, GitHub Copilot code review, перевіряв 7 118 запитів в усіх дванадцяти репозиторіях; далі йдуть CodeRabbit (2 748 запитів, три репозиторії), Vercel Agent (1 403, один), Graphite Diamond (517, один), Claude Code (433, чотири) і OpenAI Codex (175, чотири); решта шість мають від одного до 28 запитів кожен. Один запит можуть перевіряти кілька асистентів; принаймні одну перевірку асистентом мають 10 278 злитих запитів, тобто 15,9 %. Участь з'являється стрибком з лютого 2025 року. Серед усіх зібраних запитів, злитих і відкритих разом, частка перевірених асистентом за роком створення становить 0,3 % для 2023 року (відкриті запити, перевірені у 2025 році), 0,1 % для 2024, 14,3 % для 2025 і 35,9 % для 2026 року (Рис. 3а). Окремо зафіксовано авторство: 663 злиті запити (1,0 %) створені агентами на основі великих мовних моделей, майже всі в одному репозиторії.

Розподіл участі між репозиторіями (Рис. 3б; Табл. 4) є ключовим спостереженням для методології. Частка злитих запитів, перевірених асистентом, коливається від 0,1 % у django та react до 91,5 % у vscode; у чотирьох репозиторіях вона перевищує 10 %, у п'яти не досягає 1 %; домінантний асистент також різниться. Отже, участь асистента є переважно властивістю проекту: між репозиторіями вона різниться на три порядки, тоді як усередині репозиторію відбір за зміною є, але менший (Табл. 5). Команда вмикає асистента для репозиторію, і з цього моменту він перевіряє значну частину змін, а у vscode майже всі. Це обґрунтовує принципи П6 (конфігурація на рівні проекту) і П5 (оцінювання для кожного репозиторію): об'єднане по корпусу число про вплив асистента вимірювало б склад репозиторіїв, а не асистента.

Таблиця 4. Участь асистентів у перевірці злитих запитів за репозиторіями

Репозиторій	Основна мова	Злитих запитів	Перевірені асистентом	Частка, %	Домінантний асистент
vscode	TypeScript	6 212	5 683	91,5	GitHub Copilot
woocommerce	PHP	7 647	1 594	20,8	CodeRabbit
next.js	TypeScript	8 392	1 337	15,9	Vercel Agent
supabase	TypeScript	5 869	869	14,8	CodeRabbit
mattermost	Go	5 081	338	6,7	CodeRabbit
grafana	Go	6 420	324	5,0	GitHub Copilot
strapi	TypeScript	2 418	55	2,3	Claude Code
node	JavaScript	1 872	12	0,6	GitHub Copilot
discourse	Ruby	9 319	47	0,5	GitHub Copilot
angular	TypeScript	3 080	9	0,3	Gemini Code Assist
react	JavaScript	7 127	8	0,1	GitHub Copilot
django	Python	1 383	2	0,1	GitHub Copilot
Разом		64 820	10 278	15,9	

Джерело: розроблено авторами

Стовпці «Злитих запитів» і «Частка, %» відтворюють дані з Табл. 2 з [14], решта нові Повні назви репозиторіїв: microsoft/vscode, woocommerce/woocommerce, vercel/next.js, supabase/supabase, mattermost/mattermost, grafana/grafana, strapi/strapi, nodejs/node, discourse/discourse, angular/angular, facebook/react, django/django. GitHub Copilot тут означає GitHub Copilot code review; node містить також код мовою C++

Асистент і характеристики процесу. Чи відрізняються характеристики процесу для запитів, перевірених асистентом, і без такої перевірки? Порівняння є описовим, лише всередині одного репозиторію і лише для запитів, створених у 2025-2026 роках, коли обидві групи співіснують. Для восьми репозиторіїв, у яких кожна з груп містить не менше 20 запитів, порівняно кількість ітерацій, час до злиття і розмір зміни (додані рядки) за двобічним критерієм Манна-Вітні. Мірою ефекту є рангова бісеріальна кореляція з 95 % бутстреп-інтервалом (2 000 повторень), а до 24 значень p застосовано поправку Холма (Табл. 5). Додатна міра означає, що величина більша у групі з асистентом; вона вимірює зсув розподілів, а не медіан.

