НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кваліфікаційна наукова
праця на правах рукопису

КУРІНЬКО ДМИТРО ДМИТРОВИЧ

УДК 004.416:004.8

ДИСЕРТАЦІЯ

МОДЕЛІ ТА МЕТОДИ МАШИННОГО НАВЧАННЯ ДЛЯ ВИЯВЛЕННЯ ТА УСУНЕННЯ АНТИПАТЕРНІВ В ПРОГРАМНИХ КОМПОНЕНТАХ

Спеціальність 122 – Комп’ютерні науки
Галузь знань 12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результаті і текстів інших авторів мають посилання на відповідне джерело.

_____ Д. Д. Курінько

Науковий керівник:
Кривда Вікторія Ігорівна, кандидат технічних наук, доцент

Одеса – 2026

АНОТАЦІЯ

Курінько Д.Д. Моделі та методи машинного навчання для виявлення та усунення антипатернів в програмних компонентах. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 122 – Комп'ютерні науки. – Національний університет «Одеська політехніка», МОН України, Одеса, 2026.

У **вступі** показана актуальність вирішення задач автоматизованого виявлення потреби в рефакторингу об’єктно-орієнтованого програмного коду в сучасних програмних системах. Підкреслено недоліки наявних моделей і методів виявлення рефакторинг-можливостей, зокрема підходів на основі «запахів» коду, програмних метрик і методів машинного навчання, які не враховують контекст проєкту, взаємозалежність ознак та зашумленість даних. Визначено об’єкт, предмет, мету, задачі та методи дослідження; наведено наукову новизну та практичне значення отриманих результатів; висвітлено особистий внесок здобувача.

У **першому розділі** дисертаційної роботи виконано системний огляд проблематики виявлення та усунення антипатернів у програмних компонентах як ключового чинника зниження внутрішньої якості та ускладнення еволюції промислових програмних систем. Показано, як зростання масштабів і складності програмного забезпечення призводить до накопичення дефектів проєктування та зв’язок антипатернів з деградацією підтримуваності, ускладненням модифікацій та підвищенням вартості розвитку. На цій основі сформульовано постановку задачі автоматичного виявлення та усунення антипатернів і окреслено місце таких рішень у сучасному інженерному процесі.

Далі наведено класифікацію антипатернів і показано, що їх прояви не зводяться до одного типу сигналів: узагальнено структурні, семантичні та еволюційні (історичні) ознаки, а також підкреслено мультимодальність проявів і пов’язану з цим проблему невизначеності. Після цього проаналізовано традиційні

підходи (метрики, правила, детектори), а також окремо розглянуто структурні і текстові/LLM-орієнтовані підходи, із фокусом на їх сильні сторони та обмеження з погляду практичного застосування.

Окремо були проаналізовані сучасні підходи на базі машинного та глибинного навчання: описано класичні моделі на метриках і мітках «запахів» коду, графові та репрезентаційні моделі подання коду, гібридні рішення та uncertainty-aware підходи, а також напрями, де ці підходи використовується для підтримки рефакторингу та роботи з дублікатами коду.

Аналіз робіт в області автоматичного усунення антипатернів показав, що існуючі моделі, як правило, спираються на один тип ознак та не відображають багатовимірну природу антипатернів, що спричиняє хибні спрацювання або пропуски. Для подолання цього недоліку була запропонована багаторівнева модель на гібридному графовому поданні коду з об’єднанням структурних, семантичних і еволюційних ознак в єдиному просторі.

Подальший аналіз існуючих систем виявлення антипатернів показав, що детекція антипатернів на рівні методу/класу не виявляє компонентні/системні антипатерни та не враховує динаміку якості між релізами. Для усунення цього недоліку було запропонованого багаторівневе подання та аналіз із урахуванням зв’язків між рівнями, а також метод композиції «мінімально інвазивних» послідовностей рефакторингів, що враховує причинно-наслідкові залежності змін.

Аналіз сучасних підходів виявлення антипатернів показав, що вони розвиваються розрізнено, тому можуть давати суперечливі висновки та рекомендації. Було показано, що перспективним є використання гібридних подань, які інтегрують ознаки в єдиний вектор. У якості такого рішення, в роботі було запропоноване уніфіковане гібридне подання як спільну основу і для детекції антипатернів, і для оптимізації планів рефакторингу.

Подальший аналіз методів виявлення антипатернів показав, що їх робота базується на припущенні, що всі типи антипатернів відомі заздалегідь, що не відповідає сучасній практиці розробки програмного забезпечення. Крім того, більшість методів не надає каліброваної оцінки невизначеності прогнозів, що

підвищує ризик агресивних або, навпаки, надто обережних рішень. Для подолання даних недоліків, у роботі пропонується введення open-set/low-confidence режиму в моделі виявлення антипатернів та явного моделювання невизначеності при рекомендації рефакторингів.

Аналіз практики впровадження інструментів виявлення антипатернів показав, що їх результати погано інтегровані в роботу команд та сучасні практики безперервної інтеграції та доставлення. Для подолання цього недоліку, в роботі був запропонований модуль рекомендації рефакторингів, метод композиції послідовностей змін та модель статистичного контролю процесів, що дозволяє формалізовано оцінювати результативність втручань і знижувати ризик хибних рішень.

За результатами проведеного аналізу сформульовано мету та завдання досліджень.

У **другому розділі** дисертаційної роботи уточнено постановку задачі автоматичного виявлення антипатернів з урахуванням їх мультимодальної природи (структура, семантика, метрики, еволюція), багаторівневого контексту («метод – компонент – проєкт») та потреби open-set/low-confidence режиму з явним урахуванням невизначеності. Обґрунтовано обмеження правил та метрик і одно-модальних підходів на базі машинного навчання та сформульовано вимоги до нової моделі (переносимість, інкрементальність для безперервної інтеграції та доставлення).

Запропоновано гібридне графове подання коду у вигляді Code Property Graph з додатковими семантичними зв’язками та 4-канальними ознаками вузлів (структурні, семантичні ембеддинги, метричні, еволюційні), включно з нормалізацією на рівні проєкту та інкрементальним оновленням.

Розроблено багаторівневу модель – на локальному рівні поєднано послідовнісний енкодер і гетерогенну GNN над підграфом CPG зі зв’язком span-to-node та fusion; на рівні компонента виконано агрегацію локальних дескрипторів; на проєктному рівні побудовано граф взаємодії компонентів і застосовано «message passing», а фінальне подання отримано через ієрархічну увагу.

Для прийняття рішень використано мультиміткову класифікацію з оцінюванням невизначеності (ентропія/енергія, епістемічна невизначеність) та механізмом селективного передбачення або відмови; формалізовано відповідні функції втрат. Також описано конвеєр застосування моделі в індустріальних умовах (у тому числі, для pull request), засоби інтерпретованості та експериментальну перевірку на мультимовному корпусі з порівнянням з базовими та абляційними варіантами.

Таким чином, сформульовано **перший пункт наукової новизни:** *вперше* запропоновано багаторівневу модель виявлення антипатернів на основі гібридного графа, що уніфікує синтаксичні дерева, графи потоку керування і графи залежностей у граф властивостей коду, з додатковими семантичними та еволюційними ознаками. Запропонована модель дозволяє виявляти типові і нетипові антипатерни, працювати з багатомовними програмними проєктами і забезпечувати сумісність із процесами безперервної інтеграції та доставлення.

В **третьому розділі** дисертаційної роботи запропоновані моделі та методи рекомендації, планування та процесного оцінювання рефакторингів для усунення антипатернів.

Для безпечного, прозорого автоматизованого усунення дублікатів коду із прогнозованим впливом на процес був запропонований метод рекомендації рефакторингів через багатоцільову оптимізацію з оцінкою невизначеності, спеціалізований на усуненні дублікатів коду

Ідея методу полягає в тому, що для кожної групи дублікатів розглядається кілька можливих рефакторингів. Кожен варіант оцінюється одночасно за трьома цілями – вигода, трудомісткість, CI-ризик, причому разом із прогнозом обчислюється невизначеність. Далі через багатоцільову (Парето) оптимізацію відбираються найкращі компромісні рішення, а за високої невизначеності система переходить у low-confidence/open-set режим і може утриматися від рекомендації.

Робота запропонованого методу складається з наступних кроків:

1) виявлення клонів – формуються групи дублікатів коду (клон-класи) у проєкті;

2) генерація дій – для кожної групи задається набір кандидатних рефакторингів + варіант «No refactoring»;

3) гібридне подання – для фрагментів клонів будується CPG (AST/CFG/PDG) та фіксується контекст компонентів;

4) обчислення ознак – обчислюються структурні, семантичні, метричні та еволюційні ознаки, які агрегуються на рівні групи клонів;

5) прогноз ефектів – для кожної пари «група–дія» оцінюються вигода, трудомісткість і CI-ризик;

6) невизначеність – через ймовірнісну модель отримуються дисперсія (ентропія) та калібрована впевненість прогнозів;

7) low-confidence фільтр – кандидати з високою невизначеністю відсікаються або переводяться в ручний перегляд (open-set);

8) парето-відбір – серед решти будуються Парето-оптимальні рішення за (вигода, трудомісткість, ризик);

9) ранжування зі штрафом – парето-множина впорядковується з урахуванням політики проєкту та штрафу за невизначеність;

10) вихід/утримання – повертається пріоритизований список рекомендацій або рішення «утриматися», якщо безпечних варіантів немає.