В усіх восьми репозиторіях запити, перевірені асистентом, проходять більше ітерацій; у семи з восьми вони також значуще більші за обсягом (у 2,4-6,7 раза за медіаною доданих рядків), а у шести з восьми вони також довше йдуть до злиття. Винятки пояснююані: у vscode асистент перевіряє майже кожну зміну, і без перевірки лишаються дрібні швидкі запити; у strapi група з асистентом налічує лише 55 запитів. Причинного прочитання ці дані не допускають: команди вмикають асистента там, де очікують значних зусиль на перевірку. Різниця у розмірі між групами сама здатна пояснити більшу кількість ітерацій, бо розмір і структура зміни є доміантними предикторами ітерацій [14], тому порівняння ітерацій повторено всередині квартилів доданих рядків, як того вимагає принцип П5 (Табл. 6).

Відмінність у кількості ітерацій зберігається всередині кожного квартиля розміру у п'яти репозиторіях з шести і здебільшого зростає з розміром зміни; зникає вона лише для найдрібніших змін у vscode, де майже всі запити проходять один прохід. Отже, більша кількість ітерацій не пояснюється самим лише розміром зміни; це сумісне з припущенням, що вихід асистента входить у цикл доопрацювання, але не доводить його, бо відбір за іншими ознаками, наприклад, за ризиковістю змінених ділянок, залишається можливим. Для методології це аргумент на користь П5 і підстава керувати обсягом виходу асистента через ранжування (Т2, П4) і перехоплювати найбільші зміни у Т1.

Таблиця 5. Описове порівняння злитих запитів, перевірених асистентом, і без такої перевірки, 2025-2026 роки

Репозиторій	Запитів з / без асистента	Ітерації: медіани; міра ефекту	Час до злиття, год: медіани; міра ефекту	Додані рядки: медіани; міра ефекту
vscode	5 683 / 529	1 / 1; 0,24 ± 0,03	1,0 / 5,5; -0,34 ± 0,05	17 / 6; 0,23 ± 0,05
woocommerce	1 593 / 2 531	3 / 1; 0,53 ± 0,03	73,7 / 28,0; 0,26 ± 0,03	71 / 19; 0,32 ± 0,03
next.js	1 336 / 4 898	2 / 1; 0,50 ± 0,03	46,4 / 12,2; 0,36 ± 0,03	119,5 / 33; 0,34 ± 0,03
supabase	869 / 4 552	2 / 1; 0,55 ± 0,03	28,8 / 14,6; 0,25 ± 0,04	102 / 18; 0,38 ± 0,04
mattermost	338 / 2 758	3 / 1; 0,51 ± 0,06	163,2 / 19,3; 0,40 ± 0,06	81,5 / 34; 0,24 ± 0,06
grafana	324 / 6 096	2 / 1; 0,54 ± 0,05	47,8 / 15,2; 0,32 ± 0,05	267 / 40; 0,45 ± 0,05
strapi	55 / 789	2 / 1; 0,23 ± 0,14*	63,8 / 69,3; -0,07 ± 0,15**	98 / 52; 0,13 ± 0,15**
discourse	47 / 7 298	2 / 1; 0,40 ± 0,14	28,6 / 3,6; 0,40 ± 0,14	105 / 18; 0,46 ± 0,15

Джерело: розроблено авторами

Медіани наведено для груп з асистентом / без; ± половина ширини 95 % бутстреп-інтервалу. Усі відмінності значущі $p < 0,001$ після поправки Холма, крім: * $p = 0,005$; ** незначуще ($p = 0,40$ і $0,21$). Групи woocommerce і next.js менші, ніж у Табл. 4, на один запит, створений до 2025 року

Таблиця 6. Міра ефекту для кількості ітерацій усередині кuartилів доданих рядків, 2025-2026 роки

Репозиторій	Перший кuartиль	Другий кuartиль	Третій кuartиль	Четвертий кuartиль
vscode	-0,01 ± 0,05**	0,14 ± 0,07	0,31 ± 0,08	0,48 ± 0,11
woocommerce	0,36 ± 0,07	0,42 ± 0,07	0,47 ± 0,06	0,60 ± 0,05
next.js	0,40 ± 0,09	0,44 ± 0,07	0,41 ± 0,06	0,55 ± 0,05
supabase	0,33 ± 0,12	0,35 ± 0,08	0,52 ± 0,07	0,56 ± 0,06
mattermost	0,14 ± 0,13*	0,48 ± 0,12	0,50 ± 0,09	0,59 ± 0,08
grafana	0,37 ± 0,21	0,36 ± 0,16	0,35 ± 0,12	0,52 ± 0,07

Джерело: розроблено авторами

Кuartили обчислено за всіма злитими запитамі репозиторію 2025-2026 років; у кожній клітинці не менше 24 запитів з асистентом і 65 без; ± половина ширини 95 % бутстреп-інтервалу. Усі значення значущі, $p < 0,05$ після поправки Холма на 25 тестів, крім: * $p = 0,03$; ** незначуще. У strapi усі чотири, у discourse три кuartили з чотирьох містять менше 20 запитів з асистентом; четвертий кuartиль discourse (27 запитів; $+0,45 \pm 0,18$; $p < 0,001$) є двадцять п'ятим тестом і в таблицю не включений

Прогнозованість входу. Чи можна керувати асистентом за прогнозом? У роботі [14] перевірено, чи додає частка запитів, перевірених асистентом, в історії репозиторію інформацію до прогнозів ітерацій і застарілості; не додає, бо майже повністю пояснюється ідентичністю репозиторію. Отже, прогнози обчислюються незалежно від участі асистента, і вона лишається керованою змінною, а не предиктором; це знімає циклічну залежність у ПЗ.

Реакції на вихід асистентів. Чи фіксує платформа те, що потрібно для контуру зворотного зв'язку? Відповідь дають 21 815 обговорень, розпочатих асистентами у 7 083 злитих запитах (обговорення в рядках коду є не в кожному перевіреному запиті), і 18 957 подій перевірки від асистентів (Табл. 7).

По-перше, 87,2 % зауважень з'явилися протягом години після відкриття запиту або нової ревізії, тобто на тригері задачі асистента в моделі (П2).

По-друге, вердикт містить лише 1,9 % подій перевірки, і майже всі вони припадають на mattermost, де CodeRabbit подає схвалення і запити змін; саме там П1 є обмеженням, а не описом практики.

По-третє, реакції фіксуються без спеціальних засобів (П4): вирішено 82,4 % обговорень, відповідь людини є у 17,6 % (мовчазне прийняття невидиме), принаймні один із двох сигналів має 84,3 % обговорень; готове виправлення міститься у 42,8 % зауважень, і 20,7 % таких виправлень у запитах, створених людьми, застосовано комітом «Apply suggestions from code review» або комітом зі співавторством асистента. Обговорення вирішує автор або рецензент, і рецензент може відкрити його знову, тому вирішення є дією людини, але не розкриває, чи пропозицію прийнято: 54,6 % усіх обговорень вирішено після нового коміту без відповіді, 15,7 % з відповіддю, 12,1 % мовчазно. Між інструментами частка відповідей різнилась від 10,6 % у GitHub Copilot code review до 42,9 % в OpenAI Codex, між репозиторіями ще більше (П5). Водночас відповідь людини слабо передбачувана за властивостями зауваження: у чотирьох найбільших репозиторіях модель, навчена на 75 % історії, відрізняється від найкращої тривіальної евристики не більш як на 0,05 площі під кривою точність-повнота; приріст є лише у grafana та mattermost (0,16 і 0,33 при 62 і 40 позитивних прикладах у тесті). Для мітки «застосовано виправлення» приріст досягає 0,18–0,19 у next.js і supabase, тому модель корисності для T2 потребує саме міток на рівні зміни коду, які й має накопичувати контур.