Експериментальне дослідження показало, що запропонований метод показав високу якість вибору типу рефакторингу (Accuracy = 0,80 та F1macro = 0,76), а також високу якість пріоритизації (NDCG@5 = 0,76 і U@5 = 0,62) тобто метод формує більш корисні top-k рекомендації та підтримує «обережну» поведінку через відсікання невпевнених рішень.

Таким чином, сформульовано **другий пункт наукової новизни:** *удосконалено* метод рекомендації рефакторингів через багатоцільову оптимізацію з оцінкою невизначеності, спеціалізований на усуненні дублікатів коду. Використання методу забезпечує безпечне, прозоре автоматизоване усунення дублікатів коду із прогнозованим впливом на процес.

Для максимізації очікуваної користі від послідовності рефакторингів за умов обмежень інтерфейсних змін і бюджету безперервної інтеграції та

доставлення, був запропонований метод композиції «мінімально інвазивних» послідовностей рефакторингів на основі причинно-наслідкового виведення з історії репозиторію.

Ідея роботи методу полягає в тому, що він будує узгоджений план рефакторингів із набору кандидатів так, щоб максимізувати очікуваний (консервативний) приріст якості та одночасно обмежити інвазивність і ризик: враховуються бюджети змін, сумісність кроків, вплив на API, а також CI/CD-ризик. Невизначеність прогнозів використовується для «обережного» вибору – перевага надається діям із надійним ефектом, а небезпечні/невпевнені варіанти відсікаються або приводять до утримання від рекомендації.

Робота запропонованого методу складається з наступних кроків:

1) збір кандидатів у рефакторинги з їхніми оцінками і метаданими впливу;

2) опис поточного стану компонентів через метрики якості, антипатерни, еволюційні та CI/CD показники;

3) оцінка очікуваного внеска у покращення з урахуванням контексту компонента;

4) корекція ефекту за невизначеністю (перехід від точкової оцінки до нижньої довірчої межі);

5) формулювання багатоцільової задачі планування (максимізація корисності при мінімізації інвазивності/ризику та дотриманні бюджетів);

6) накладання обмеження сумісності та порядку і інваріанти коректності;

7) пошук найкращої допустимої послідовності через евристичний пошук із відсіканням гілок за бюджетами та ризиком;

8) формування плану або утримання від рекомендації, якщо безпечного рішення в межах обмежень не існує.

Результати експериментального дослідження показали, що запропонований метод забезпечив середній приріст якості 0,084, при середній інвазивності 0,036 та середній зміні частки невдалих збірок −0,004.

Таким чином, сформульовано **третій пункт наукової новизни:** *вперше* запропоновано метод композиції «мінімально інвазивних» послідовностей

рефакторингів на основі причинно-наслідкового виведення з історії репозиторію. Використання методу дозволяє максимізувати очікувану користь за умов обмежень інтерфейсних змін і бюджету безперервної інтеграції та доставлення.

Для зменшення хибних тривог та забезпечення керованого зворотного зв'язку для безперервної інтеграції та доставлення в процесі проведення рефакторингів, була запропонована модель оцінювання ефективності рефакторингів на основі статистичного контролю процесів із атрибуцією впливу.

Запропонована модель розглядає рефакторинг не як разову «подію до/після», а як інтервенцію у часовому процесі розробки, тому ефективність змін оцінюється на основі статистичного контролю процесів (SPC) на часових рядах метрик.

Її ключова ідея полягає в тому, щоб у CI/DevOps-умовах відрізняти реальний, стійкий ефект рефакторингу від природних коливань, шуму та супутніх змін (фіч, тестів, залежностей), і формалізовано робити один із трьох висновків: «покращення», «погіршення» або «недостатньо доказів» із контрольованим рівнем хибних сигналів.

У роботі це реалізується як двоконтурний моніторинг: одночасно відстежуються метрики якості коду та процесні CI-метрики, показники попередньо уніфікуються за напрямом «більше = краще», нормуються та агрегуються робастно, а для рядів із залежностями застосовується усунення автокореляції («prewhitening»), щоб зменшити хибні тривоги. Далі момент інтервенції фіксується, і на постінтервенційному відрізку SPC-методами (EWMA/CUSUM та їх узагальненнями) детектуються малі, але стійкі зсуви відносно базового режиму з межами, налаштованими під заданий рівень ризику; за потреби ефект додатково атрибутується з урахуванням коваріат (наприклад, churn/зміни тестів), після чого формується підсумковий вердикт і зворотний зв'язок для налаштування політик відбору «безпечних» рефакторингів у наступних ітераціях.

Експериментальне дослідження моделі показало, що найбільш результативним для виявлення зсувів у метриках CI-стабільності є підхід на базі CUSUM: порівняно з EWMA частка хибних тривог зменшується на 16,7%, чутливість зростає на 9,7%, імовірність детекції збільшується на 3,2%, а середній

час до сигналу скорочується на 32,1%, тобто небажані зсуви фіксуються суттєво швидше. Водночас варіант EWMA з бутстреп-межами демонструє консервативнішу поведінку – хибні тривоги також зменшуються на 16,7%, але ціною невеликого падіння чутливості на 2,8% та імовірності детекції на 2,1% і незначного збільшення часу до сигналу (+3,6%).

Таким чином, сформульовано **четвертий пункт наукової новизни:** *удосконалено* модель оцінювання ефективності рефакторингів на основі статистичного контролю процесів із атрибуцією впливу. Використання моделі забезпечує доказові рішення «покращення, погіршення, змін не виявлено» з заданим рівнем довіри, зменшує хибні тривоги та надає керований зворотний зв'язок для безперервної інтеграції та доставлення.

В **четвертому розділі** дисертаційної роботи проведено дослідження інтегрованого інструментального засобу.

Реалізовано та описано інтегрований інструментальний засіб для наскрізного керування антипатернами (end-to-end pipeline):

– побудова гібридного подання коду (CPG + семантичні ембеддинги + VCS-ознаки);

– виявлення антипатернів;

– генерація/багатокритеріальне ранжування рефакторингів (Парето + нижні довірчі межі з урахуванням невизначеності);

– планування мінімально інвазивної послідовності з бюджетами/ризик-обмеженнями;

– SPC-оцінювання ефекту (EWMA/CUSUM);

– формування інженерних артефактів для відтворюваності і трасованості, включно з механізмом утримання за низької впевненості.

Експериментальне дослідження проведено на 7 Java-проєктах (JUnit5, Commons Lang, Guava, Spring Boot, Elasticsearch, Hadoop, Jenkins) у фіксованих релізних зрізах із контрольованим середовищем і порівнянням з baseline-підходами. Отримано узагальнений end-to-end ефект: середнє зменшення Technical Debt Index (TDI) на 16,4%, приріст Maintainability Index (MI) у середньому на +5,66,

зниження Cognitive Complexity (CC) на 12,1% та зменшення Clone density на 32,0%. Узгодженість прогнозованого та фактичного ефектів є високою ($R^2 \approx 0{,}91$; розбіжності «Expected vs Actual» у межах приблизно від −1,6 до +0,3%).

У порівнянні з базовими засобами інтегрована система дає більший сукупний ефект (для класичних аналізаторів покращення близькі до нуля, для часткових рекомендаторів – помітно нижчі за інтегровані), а також забезпечує повний цикл «виявлення – план рефакторингів – підтвердження ефекту».

Таким чином, отримані результати підтверджують коректність і ефективність технічних рішень, запропонованих у дисертаційній роботі.

Розроблені в роботі моделі та методи, а також інструментальний засіб отримав впровадження у діяльності НВП «Каре» та знайшли відображення у навчальному процесі та науково-дослідницькій діяльності Національного університету «Одеська політехніка».

Ключові слова: комп’ютерні науки, машинне навчання, інтелектуальний аналіз даних, моделі та методи аналізу даних, аналіз складних структур даних, графові та структурні моделі, оцінювання невизначеності, прийняття рішень в умовах невизначеності.