Таблиця 7. Реакції людей на вихід асистентів у злитих запитах, 2025–2026 роки

Репозиторій	Обговорень / подій перевірки	≤ 1 год після тригера, %	Відповідь людини / вирішено, %	Готове виправлення / з них застосовано, %	Подій перевірки з вердиктом, %
vscode	8 357 / 6 004	98,3	7,6 / 78,0	49,4 / 25,4	0,0
woocommerce	5 752 / 6 061	85,0	31,3 / 89,7	34,7 / 14,4	0,0
next.js	2 781 / 3 117	67,1	16,4 / 87,6	46,2 / 15,9	0,0
supabase	2 717 / 1 911	91,8	14,7 / 72,5	38,7 / 20,7	0,1
mattermost	681 / 932	83,3	30,5 / 81,2	40,1 / 31,9	39,2
grafana	1 170 / 604	65,4	25,6 / 87,6	36,6 / 12,6	0,0
strapi	177 / 218	53,7	8,5 / 82,5	32,8 / 50,0	0,0
discourse	76 / 51	78,9	19,7 / 93,4	69,7 / 17,0	0,0
Разом	21 815 / 18 957	87,2	17,6 / 82,4	42,8 / 20,7	1,9

Джерело: розроблено авторами

Разом: усі дванадцять репозиторіїв; angular, node, react і django (разом 104 обговорення і 59 подій перевірки) окремо не показано. Тригер: відкриття запиту або нова ревізія. Застосовано: частка зауважень із блоком suggestion у запитах, створених людьми, після яких є коміт “Apply suggestions from code review” або зі співавторством асистента. Вердикт: стан APPROVED або CHANGES_REQUESTED

Обговорення та обмеження. Отримані результати наповнюють роль асистента в моделі процесу емпіричним змістом; підстави для кожного принципу наведені в Табл. 2, а нормативний зміст П1, П2 і П4, тобто заборона вердикту як правило, виключення підтримки поза процесом і приріст від накопичення реакцій, залишається обґрунтованим лише роботами [10], [11], [20], [24].

Обмеження з боку даних: корпус охоплює лише проекти з відкритим кодом на одній платформі і лише ранній етап поширення асистентів (2025–2026 роки), тому висновки про комерційну розробку є екстраполяцією; для відкритих запитів обговорення в рядках коду недоступне, тому частка перевірених асистентом серед них є нижньою оцінкою; перелік облікових записів асистентів застаріває, а асистенти під обліковими записами людей

невидимі; запити, створені агентами, не вилучалися (у групах Табл. 5 їх 274, з них 260 з асистентом у vscode).

Обмеження з боку методології: приріст якості ранжування з накопиченням реакцій не показано, а наявні реакції слабо передбачувані за властивостями зауважень, тому модель корисності для T2 потребує міток на рівні зміни коду; точки T3 і T6 спираються на метод ранжування рецензентів, який тут не оцінювався, і на прогноз застарілості, який потребує калібрування на кожному проекті.

Зіставлення з опублікованими результатами. Довший час до злиття для запитів, перевірених асистентом, узгоджується з [22], де після впровадження автоматизованої перевірки в компанії час закриття запиту зріс з приблизно шести до понад восьми годин; механізм пояснюють [10], [11], [19]: зауваження без відбору з часом ігноруються, що й є підставою для ранжування у T2.

Наступні кроки: мітки корисності на рівні зміни коду в місці зауваження та перевірка приросту ранжування з їх накопиченням, і інформаційна технологія супроводу перевірки, яка реалізує методологію як конвеєр (збір журналу подій, прогнозування, політика входу, ранжування, збір реакцій).

Висновки. У роботі розвинуто методологію перевірки програмного коду з підтримкою штучного інтелекту, побудовану на явній моделі процесу, у якій асистент на основі великої мовної моделі є повноправним учасником без права на вердикт. Вона доповнює методологію сучасної перевірки коду шістьма принципами, шістьма точками інтеграції з визначеними атрибутами, правилами маршрутизації за прогнозом, контуром накопичення зворотного зв'язку і процедурою впровадження з явним критерієм відкату. Наукова новизна полягає у тому, що підтримка асистента задана як елемент явної моделі процесу і керується прогнозами характеристик самого процесу, а не додається до нього як зовнішній інструмент. Практичне значення полягає в процедурі впровадження, яку команда може виконати на власній історії перевірки без спеціальних засобів для збору реакцій.

Емпіричне дослідження на корпусі з дванадцяти репозиторіїв показало, що асистенти вже є дійсними учасниками процесу: 15,9 % злитих запитів мають принаймні одну їхню перевірку, а участь є властивістю проекту з розкидом від 0,1 до 91,5 % між репозиторіями. Запити, перевірені асистентом, систематично більші і потребують більше ітерацій, і ця відмінність зберігається всередині кuartилів розміру зміни. Емпірично обґрунтовано принципи П5 і П6; П3 спирається на результати [14]; для П1, П2 і П4 перевірено передумови: лише 1,9 % подій перевірки від асистентів містять вердикт, 87,2 % зауважень з'являються на тригері моделі, а відповідь або вирішення платформа зафіксувала для 84,3 % обговорень. Наступний крок: мітки корисності на рівні змін у коді та реалізація методології в інформаційній технології супроводу перевірки.

Конфлікт інтересів: Автори декларують, що не мають конфлікту інтересів, зокрема фінансового, особистого, авторського чи будь-якого іншого характеру, який міг би вплинути на дослідження, а також на результати, опубліковані в цій статті

Фінансування: Дослідження проводилося без фінансової підтримки

Доступність даних: Дані зібрано з відкритого програмного інтерфейсу платформи GitHub. Перелік облікових записів асистентів з правилами класифікації, агреговані таблиці участі за репозиторіями і роками, результати порівнянь (табл. 5–7) та скрипти рисунків доступні від авторів за запитом; повний набір даних готується до оприлюднення з постійним ідентифікатором, як зазначено в [14]

Використання засобів штучного інтелекту: Під час підготовки рукопису використовувався інструмент штучного інтелекту Claude Code (Anthropic; версія 2.1.273; модель Claude Fable 5.1) для редагування тексту, перевірки узгодженості числових даних із вихідними таблицями і форматування за шаблоном журналу. Усі фрагменти, опрацьовані за його допомогою, перевірені й відредаговані авторами. Методологія, дані, результати та висновки є власним інтелектуальним внеском авторів

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bacchelli, A. & Bird, C. "Expectations, outcomes, and challenges of modern code review". *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. San Francisco, CA, USA. 2013. p. 712–721, <https://www.scopus.com/pages/publications/84886394450>. DOI: <https://doi.org/10.1109/ICSE.2013.6606617>.