Список публікацій здобувача за темою дисертації

1) Годовиченко М.А.; Курінько Д.Д. «Аналіз існуючих підходів до автоматизації рефакторингу об’єктно-орієнтованих програмних систем» Publ. Nauka i Tekhnika. Odesa: Ukraine. Herald of Advanced Information Technology 8 (2), 179-196. https://doi.org/10.15276/hait.08.2025.11. (Index Scopus).

https://hait.op.edu.ua/index.php/journal/article/view/187

2) Курінько Д.Д. Модель оцінювання ефективності рефакторингів на основі статистичного контролю // Наука і техніка сьогодні. Серія «Техніка». – 2025. – Т. 50, № 9. – С. 1265–1280. – DOI: https://doi.org/10.52058/2786-6025-2025-9(50)-1265-1280. (Index Copernicus)

https://perspectives.pp.ua/index.php/nts/article/view/29453

3) Курінько Д.Д.; Кривда В.І. «Рекомендація рефакторингів із багатоцільовою оптимізацією та урахуванням невизначеності для дублювання коду» Publ. Nauka i Tekhnika. Odesa: Ukraine. Herald of Advanced Information Technology 8 (3), 301–315. https://doi.org/10.15276/hait.08.2025.19 (Index Scopus).

https://hait.op.edu.ua/index.php/journal/article/view/209

4) Курінько Д.Д. «Гібридні графи для запахів коду: багаторівнева модель виявлення антипатернів у програмних компонентах» Publ. Nauka i Tekhnika. Odesa: Ukraine. AAIT 8 (3), 274–285. https://doi.org/10.15276/aait.08.2025.18 (Index Copernicus).

https://aait.op.edu.ua/index.php/journal/article/view/201

5) Курінько Д.Д.; Кривда В.І. «Від комітів до причинності: побудова маловпливових послідовностей рефакторингів черезпричинно-наслідковий аналіз репозиторіїв»// Наука і техніка сьогодні. Серія «Техніка». – 2025. – Т. 52, № 11. – С. 1752–1773. – DOI: https://doi.org/10.52058/2786-6025-2025-11(52)-1752-1773 . (Index Copernicus).

https://perspectives.pp.ua/index.php/nts/article/view/32491

6) Курінько Д.Д. Інтегрований підхід до виявлення рефакторингу в ООП-системах // Modern The Integration of Research, Innovation and Economy : Proceedings of the 1st International Scientific and Practical Conference. – Seville, Spain : International Scientific Unity, October 8–10, 2025. – P. 26–30.

https://isu-conference.com/en/archive/the-integration-of-research-innovation-and-economy-08-10-25/

7) Курінько Д.Д., Кривда В.І. Мультимодальні графові подання для надійного виявлення антипатернів в еволюційних кодових базах // Інформатика. Культура. Техніка. – 2025. – Т. 2, № 2. – С. 294–299. – DOI: https://doi.org/10.15276/ict.02.2025.45 .

https://ict.op.edu.ua/index.php/journal/article/view/56

8) Курінько Д.Д. Проблеми виявлення потреби у рефакторингу в об’єктно-орієнтованому коді // Innovative Research in Science and Economy : Proceedings of the

2nd International Scientific and Practical Conference. – Brussels, Belgium : International Scientific Unity, December 3–5, 2025. – P. 764–767.

https://isu-conference.com/en/archive/innovative-research-in-science-and-economy-03-12-25/

9) Курінько Д.Д. Маловпливові послідовності рефакторингів: причинно-орієнтований підхід до аналізу історій репозиторіїв // Abstracts of XV International Scientific and Practical Conference. – Sofia, Bulgaria. – P. 264–267.

https://eu-conf.com/en/events/the-impact-of-modern-digital-technologies-on-the-future-of-professions/

10) Курінько Д.Д. Селективне виявлення антипатернів з open-set-головками та оцінкою невизначеності // Proceedings of the 5th International Scientific and Practical Conference. – Liverpool, United Kingdom : Cognum Publishing House, 2025. – P. 249–252.

https://sci-conf.com.ua/v-mizhnarodna-naukovo-praktichna-konferentsiya-modern-science-trends-challenges-solutions-11-13-12-2025-liverpul-velikobritaniya-arhiv/

11) Курінько Д.Д. Інкрементальне оновлення Code Property Graph для прискорення виявлення антипатернів у CI/CD-конвеєрах // Abstracts of XVI International Scientific and Practical Conference. – Munich, Germany. – P. 283–287.

https://eu-conf.com/en/events/conceptual-framework-and-dynamics-of-the-development-of-science/

ABSTRACT

Kurinko D.D. Machine Learning Models and Methods for Detecting and Eliminating Anti-patterns in Software Components. – Qualification scientific work in the form of manuscript.

Thesis for the PhD degree in specialty 122 Computer science. – Odessa Polytechnic National University, Ministry of Education and Science of Ukraine, Odesa, 2026.

The **introduction** highlights the relevance of solving problems related to the automated detection of the need for refactoring object-oriented software code in modern software systems. It highlights the shortcomings of existing models and methods for identifying refactoring opportunities, in particular approaches based on code "smells," software metrics, and machine learning methods that do not take into account the context of the project, the interdependence of features, and data noise. The object, subject, purpose, tasks, and methods of the study are defined; the scientific novelty and practical significance of the results obtained are presented; the personal contribution of the applicant is highlighted.

The **first chapter** of the dissertation provides a systematic review of the issues of identifying and eliminating anti-patterns in software components as a key factor in reducing internal quality and complicating the evolution of industrial software systems. It shows how the growth in the scale and complexity of software leads to the accumulation of design defects, and anti-patterns are associated with degradation of maintainability, complication of modifications, and increased development costs. On this basis, the task of automatic detection and elimination of anti-patterns is formulated, and the place of such solutions in the modern engineering process is outlined.

Next, a classification of anti-patterns is presented and it is shown that their manifestations are not limited to one type of signal: structural, semantic, and evolutionary (historical) features are generalized, and the multimodality of manifestations and the associated problem of uncertainty are emphasized. After that, traditional approaches (metrics, rules, detectors) are analyzed, and structural and text/LLM-oriented approaches

are considered separately, with a focus on their strengths and limitations from the point of view of practical application.

Modern approaches based on machine and deep learning were analyzed separately: classic models based on metrics and smell tags, graph and representation models of code representation, hybrid solutions and uncertainty-aware approaches, as well as areas where these approaches are used to support refactoring and working with code duplicates were described.

An analysis of works in the field of automatic anti-pattern removal showed that existing models tend to rely on a single type of feature and do not reflect the multidimensional nature of anti-patterns, which causes false positives or false negatives.

To overcome this shortcoming, a multi-level model based on a hybrid graph representation of code was proposed, combining structural, semantic, and evolutionary features in a single space.

Further analysis of existing anti-pattern detection systems showed that detection at the method/class level does not detect component/system anti-patterns and does not take into account the dynamics of quality between releases.

To address this shortcoming, a multi-level representation and analysis was proposed, taking into account the relationships between levels, as well as a method of composing "minimally invasive" refactoring sequences that considers the cause-and-effect relationships of changes.

An analysis of current approaches to identifying anti-patterns has shown that they are developing in isolation, which can lead to conflicting conclusions and recommendations. It has been shown that the use of hybrid representations that integrate features into a single vector is promising.

As such a solution, the paper proposed a unified hybrid representation as a common basis for both anti-pattern detection and refactoring plan optimization.

Further analysis of anti-pattern detection methods showed that their work is based on the assumption that all types of anti-patterns are known in advance, which does not correspond to modern software development practices. In addition, most methods do not

provide a calibrated assessment of prediction uncertainty, which increases the risk of aggressive or, conversely, overly cautious decisions.

To overcome these shortcomings, the paper proposes the introduction of an open-set/low-confidence mode in anti-pattern detection models and explicit modeling of uncertainty when recommending refactorings.

An analysis of the practice of implementing anti-pattern detection tools has shown that their results are poorly integrated into the work of teams and modern practices of continuous integration and delivery.

To overcome this shortcoming, the paper proposes a refactoring recommendation module, a method for composing sequences of changes, and a statistical process control model that allows for formalized evaluation of the effectiveness of interventions and reduces the risk of wrong decisions.

Based on the results of the analysis, the research goals and objectives were formulated.

The **second chapter** of the dissertation clarifies the task of automatic detection of anti-patterns, taking into account their multimodal nature (structure, semantics, metrics, evolution), multi-level context (method – component – project), and the need for an open-set/low-confidence mode with explicit consideration of uncertainty. The limitations of rules/metrics and single-modal approaches based on machine learning are justified, and requirements for a new model (portability, incrementality for continuous integration and delivery) are formulated.

A hybrid graph representation of code in the form of a Code Property Graph with additional semantic links and 4-channel node features (structural, semantic embeddings, metric, evolutionary) is proposed, including normalization at the project level and incremental updating.

A multi-level model has been developed: at the local level, a sequential encoder and a heterogeneous GNN over a CPG subgraph with span-to-node and fusion connections are combined; at the component level, local descriptors are aggregated; at the project level, a component interaction graph is constructed and message passing is applied, and the final representation is obtained through hierarchical attention.

For decision-making, multi-label classification with uncertainty estimation (entropy/energy, epistemic uncertainty) and a selective prediction/rejection mechanism are used; the corresponding loss functions are formalized. The application pipeline in industrial conditions (including for pull requests), interpretability tools, and experimental verification on a multilingual corpus with comparison to baseline and ablation variants are also described.

Thus, **the first point of scientific novelty** is formulated: for the first time, a multilevel model for detecting anti-patterns based on a hybrid graph is proposed, which unifies syntactic trees, control flow graphs, and dependency graphs into a code property graph, with additional semantic and evolutionary features. The proposed model allows detecting typical and atypical anti-patterns, working with multilingual software projects, and ensuring compatibility with continuous integration and delivery processes.

The **third section** of the dissertation proposes models and methods for recommending, planning, and process-based evaluation of refactorings to eliminate anti-patterns.

For safe, transparent, automated elimination of code duplicates with a predictable impact on the process, a method for recommending refactorings through multi-objective optimization with uncertainty assessment, specialized in eliminating code duplicates, was proposed.

The idea behind the method is that several possible refactorings are considered for each group of duplicates. Each option is evaluated simultaneously according to three objectives–benefit, labor intensity, and CI risk – and uncertainty is calculated along with the forecast. Next, the best compromise solutions are selected through multi-objective (Pareto) optimization, and in case of high uncertainty, the system switches to low-confidence/open-set mode and may refrain from making a recommendation.

The proposed method consists of the following steps:

1) clone detection – groups of code duplicates (clone classes) are formed in the project;

2) action generation – for each group, a set of candidate refactorings + the "No refactoring" option is specified.

3) hybrid representation – a CPG (AST/CFG/PDG) is constructed for clone fragments and the context of components is fixed;

4) feature calculation – structural, semantic, metric, and evolutionary features are calculated and aggregated at the clone group level;

5) effect prediction – for each "group–action" pair, the benefit, labor intensity, and CI risk are evaluated;

6) uncertainty – dispersion (entropy) and calibrated confidence of predictions are obtained through a probabilistic model;

7) low-confidence filter – candidates with high uncertainty are cut off or transferred to manual review (open-set);

8) Pareto selection – Pareto-optimal solutions are constructed among the rest based on (benefit, labor intensity, risk);

9) Ranking with penalty – the Pareto set is sorted taking into account the project policy and the penalty for uncertainty;

10) exit/hold – a prioritized list of recommendations or a "hold" decision is returned if there are no safe options.

Experimental research has shown that the proposed method demonstrated high quality of refactoring type selection (Accuracy=0.80 and F1macro=0.76), as well as high quality of prioritization (NDCG@5=0.76 and U@5=0.62), i.e., the method generates more useful top-k recommendations and supports "cautious" behavior by cutting off uncertain decisions.

Thus, the **second point of scientific novelty** is formulated: the refactoring recommendation method has been improved through multi-objective optimization with uncertainty estimation, specialized in eliminating code duplicates. The use of the method ensures safe, transparent, automated elimination of code duplicates with a predictable impact on the process.

To maximize the expected benefits of a sequence of refactorings under the constraints of interface changes and the budget for continuous integration and delivery, a method was proposed for composing "minimally invasive" sequences of refactorings based on causal inference from repository history.

The idea behind the method is that it builds a coordinated refactoring plan from a set of candidates in order to maximize the expected (conservative) quality gain while limiting invasiveness and risk: change budgets, compatibility of steps, impact on the API, and CI/CD risk are taken into account. The uncertainty of forecasts is used for "cautious" selection–preference is given to actions with a reliable effect, while dangerous/uncertain options are cut off or lead to refraining from a recommendation.

The proposed method consists of the following steps:

1) collecting refactoring candidates with their assessments and impact metadata;

2) describing the current state of components through quality metrics, anti-patterns, evolutionary and CI/CD indicators;

3) assessing the expected contribution to improvement, taking into account the context of the component;

4) correcting the effect for uncertainty (moving from a point estimate to a lower confidence limit);

5) formulation of a multi-objective planning task (maximizing utility while minimizing invasiveness/risk and adhering to budgets);

6) imposing compatibility and order constraints and correctness invariants;

7) searching for the best acceptable sequence through heuristic search with pruning branches by budget and risk.

8) forming a plan or refraining from making a recommendation if there is no safe solution within the constraints.

The results of the experimental study showed that the proposed method provided an average quality increase of 0.084, with an average invasiveness of 0.036 and an average change in the proportion of failed builds of −0.004.

Thus, **the third point of scientific novelty** is formulated: for the first time, a method for composing "minimally invasive" refactoring sequences based on causal inference from the repository history is proposed. Using this method allows maximizing the expected benefit under the constraints of interface changes and the budget for continuous integration and delivery.

To reduce false alarms and ensure controlled feedback for continuous integration and delivery during refactoring, a model for evaluating the effectiveness of refactoring based on statistical process control with attribution of impact has been proposed.

The proposed model considers refactoring not as a one-time "before/after" event, but as an intervention in the development process over time, so the effectiveness of changes is assessed based on statistical process control (SPC) on time series metrics.

Its key idea is to distinguish, in CI/DevOps conditions, the real, sustainable effect of refactoring from natural fluctuations, noise, and accompanying changes (features, tests, dependencies), and formally draw one of three conclusions: improvement, deterioration, or insufficient evidence with a controlled level of false signals.

In practice, this is implemented as dual-loop monitoring: code quality metrics and CI process metrics are tracked simultaneously, indicators are pre-unified according to the principle "more = better," normalized and aggregated robustly, and for series with dependencies, autocorrelation removal (prewhitening) is applied to reduce false alarms. Next, the moment of intervention is recorded, and in the post-intervention segment, SPC methods (EWMA/CUSUM and their generalizations) detect small but persistent shifts relative to the baseline mode with limits set for a given risk level; if necessary, the effect is additionally attributed taking into account covariates (e.g., churn/test changes), after which a final verdict and feedback are formed to adjust the policies for selecting "safe" refactorings in subsequent iterations.

An experimental study of the model showed that the most effective approach for detecting shifts in CI stability metrics is the CUSUM-based approach: compared to EWMA, the false alarm rate decreases by 16.7%, sensitivity increases by 9.7%, the probability of detection increases by 3.2%, and the average time to signal is reduced by 32.1%, meaning that unwanted shifts are detected significantly faster. At the same time, the EWMA variant with bootstrap limits demonstrates more conservative behavior–false alarms also decrease by 16.7%, but at the cost of a slight drop in sensitivity by 2.8% and detection probability by 2.1% and a slight increase in time to signal (+3.6%).

Thus, **the fourth point of scientific novelty** is formulated: the model for evaluating the effectiveness of refactorings based on statistical process control with attribution of influence has been improved. The use of the model provides evidence-based decisions of "improvement, deterioration, no changes detected" with a given level of confidence, reduces false alarms, and provides manageable feedback for continuous integration and delivery.

The **fourth section** of the dissertation examines an integrated tool.

An integrated tool for end-to-end pipeline management of anti-patterns has been implemented and described:

– construction of a hybrid code representation (CPG + semantic embeddings + VCS features);

– detection of anti-patterns;

– generation/multi-criteria ranking of refactorings (Pareto + lower confidence limits taking into account uncertainty);

– planning of a minimally invasive sequence with budgets/risk constraints;

– SPC effect evaluation (EWMA/CUSUM);

– formation of engineering artifacts for reproducibility and traceability, including a mechanism for retention at low confidence.

The experimental study was conducted on 7 Java projects (JUnit5, Commons Lang, Guava, Spring Boot, Elasticsearch, Hadoop, Jenkins) in fixed release slices with a controlled environment and comparison with baseline approaches. A generalized end-to-end effect was obtained: an average decrease in the Technical Debt Index (TDI) by 16.4%, an increase in the Maintainability Index (MI) by an average of +5.66, a decrease in Cognitive Complexity (CC) by 12.1%, and a decrease in Clone density by 32.0%. The consistency between the predicted and actual effects is high ($R^2 \approx 0.91$; "Expected vs Actual" discrepancies range from approximately −1.6 to +0.3%).

Compared to basic tools, the integrated system provides a greater cumulative effect (for classic analyzers, the improvements are close to zero, for partial recommenders, they are significantly lower than for integrated ones) and also provides a complete cycle of "detection – refactoring plan – confirmation of effect."

Thus, the obtained results confirm the correctness and effectiveness of the technical solutions proposed in the dissertation.

The models and methods developed in the work, as well as the tool, have been implemented in the activities of NVP "Kare" and are reflected in the educational process and research activities of the National University "Odessa Polytechnic".

Keywords: computer science, machine learning, intelligent data analysis, models and methods of data analysis, analysis of complex data structures, graph-based and structural models, uncertainty estimation, decision-making under uncertainty.