2. Sadowski, C., Söderberg, E., Church, L., Sipko, M. & Bacchelli, A. “Modern code review: a case study at Google”. *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Gothenburg, Sweden. 2018. p. 181–190, <https://www.scopus.com/pages/publications/85049691666>. DOI: <https://doi.org/10.1145/3183519.3183525>.
3. Badampudi, D., Unterkalmsteiner, M. & Britto, R. “Modern code reviews: survey of literature and practice”. *ACM Transactions on Software Engineering and Methodology*. 2023; 32 (4): 1–61, <https://www.scopus.com/pages/publications/85163852501>. DOI: <https://doi.org/10.1145/3585004>.
4. Rigby, P. C., German, D. M., Cowen, L. & Storey, M.-A. “Peer review on open-source software projects: parameters, statistical models, and theory”. *ACM Transactions on Software Engineering and Methodology*. 2014; 23 (4): 1–33, <https://www.scopus.com/pages/publications/84907033452>. DOI: <https://doi.org/10.1145/2594458>.
5. Fuggetta, A. “Software process: a roadmap”. *Proceedings of the Conference on the Future of Software Engineering (ICSE)*. Limerick, Ireland. 2000. p. 25–34, <https://www.scopus.com/pages/publications/85023607484>. DOI: <https://doi.org/10.1145/336512.336521>.
6. García-Borgoñón, L., Barcelona, M. A., García-García, J. A., Alba, M. & Escalona, M. J. “Software process modeling languages: a systematic literature review”. *Information and Software Technology*. 2014; 56 (2): 103–116, <https://www.scopus.com/pages/publications/84889885711>. DOI: <https://doi.org/10.1016/j.infsof.2013.10.001>.
7. Tufano, R., Masiero, S., Mastropaolo, A., Pascarella, L., Poshyvanyk, D. & Bavota, G. “Using pre-trained models to boost code review automation”. *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. Pittsburgh, PA, USA. 2022. p. 2291–2302. DOI: <https://doi.org/10.1145/3510003.3510621>.
8. Li, L., Yang, L., Jiang, H., Yan, J., Luo, T., Hua, Z., Liang, G. & Zuo, C. “AUGER: automatically generating review comments with pre-training models”. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. Singapore. 2022. p. 1009–1021. DOI: <https://doi.org/10.1145/3540250.3549099>.
9. Lu, J., Yu, L., Li, X., Yang, L. & Zuo, C. “LLaMA-Reviewer: advancing code review automation with large language models through parameter-efficient fine-tuning”. *Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. Florence, Italy. 2023. p. 647–658. DOI: <https://doi.org/10.1109/ISSRE59848.2023.00026>.
10. Wessel, M., de Souza, B. M., Steinmacher, I., Wiese, I. S., Polato, I., Chaves, A. P. & Gerosa, M. A. “The power of bots: characterizing and understanding bots in OSS projects”. *Proceedings of the ACM on Human-Computer Interaction*. 2018; 2 (CSCW): 1–19, <https://www.scopus.com/pages/publications/85064157845>. DOI: <https://doi.org/10.1145/3274451>.
11. Wessel, M., Wiese, I., Steinmacher, I. & Gerosa, M. A. “Don't disturb me: challenges of interacting with software bots on open source software projects”. *Proceedings of the ACM on Human-Computer Interaction*. 2021; 5 (CSCW2): 1–21. DOI: <https://doi.org/10.1145/3476042>.
12. Gousios, G., Pinzger, M. & van Deursen, A. “An exploratory study of the pull-based software development model”. *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. Hyderabad, India. 2014. p. 345–355, <https://www.scopus.com/pages/publications/84994130040>. DOI: <https://doi.org/10.1145/2568225.2568260>.
13. Thongtanunam, P., McIntosh, S., Hassan, A. E. & Iida, H. “Review participation in modern code review: an empirical study of the Android, Qt, and OpenStack projects”. *Empirical Software Engineering*. 2017; 22 (2): 768–817, <https://www.scopus.com/pages/publications/84991786769>. DOI: <https://doi.org/10.1007/s10664-016-9452-6>.
14. Aratovskyi, O. S. & Liubchenko, V. V. “A process-grounded framework for submission-time prediction of code review outcomes”. *Herald of Advanced Information Technology*. 2026; 9 (4). DOI: <https://doi.org/10.15276/hait.09.2026> (у друці).