List of publications of the applicant on the topic of the dissertation

1) Hodovychenko M. A., Kurinko D. D. Analysis of Existing Approaches to Automated Refactoring of Object-Oriented Software Systems // Herald of Advanced Information Technology. – Odesa : Nauka i Tekhnika, 2025. – Vol. 8, № 2. – P. 179–196. – DOI: https://doi.org/10.15276/hait.08.2025.11. (Index Scopus).

https://hait.op.edu.ua/index.php/journal/article/view/187

2) Kurinko D. D. A Model for Evaluating Refactoring Effectiveness Based on Statistical Process Control // Science and Technology Today. Series «Engineering». – 2025. – T. 50, № 9. – C. 1265–1280. – DOI: https://doi.org/10.52058/2786-6025-2025-9(50)-1265-1280. (Index Copernicus).

https://perspectives.pp.ua/index.php/nts/article/view/29453

3) Kurinko D. D., Kryvda V. I. Uncertainty-Aware Multi-Objective Refactoring for Code Duplication // Herald of Advanced Information Technology. – Odesa : Nauka i Tekhnika, 2025. – Vol. 8, № 3. – P. 301–315. – DOI: https://doi.org/10.15276/hait.08.2025.19. (Index Scopus).

https://hait.op.edu.ua/index.php/journal/article/view/209

4) Kurinko D. D. Hybrid Graphs for Code Smells: A Multi-Level Model for Anti-Pattern Detection in Software Components // Applied Aspects of Information Technology. – 2025. – Vol. 8, № 3. – P. 274–285. – DOI: https://doi.org/10.15276/aait.08.2025.18. (Index Copernicus).

https://aait.op.edu.ua/index.php/journal/article/view/201

5) Kurinko D. D., Kryvda V. I. From Commits to Causality: Constructing Low-Impact Refactoring Sequences via Repository-Based Causal Inference // Science and Technology Today. Series «Engineering». – 2025. – T. 52, № 11. – C. 1752–1773. – DOI: https://doi.org/10.52058/2786-6025-2025-11(52)-1752-1773. (Index Copernicus).

https://perspectives.pp.ua/index.php/nts/article/view/32491

6) Kurinko D. D. An Integrated Approach to Detecting Refactoring Needs in OOP Systems // Modern The Integration of Research, Innovation and Economy : Proceedings of the 1st International Scientific and Practical Conference. – Seville, Spain : International Scientific Unity, October 8–10, 2025. – P. 26–30.

https://isu-conference.com/en/archive/the-integration-of-research-innovation-and-economy-08-10-25/

7) Kurinko D. D., Kryvda V. I. Multimodal Graph Representations for Reliable Anti-Pattern Detection in Evolving Codebases // Informatics. Culture. Technology. – 2025. – T. 2, № 2. – C. 294–299. – DOI: https://doi.org/10.15276/ict.02.2025.45.

https://ict.op.edu.ua/index.php/journal/article/view/56

8) Kurinko D. D. Challenges of Detecting the Need for Refactoring in Object-Oriented Code // Innovative Research in Science and Economy : Proceedings of the 2nd International Scientific and Practical Conference. – Brussels, Belgium : International Scientific Unity, December 3–5, 2025. – P. 764–767.

https://isu-conference.com/en/archive/innovative-research-in-science-and-economy-03-12-25/

9) Kurinko D. D. Low-Impact Refactoring Sequences: A Causality-Oriented Approach to Repository History Analysis // Abstracts of XV International Scientific and Practical Conference. – Sofia, Bulgaria. – P. 264–267.

https://eu-conf.com/en/events/the-impact-of-modern-digital-technologies-on-the-future-of-professions/

10) Kurinko D. D. Selective Anti-Pattern Detection with Open-Set Heads and Uncertainty Estimation // Proceedings of the 5th International Scientific and Practical

Conference. – Liverpool, United Kingdom : Cognum Publishing House, 2025. – P. 249–252.

https://sci-conf.com.ua/v-mizhnarodna-naukovo-praktichna-konferentsiya-modern-science-trends-challenges-solutions-11-13-12-2025-liverpul-velikobritaniya-arhiv/

11) Kurinko D. D. Incremental Updating of the Code Property Graph to Accelerate Anti-Pattern Detection in CI/CD Pipelines // Abstracts of XVI International Scientific and Practical Conference. – Munich, Germany. – P. 283–287.

https://eu-conf.com/en/events/conceptual-framework-and-dynamics-of-the-development-of-science/

ЗМІСТ

ВСТУП

Сучасна індустрія програмного забезпечення розвивається в умовах швидких релізних циклів, DevOps/CI/CD-практик і постійного ускладнення архітектур (багатомодульні та багатомовні системи, мікросервіси, розподілені компоненти). За таких умов ключовим чинником конкурентоспроможності стає не лише швидкість доставки функціоналу, а і здатність підтримувати кодову базу в стані, придатному до безпечної еволюції. Накопичення технічного боргу та поява антипатернів у програмних компонентах (надмірна зв'язаність, слабка когезія, дублікати коду, «розмиті» відповідальності тощо) призводять до зростання вартості змін, деградації підтримуваності, підвищення ризику регресій, збільшення часу CI-перевірок і зниження стабільності релізів. У результаті організації стикаються з проблемою – кожна наступна модифікація потребує більше часу, а ймовірність помилок і непередбачуваного впливу на систему зростає.

Практикою, що дозволяє стримувати деградацію дизайну, є рефакторинг. Проте визначення того, де саме і коли доцільно виконувати рефакторинги, залишається нетривіальним завданням, особливо у великих кодових базах із різнорідними технологіями та активною історією змін. Існуючі підходи часто спираються на ізольовані індикатори (метрики, «запахи» коду, окремі історичні сигнали), які є контекстно залежними та шумними; внаслідок цього зростає кількість хибних спрацювань і знижується довіра розробників до автоматизованих рекомендацій. Додатковою проблемою є обмежена переносимість моделей між проєктами/мовами, недостатня пояснюваність «чорних скриньок» та складність прогнозування реального ефекту рефакторингу для процесів CI/CD (ризик збільшення тривалості конвеєрів, порушення інтерфейсів, нестабільність тестів). Такі бар'єри стримують впровадження інтелектуальних засобів підтримки рефакторингу в практичні робочі процеси та актуалізують потребу у нових, більш надійних і доказових підходах до виявлення потреби в рефакторингу та оцінювання його наслідків.

Таким чином, існує протиріччя між, з одного боку, зростаючим запитом на автоматизоване, масштабоване та безпечне виявлення й усунення антипатернів у програмних компонентах із урахуванням багатовимірної природи коду та вимог CI/CD, і, з іншого боку, недостатністю наявних методів, які або не враховують семантику та еволюцію кодової бази, або не забезпечують керованої невизначеності, причинно-наслідкової обґрунтованості рішень та доказового контролю ефекту рефакторингів у процесі розробки.

У зв’язку з цим актуальними є дослідження, спрямовані на розроблення моделей та методів машинного навчання, що забезпечують багаторівневе виявлення типових і нетипових антипатернів на основі структурних, семантичних та еволюційних ознак, формують прозорі й безпечні рекомендації рефакторингів з урахуванням багатоцільових компромісів і невизначеності, будують «мінімально інвазивні» послідовності перетворень з огляду на причинно-наслідкові залежності в історії репозиторію та обмеження CI/CD, а також надають доказове оцінювання ефективності рефакторингів методами статистичного контролю процесів із атрибуцією впливу.

Метою дослідження є підвищення ефективності виявлення та усунення антипатернів у програмних компонентах шляхом розроблення моделей і методів машинного навчання для багаторівневого аналізу коду, безпечного формування рекомендацій і послідовностей рефакторингів з урахуванням невизначеності та обмежень CI/CD, а також доказового оцінювання їх ефекту методами статистичного контролю процесів.

Для досягнення поставленої мети в роботі передбачено виконання таких завдань:

– провести аналіз сучасних підходів до автоматизованого виявлення антипатернів і рекомендації рефакторингів, визначити їх обмеження в багатомовних/багатомодульних проєктах та в умовах CI/CD;

– сформувати вимоги до інтелектуальної системи виявлення й усунення антипатернів з урахуванням багатовимірної природи коду (структура, семантика, еволюція) та необхідності роботи з невизначеністю;

– розробити багаторівневу модель виявлення антипатернів на основі гібридного графового подання коду з інтеграцією семантичних та еволюційних ознак і підтримкою виявлення типових/нетипових антипатернів;

– удосконалити метод рекомендації рефакторингів для усунення дублікатів коду на основі багатоцільової оптимізації з оцінкою невизначеності та прогнозуванням впливу на процес розробки;

– розробити метод композиції «мінімально інвазивних» послідовностей рефакторингів на основі причинно-наслідкового виведення з історії репозиторію з урахуванням обмежень інтерфейсних змін і бюджету CI/CD;

– удосконалити модель оцінювання ефективності рефакторингів на основі статистичного контролю процесів із атрибуцією впливу для прийняття доказових рішень щодо ефекту змін;

– розробити інструментальні засоби та програмний прототип, що реалізують запропоновані моделі й методи та інтегруються у типові процеси розробки/CI/CD;

– провести експериментальне дослідження запропонованих рішень, оцінити точність виявлення, якість рекомендацій, безпечність/інвазивність рефакторингів і їх вплив на метрики якості та показники CI/CD, порівнявши з базовими підходами.

Об’єктом дослідження є процеси аналізу якості програмних компонентів у життєвому циклі розроблення програмного забезпечення, пов’язані з виявленням антипатернів і прийняттям рішень щодо їх усунення в умовах еволюції кодової бази та застосування практик CI/CD.

Предметом дослідження є моделі та методи машинного навчання для багаторівневого виявлення антипатернів у програмних компонентах і формування безпечних рекомендацій рефакторингів з урахуванням невизначеності та оцінювання їх ефекту.

Методи дослідження. Для розв’язання поставлених задач використано методи аналізу та моделювання програмного коду на основі графових подань (AST/CFG/PDG та їх уніфікація у граф властивостей коду), методи машинного навчання і глибинного навчання для побудови моделей виявлення антипатернів із використанням семантичних ембеддингів та еволюційних ознак репозиторію, а

також методи оцінювання невизначеності прогнозів. Для формування рекомендацій рефакторингів застосовано багатоцільову оптимізацію, ранжування та обмежувальні стратегії для забезпечення «безпечності» і мінімальної інвазивності змін, з урахуванням причинно-наслідкових залежностей, отриманих з історії проєкту. Для перевірки результатів використано методи математичної статистики та експериментальної валідації, зокрема статистичний контроль процесів (CUSUM/EWMA), оцінювання ефекту інтервенцій і порівняння з базовими підходами за метриками якості коду та показниками CI/CD.

Наукова новизна отриманих результатів. Основний науковий результат полягає у розв'язанні важливої науково-практичної задачі підвищення ефективності виявлення та усунення антипатернів у програмних компонентах шляхом розроблення і вдосконалення моделей та методів машинного навчання для багаторівневого аналізу коду, формування безпечних рекомендацій і «мінімально інвазивних» послідовностей рефакторингів з урахуванням невизначеності та обмежень CI/CD, а також доказового оцінювання ефекту рефакторингів методами статистичного контролю процесів із атрибуцією впливу.

У рамках виконаних досліджень отримано такі наукові результати:

– *вперше* запропоновано багаторівневу модель виявлення антипатернів на основі гібридного графа, що уніфікує синтаксичні дерева, графи потоку керування і графи залежностей у граф властивостей коду, з додатковими семантичними та еволюційними ознаками. Запропонована модель дозволяє виявляти типові і нетипові антипатерни, працювати з багатомовними програмними проєктами і забезпечувати сумісність із процесами безперервної інтеграції та доставлення;

– *удосконалено* метод рекомендації рефакторингів через багатоцільову оптимізацію з оцінкою невизначеності, спеціалізований на усуненні дублікатів коду. Використання методу забезпечує безпечне, прозоре автоматизоване усунення дублікатів коду із прогнозованим впливом на процес;

– *вперше* запропоновано метод композиції «мінімально інвазивних» послідовностей рефакторингів на основі причинно-наслідкового виведення з історії репозиторію. Використання методу дозволяє максимізувати очікувану

користь за умов обмежень інтерфейсних змін і бюджету безперервної інтеграції та доставлення;

– *удосконалено* модель оцінювання ефективності рефакторингів на основі статистичного контролю процесів із атрибуцією впливу. Використання моделі забезпечує доказові рішення «покращення, погіршення, змін не виявлено» з заданим рівнем довіри, зменшує хибні тривоги та надає керований зворотний зв'язок для безперервної інтеграції та доставлення.

Практичне значення одержаних результатів. Результати дисертаційного дослідження мають прикладне значення для підвищення якості та підтримуваності програмних систем у промисловій розробці. Запропоновані моделі й методи забезпечують надійніше виявлення антипатернів у багатомодульних і багатомовних кодових базах, а також формування безпечних «мінімально інвазивних» рекомендацій і послідовностей рефакторингів з урахуванням невизначеності прогнозів та обмежень CI/CD, що зменшує ризики регресій і небажаних змін.

Практичні результати можуть бути використані при створенні інструментальних засобів аналізу коду та підтримки рефакторингу, інтегрованих у середовища розробки й конвеєри безперервної інтеграції та доставлення. Доказове оцінювання ефекту рефакторингів методами статистичного контролю процесів надає керований зворотний зв'язок щодо результату змін («покращення», «погіршення», «змін не виявлено») і підвищує обґрунтованість рішень під час супроводження та розвитку програмного забезпечення.

Дисертацію виконано згідно тематичних планів НДР Національного університету «Одеська політехніка» за період 2024 – 2025 рр. в рамках держбюджетної теми № 237-177 – «Програмні системи машинного навчання та їх застосування в галузі інтелектуального аналізу даних».

Розроблені в дисертаційній роботі моделі та методи отримали впровадження у діяльності науково-виробничого підприємства «КАРЕ», а також знайшли відображення у навчальному процесі та науково-дослідницький діяльності Національного університету «Одеська політехніка».

Особистий внесок здобувача. Основні наукові положення, висновки і рекомендації, що містяться в дисертації та виносяться на захист, отримано здобувачем особисто.

В роботах [1], [8] виконано аналіз підходів до автоматизованого рефакторингу та проблеми виявлення потреби у ньому. У роботах [4], [7], [10], [11] запропоновано підходи до виявлення антипатернів на основі гібридних/мультимодальних графових подань коду, зокрема open-set режим та інкрементальне оновлення CPG для CI/CD. У роботах [3], [5], [9] розроблено методи рекомендації та побудови маловпливових послідовностей рефакторингів, зокрема для усунення дублікатів коду та з урахуванням причинно-наслідкових залежностей в історії репозиторію. У роботі [2] удосконалено модель оцінювання ефективності рефакторингів на основі статистичного контролю процесів із атрибуцією впливу. У матеріалах [6] висвітлено інтегрований підхід до виявлення рефакторингу в ООП-системах.

Апробація результатів дисертації. Основні положення і практичні результати дисертаційної роботи доповідалися та одержали схвалення на таких конференціях: XI міжнародна науково-практична конференція «Інформатика. Культура. Техніка» (м. Одеса, 2025 рік), а також на різних міжнародних науково-практичних онлайн-конференціях, які проходили у Іспанії, Бельгії, Болгарії, Великобританії та Німеччині.

Публікації. Основні результати дисертаційної роботи викладено в 11 публікаціях з них: 3 статті у наукових фахових виданнях України з технічних наук, 2 публікації у зарубіжних наукових періодичних виданнях з напряму, з якого підготовлено дисертацію, що індексується наукометричною базою SCOPUS, 6 публікацій у працях і матеріалах наукових конференцій.

Структура і обсяг роботи. Дисертація складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. Повний обсяг дисертації складає 308 сторінок, в тому числі: 240 сторінок основного тексту, 59 рисунків і 55 таблиць, список використаних джерел зі 212 найменувань і 5 додатків.

1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ВИЯВЛЕННЯ ТА УСУНЕННЯ АНТИПАТЕРНІВ У ПРОГРАМНИХ КОМПОНЕНТАХ

1.1 Значення внутрішньої якості програмного забезпечення та роль антипатернів у її зниженні

1.1.1 Еволюція промислових програмних систем

Сучасні промислові програмні системи еволюціонували від відносно компактних монолітних застосунків до довготривалих, масштабних і динамічних екосистем. На відміну від програмних продуктів 1990-х років із обмеженим функціональним обсягом і низькою частотою релізів, сучасні системи розвиваються десятки років, охоплюють мільйони рядків коду, сотні модулів і підтримуються географічно розподіленими командами. Емпіричні дослідження підтверджують стале зростання розміру кодових баз і кількості залежностей упродовж життєвого циклу програмних систем, що узгоджується із законом Лемана про постійну еволюцію програмного забезпечення (рис. 1.1) [1].

Для кількісного опису еволюції розміру кодової бази $S(t)$ у часі t розглядають інтегральну модель приросту, яка враховує інтенсивність змін коду:

$$S(t) = S_0 + \int_0^t v(\tau)d\tau, \tag{1.1}$$

де S_0 – початковий розмір системи;

$v(\tau)$ – швидкість приросту коду (code churn) у момент часу τ, що характеризує сумарну кількість доданих та вилучених рядків коду за одиницю часу.

Аналіз історій змін показує, що внутрішня структура навіть зрілих систем зазнає постійного переформатування, а значна частина коду активно змінюється протягом усього життєвого циклу [2].