15. Maddila, C., Bansal, C. & Nagappan, N. “Predicting pull request completion time: a case study on large scale cloud services”. *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. Tallinn, Estonia. 2019. p. 874–882, <https://www.scopus.com/pages/publications/85071919194>. DOI: <https://doi.org/10.1145/3338906.3340457>.
16. Maddila, C. S., Upadrasta, S. S., Bansal, C., Nagappan, N., Gousios, G. & van Deursen, A. “Nudge: accelerating overdue pull requests toward completion”. *ACM Transactions on Software Engineering and Methodology*. 2023; 32 (2): 1–30, <https://www.scopus.com/pages/publications/85143061311>. DOI: <https://doi.org/10.1145/3544791>.
17. Chouchen, M., Ouni, A., Olongo, J. & Mkaouer, M. W. “Learning to predict code review completion time in modern code review”. *Empirical Software Engineering*. 2023; 28 (4): 82, <https://www.scopus.com/pages/publications/85160020919>. DOI: <https://doi.org/10.1007/s10664-023-10300-3>.
18. Khatoonabadi, S., Abdellatif, A., Costa, D. E. & Shihab, E. “Predicting the first response latency of maintainers and contributors in pull requests”. *IEEE Transactions on Software Engineering*. 2024; 50 (10): 2529–2543, <https://www.scopus.com/pages/publications/85201309320>. DOI: <https://doi.org/10.1109/TSE.2024.3443741>.
19. Watanabe, M., Kashiwa, Y., Lin, B., Hirao, T., Yamaguchi, K. & Iida, H. “On the use of ChatGPT for code review: do developers like reviews by ChatGPT?”. *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. Salerno, Italy. 2024. p. 375–380. DOI: <https://doi.org/10.1145/3661167.3661183>.
20. Frömmgen, A., Austin, J., Choy, P., Ghelani, N., Kharatyan, L., Surita, G., Khrapko, E., Lamblin, P., Manzagol, P.-A., Revaj, M., Tabachnyk, M., Tarlow, D., Villela, K., Zheng, D., Chandra, S. & Maniatis, P. “Resolving code review comments with machine learning”. *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Lisbon, Portugal. 2024. p. 204–215, <https://www.scopus.com/pages/publications/105011043499>. DOI: <https://doi.org/10.1145/3639477.3639746>.
21. Vijayvergiya, M., Salawa, M., Budiselić, I., Zheng, D., Lamblin, P., Ivanković, M., Carin, J., Lewko, M., Andonov, J., Petrović, G., Tarlow, D., Maniatis, P. & Just, R. “AI-assisted assessment of coding practices in modern code review”. *Proceedings of the 1st ACM International Conference on AI-Powered Software (AIware)*. Porto de Galinhas, Brazil. 2024. p. 85–93. DOI: <https://doi.org/10.1145/3664646.3665664>.
22. Cihan, U., Haratian, V., İçöz, A., Gül, M. K., Devran, Ö., Bayendur, E. F., Uçar, B. M. & Tüzün, E. “Automated code review in practice”. *Proceedings of the 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Ottawa, Canada. 2025. p. 425–436. DOI: <https://doi.org/10.1109/ICSE-SEIP66354.2025.00043>.
23. Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J. & Wang, H. “Large language models for software engineering: a systematic literature review”. *ACM Transactions on Software Engineering and Methodology*. 2024; 33 (8): 1–79. DOI: <https://doi.org/10.1145/3695988>.
24. Takerngsaksiri, W., Pasuksmit, J., Thongtanunam, P., Tantithamthavorn, C., Zhang, R., Jiang, F., Li, J., Cook, E., Chen, K. & Wu, M. “Human-in-the-loop software development agents”. *Proceedings of the 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Ottawa, Canada. 2025. p. 342–352. DOI: <https://doi.org/10.1109/ICSE-SEIP66354.2025.00036>.
25. Aratovskyi, O. S. & Liubchenko, V. V. “Constructing a GitHub-based dataset for machine learning driven software effort estimation”. *Proceedings of the 2025 IEEE 13th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. Gliwice, Poland. 2025. p. 1–7, <https://www.scopus.com/pages/publications/105032752133>. DOI: <https://doi.org/10.1109/IDAACS68557.2025.11322078>.