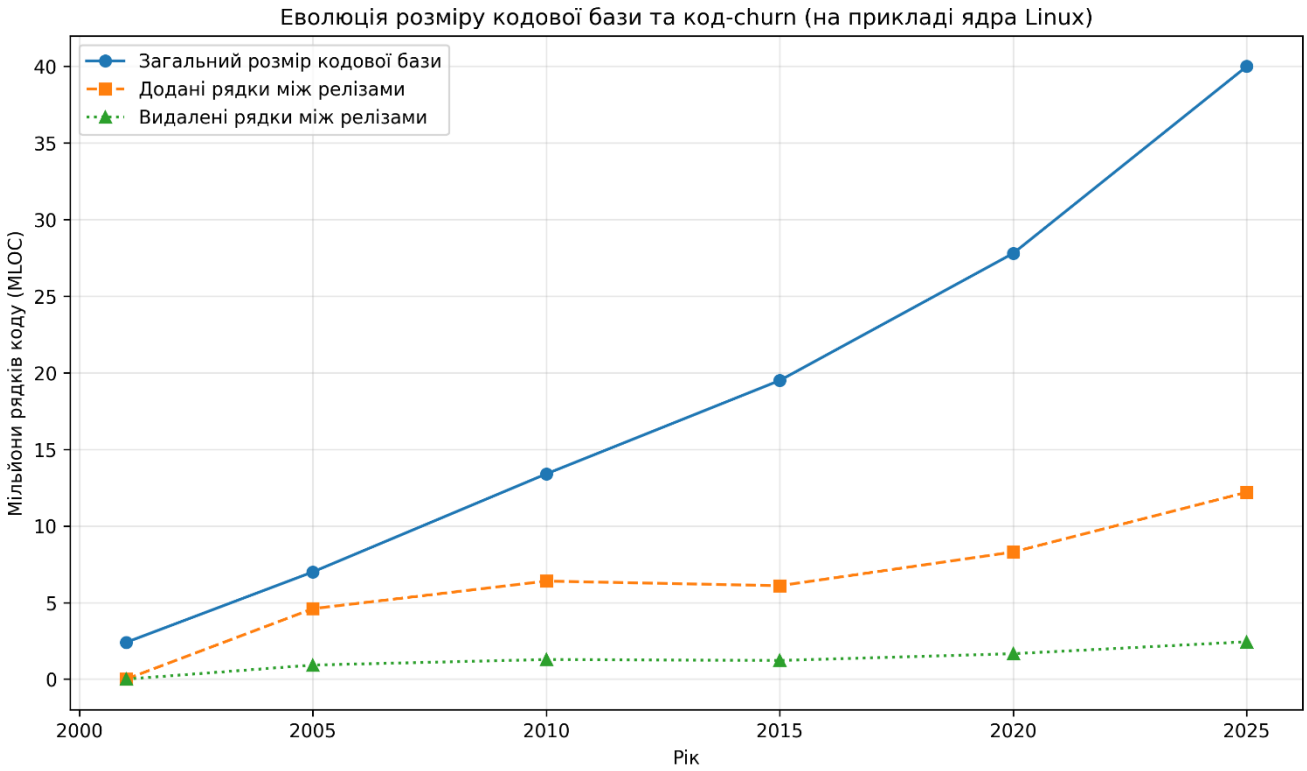

Рисунок 1.1 – Приклад еволюції розміру кодової бази та інтенсивності змін на основі відкритих оцінок для ядра Linux

Паралельно зі зростанням масштабів відбувається трансформація архітектурних стилів: від монолітів до сервіс-орієнтованих і мікросервісних архітектур. Для таких систем характерна мультимовна розробка, коли в межах одного продукту поєднуються кілька мов програмування та різні моделі доступу до даних [3–5]. Ще однією визначальною тенденцією є зростання швидкості змін, зумовлене впровадженням DevOps-практик, а також технологій CI/CD. Частоту деплойментів за період спостереження T можна формально описати як:

$$f_{dep} = \frac{N_{dep}}{T}, \tag{1.2}$$

де N_{dep} – кількість успішних деплойментів за період спостереження T.

На рівні коду цьому відповідає інтенсивність змін:

$$v_{churn} = \frac{L_{added} + L_{deleted}}{T}, \quad (1.3)$$

де L_{added} та $L_{deleted}$ – кількість доданих і вилучених рядків коду відповідно за період T.

Як показують емпіричні дані, у великих проєктах інтенсивність змін залишається високою протягом усього життєвого циклу [6]. Сукупність цих характеристик узагальнено в таблиці 1.1.

Таблиця 1.1 – Еволюція характеристик промислових програмних систем

Характеристика	«Класичні» системи (1990-ті – поч. 2000-х)	Сучасні промислові системи
Типова кількість мов у проєкті	1–2	3–10 і більше (поліглотні стек-и)
Розмір кодової бази	Сотні тисяч LOC	Мільйони / десятки мільйонів LOC
Архітектура	Моноліт / крупнозернисті модулі	SOA, мікросервіси, хмарні платформи
Частота релізів	Рази на рік	Від разів на тиждень до разів на день
Інтенсивність змін (churn)	Сплески на етапах релізів	Постійно високий рівень
Розподіленість команд	Локальні команди	Географічно розподілені команди

1.1.2 Внутрішня якість, антипатерни та їх вплив на еволюцію ПЗ

Внутрішня якість програмного забезпечення визначається сукупністю структурних характеристик – складністю, когезією, зв’язністю, рівнем дублювання та стабільністю еволюції компонентів. У моделі ISO/IEC 25010 ці характеристики безпосередньо пов’язані з підтримуваністю та здатністю системи до модифікації [7]. Порушення внутрішньої якості акумулюється у вигляді технічного боргу, коли короткострокові компроміси в проєктуванні призводять до зростання вартості супроводу та ризику дефектів на пізніших етапах [8, 9].

Дефектність компонентів часто описують через дефектну щільність:

$$D_{def} = \frac{N_{def}}{KLOC}, \tag{1.4}$$

де N_{def} – кількість виявлених дефектів;

$KLOC$ – обсяг коду в тисячах рядків

Рівень структурних проблем описують через щільність антипатернів і частку клонування коду:

$$D_{smell} = \frac{N_{smell}}{KLOC}, \qquad R_{clone} = \frac{LOC_{clone}}{LOC_{total}}, \tag{1.5}$$

де N_{smell} – кількість виявлених антипатернів;

LOC_{clone} – обсяг дубльованого коду;

LOC_{total} – загальний обсяг коду.

У низці емпіричних робіт показано, що ці показники корелюють із дефектністю та витратами на супровід [10–12]. Узагальнений вплив антипатернів і клонування на вартість супроводу може бути подано спрощеною регресійною моделлю, що відображає кумулятивний ефект деградації внутрішньої якості:

$$C_{maint} \approx C_0 + \beta_1 D_{smell} + \beta_2 R_{clone} + \varepsilon, \tag{1.6}$$

де C_{maint} – витрати на супровід;

C_0 – базовий рівень витрат;

β_1, β_2 – емпірично оцінені коефіцієнти впливу антипатернів і клонування відповідно;

ε – випадкова складова, що відображає інші фактори (тестове покриття, досвід команди тощо).

Основні зв’язки між внутрішніми чинниками якості, типовими антипатернами та їхніми наслідками для дефектності й супроводу систематизовано в таблиці 1.2.

Таблиця 1.2 – Внутрішні чинники якості та їхні наслідки

Чинник внутрішньої якості	Приклад антипатерну / клону	Типові наслідки для дефектності й супроводу
Висока цикломатична складність	Long Method, Complex Class	Зростання ймовірності помилок, складність розуміння й модифікації коду
Низька когезія, надмірна зв'язність	God Class, Feature Envy, Shotgun Surgery	Часті й широкі зміни, підвищений ризик побічних ефектів
Надмірне дублювання (клонування) коду	Клони типів I–III у кількох модулях	Множення обсягу змін, ризик неузгоджених виправлень у клонованих фрагментах
Нестабільна еволюція компонентів	Divergent Change, нестабільні «гарячі» модулі	Підвищена дефектність, складність планування релізів
Відсутність систематичного рефакторингу	Скупчення кількох антипатернів в одному класі	Накопичення технічного боргу, деградація архітектури

Антипатерни інтерпретуються як локальні симптоми неефективних проєктних рішень. Дослідження показують, що компоненти з накопиченням кількох антипатернів мають значно вищу схильність до змін і дефектів, ніж ізольовані випадки [10–12]. У термінах технічного боргу кожен новий антипатерн підвищує «відсоткову ставку» майбутніх змін, що концептуально ілюстровано на рисунку 1.2.

Дублювання (або клонування) коду є тісно пов'язаним із антипатернами чинником внутрішньої якості. За наявності n фрагментів-дублікатів, вартість узгодженої зміни можна оцінити як:

$$C_{change} \approx n \cdot c_{unit}, \tag{1.7}$$

де c_{unit} – середня вартість модифікації одного фрагмента.

Емпіричні дослідження підтверджують, що неузгоджена еволюція дублікатів є поширеним джерелом дефектів [13–15], навіть якщо не кожне дублювання безпосередньо призводить до помилок.

Рисунок 1.2 – Концептуальна схема впливу внутрішньої якості, антипатернів та клонування коду на технічний борг, дефектність, вартість супроводу та здатність системи до еволюції

1.1.3 Постановка задачі автоматичного виявлення та усунення антипатернів

Зростання масштабів і швидкості змін робить суто ручний контроль внутрішньої якості практично нереалізованим у промислових умовах [16, 17]. Це зумовлює необхідність автоматизованого виявлення антипатернів і дублікатів, інтегрованого в IDE та CI/CD-конвеєри.

Нехай $C = \{c_1, \dots, c_N\}$ – множина програмних компонентів, а $x_i \in \mathbb{R}^d$ – вектор їхніх ознак. Завдання виявлення антипатернів формалізується як побудова відображення

$$f_\theta : \mathbf{x}_i \mapsto \hat{\mathbf{y}}_i, \tag{1.8}$$

де $\hat{\mathbf{y}}_i$ – описує наявність та типи антипатернів (і, за потреби, дублікатів) у компоненті c_i – наприклад, як вектор імовірностей для множини smell-класів. Подальше прийняття рішень з порогами:

$$\hat{y}_{i,k} = \begin{cases} 1, \text{якщо } p_{i,k} \geq \tau_k \\ 0, \text{інакше} \end{cases}, \tag{1.9}$$

що породжує компроміс між точністю та повнотою [18, 19].