Отримано 02.09.2026

Отримано після перегляду 12.10.2026

Прийнято 14.10.2026

DOI: <https://doi.org/10.15276/ict.03.2026.41>

UDC 004.41:004.85

A process-model-based methodology for code review supported by artificial intelligence

Oleksandr S. Aratovskyi¹⁾

ORCID: <https://orcid.org/0009-0006-0854-8582>; alex.aratovskyi@stud.op.edu.ua. Scopus Author ID: 60503533100

Vira V. Liubchenko^{1), 2)}

ORCID: <https://orcid.org/0000-0002-4611-7832>; lvv@op.edu.ua. Scopus Author ID: 56667638800

¹⁾ Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine

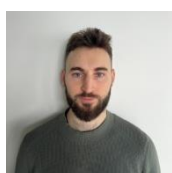
²⁾ Hamburg University of Applied Sciences, 20, Ulmenliet. Hamburg, 21033, Germany

ABSTRACT

Code review remains a central quality assurance practice in software projects, and a new participant has recently entered it: an assistant based on a large language model that generates review comments, summaries, and draft fixes. Existing descriptions treat such an assistant as an external tool and leave open where it acts and with what authority. This paper develops a methodology for code review supported by artificial intelligence, built on an explicit process model in which the assistant is a first-class participant without gating authority. The methodology comprises six principles, six integration points bound to process-model elements, routing rules driven by predicted review characteristics, a feedback accumulation loop on assistant output, and an adoption procedure. The empirical grounding uses twelve open-source repositories with about sixty-five thousand merged pull requests. Twelve such assistants take part in review; roughly one merged pull request in six carries at least one of their reviews; among all collected pull requests, the share of those with such a review grew from a fraction of a percent to over a third in two years and differs across repositories by three orders of magnitude. A descriptive within-repository comparison shows that pull requests reviewed by an assistant are systematically larger and go through more iterations, which persists within quartiles of change size. Measurability and transparent configuration are grounded empirically; prediction-driven control rests on earlier results; the preconditions of the remaining principles are verified descriptively: assistants rarely issue verdicts, act on the model trigger, and the platform records human reactions.

Keywords: Software engineering; code review; pull request; process model; methodology; artificial intelligence; large language models; code review assistant; feedback; mining software repositories

ІНФОРМАЦІЯ ПРО АВТОРІВ



Аратовський Олександр Сергійович - аспірант кафедри Інженерії програмного забезпечення. Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

Галузь досліджень: інженерія програмного забезпечення; аналіз репозиторіїв програмного забезпечення

Oleksandr S. Aratovskyi - PhD student, Department of Software Engineering. Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine

ORCID: <https://orcid.org/0009-0006-0854-8582>; alex.aratovskyi@stud.op.edu.ua. Scopus Author ID: 60503533100

Research field: software engineering; machine learning for software engineering; mining software repositories



Любченко Віра Вікторівна - доктор технічних наук, професор кафедри Інженерії програмного забезпечення. Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна; викладач, Гамбурзький університет прикладних наук, Ульменліт 20. Гамбург, 21033, Німеччина

Галузь досліджень: наука про дані; інженерія програмного забезпечення; управління проєктами

Vira V. Liubchenko - Doctor of Engineering Sciences, Professor, Department of Software Engineering. Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine; Lecturer, Hamburg University of Applied Sciences, Ulmenliet 20, Hamburg, 21033, Germany

ORCID: <https://orcid.org/0000-0002-4611-7832>; lvv@op.edu.ua. Scopus Author ID: 56667638800

Research field: Data Science; Software Engineering; Project Management