Автоматичне усунення антипатернів розглядається як задача вибору множини рефакторингів із урахуванням покращення внутрішньої якості, трудомісткості та ризиків CI/CD:

$$\mathbf{g}(r_{jk}) = (\Delta Q_{int}, \Delta R_{clone}, C_{impl}, R_{CI}), \tag{1.10}$$

де ΔQ_{int} – очікуване покращення внутрішньої якості (метрик, щільності антипатернів);

ΔR_{clone} – зміна рівня клонування;

C_{impl} – оцінка трудомісткості реалізації;

R_{CI} – ризик порушення збірки / тестів.

Далі відбувається оптимізація багатокритеріального функціонала

$$R^* \bigcup_j R_j, \tag{1.11}$$

Узагальнену схему такого конвеєра наведено на рисунку 1.3, а порівняння ручних і автоматизованих підходів – в таблиці 1.3.

Рисунок 1.3 – Узагальнена схема конвеєра автоматичного виявлення антипатернів і клонів та підтримки рефакторингу у промисловому процесі розробки

Таким чином, внутрішня якість, антипатерни та дублікати коду утворюють взаємопов’язану систему чинників, що визначає дефектність, вартість супроводу та еволюційну здатність програмних систем. Це обґрунтовує необхідність формальних моделей і методів автоматичного виявлення та усунення антипатернів, які розробляються в наступних розділах дисертації.

Таблиця 1.3 – Порівняння ручного та автоматизованого підходів до роботи з антипатернами та дублікатами коду

Аспект	Ручний аналіз	Автоматизований аналіз
Масштаб	Обмежений обсяг коду (критичні модулі, «проблемні» файли); повний огляд майже недосяжний	Аналіз усієї кодової бази, включно з малопомітними й рідко змінюваними компонентами
Послідовність	Залежність від досвіду та втоми експерта; висока варіативність результатів	Відтворюваність правил і моделей; стабільність критеріїв між релізами та командами
Вартість	Високі прямі витрати часу на рев’ю; складно масштабувати на великі репозиторії	Початкові витрати на налаштування та інтеграцію; низькі граничні витрати на повторні запуски
Типові помилки	Пропуски через неуважність, перевантаження інформацією; суб’єктивні оцінки	Хибнопозитивні спрацювання, залежність від якості правил/моделей і наборів навчальних даних
Придатність до CI/CD	Епізодичні рев’ю, прив’язані до ключових релізів	Регулярний запуск на кожному коміті / pull request, безперервний моніторинг технічного боргу
Пояснюваність результатів	Природні пояснення експерта, але без формалізації критеріїв	Формалізовані правила, метрики й моделі; потреба в додаткових механізмах пояснюваності для розробників
Адаптація під проєкт	Гнучка, але неявна (у головах експертів)	Явні конфігурації правил, порогів і профілів якості, що можуть версіонуватися та перевірятися

1.2 Класифікація антипатернів і їхні структурні, семантичні та еволюційні ознаки

1.2.1 Основні типи антипатернів

У науковій літературі антипатерни розглядаються як узагальнені патологічні шаблони, що виникають на різних рівнях програмної системи – від окремих фрагментів коду до архітектури та процесів розробки. З погляду внутрішньої якості доцільно виокремлювати три основні групи антипатернів [20–22]: кодові, архітектурні (та еволюційні), а також пов’язані з процесом розробки. Така стратифікація відображає різний масштаб і характер негативного впливу – від локального ускладнення підтримки окремих класів до деградації архітектурної цілісності та організаційної керованості проєкту.

Кодові антипатерни («запахи» коду, code smells) фіксують локальні порушення принципів об’єктно-орієнтованого проєктування, зокрема надмірну складність, низьку когезію, зайву зв’язність або дублювання логіки. До найбільш відомих і досліджених належать «Long Method», «God Class», «Feature Envy», «Duplicated Code», «Data Class», «Primitive Obsession» тощо [23–25]. Систематичні огляди показують, що саме «God Class», «Feature Envy» та «Long Method» є найчастіше підтримуваними сучасними інструментами аналізу, що корелює з їхнім негативним впливом на когезію, зв’язність і обчислювальну складність компонентів [24, 25]. У межах цієї дисертації кодові антипатерни розглядаються як базовий об’єкт автоматичного виявлення, оскільки вони безпосередньо проявляються у структурі програмних компонентів і піддаються формалізації через вимірювані ознаки.

Архітектурні та еволюційні антипатерни відображають порушення принципів модульності, ієрархії та керованої еволюції системи [20, 26]. До цієї групи належать «Cyclic Dependency», «Hub-like Dependency», «Shotgun Surgery» та «Divergent Change» [27, 28]. Такі антипатерни характеризуються не лише локальними структурними ознаками, а й характером змін у масштабі підсистем:

«Shotgun Surgery» проявляється через необхідність масових змін при одній бізнес-операції, тоді як «Divergent Change» – через накопичення несумісних причин змін в одному компоненті.

На макрорівні крайнім проявом архітектурної деградації є «Big Ball of Mud» – система з розмитою або відсутньою архітектурою, що еволюціонувала через послідовність короткострокових рішень [21, 22]. Емпіричні дослідження підтверджують, що архітектурні антипатерни не зводяться до простої суми «запахів» коду і потребують окремих методів аналізу [20, 26].

Антипатерни, пов'язані з процесом розробки, відображають системні проблеми в організації командної роботи, управлінні вимогами та технічним боргом. До них належать «Lava Flow», «Copy-and-Paste Programming», «Cargo Cult Programming», а також процесний вимір «Big Ball of Mud» [22, 29, 30]. Такі антипатерни, як правило, не можуть бути усунені виключно локальним рефакторингом і вимагають змін у процесах розробки – від практик code review та CI/CD до стратегій управління еволюцією системи. Узагальнене зіставлення основних груп антипатернів наведено в таблиці 1.4.

Таблиця 1.4 – Основні типи антипатернів на різних рівнях системи

Тип антипатерну	Типові приклади	Основний фокус порушення	Типові наслідки
Кодові (локальні)	Long Method, God Class, Feature Envy, Duplicated Code	Структура класів і методів, когезія, зв'язність	Зростання складності, дефектності, витрат на зміну
Архітектурні та еволюційні	Shotgun Surgery, Divergent Change, Cyclic Dependency, Big Ball of Mud	Структура модулів, залежності, розподіл відповідальностей	Ускладнена еволюція, крихкість системи, технічний борг
Пов'язані з процесом розробки	Lava Flow, Copy-and-Paste Programming, процесний вимір Big Ball of Mud	Організація роботи, керування вимогами й боргом	Хронічне накопичення проблем, хаотична еволюція

1.2.2 Структурні, семантичні та історичні ознаки антипатернів

Для формалізації антипатернів доцільно розглядати структурні, семантичні та історичні ознаки як доповнювані проєкції на кодову базу. У загальному випадку для програмного компонента c_i (клас, файл, метод) вводиться вектор ознак:

$$\mathbf{x}_i = \left[\mathbf{x}_i^{str}, \mathbf{x}_i^{sem}, \mathbf{x}_i^{hist}\right], \tag{1.12}$$

де $\mathbf{x}_i^{str}$ – структурні (метричні та графові) характеристики;

$\mathbf{x}_i^{sem}$ – семантичні ознаки (ідентифікатори, API-виклики, текст);

$\mathbf{x}_i^{hist}$ – історичні ознаки, отримані з системи контролю версій (VCS) [1–3].

Структурні ознаки базуються на метриках та графових поданнях коду. До них належать LOC, цикломатична складність, метрики Чидамбера–Кемерера (WMC, CBO, LCOM, RFC), а також показники структури пакетів і модулів [31–35]. Для типових антипатернів коду існують характерні профілі метрик: «Long Method» асоціюється з високими LOC і CC, тоді як «God Class» поєднує високу складність, зв’язність і низьку когезію. Для архітектурних антипатернів важливими є ознаки структури залежностей, наявність циклів та «централізованих» вузлів у графі модулів [32, 36].

Графи коду (AST, CFG, PDG, графи викликів і залежностей) дозволяють явно моделювати структуру, потоки керування та даних [32]. Вони є особливо інформативними для аналізу складних антипатернів, таких як «God Class» або «Shotgun Surgery», а також для структурного опису дублікатів коду через перекриття підграфів [35, 37].

Семантичні ознаки доповнюють структурний аналіз шляхом врахування змісту ідентифікаторів, сигнатур API, коментарів і шаблонів використання бібліотек [33, 38]. Вони дозволяють відрізняти великі, але концептуально однорідні компоненти від дійсно перевантажених відповідальністю. Сучасні підходи широко використовують векторні подання коду (ембеддинги), у яких семантично подібні фрагменти мають близькі вектори в $\mathbb{R}^d$ [38, 39].

Історичні ознаки, отримані з VCS, фіксують динаміку еволюції компонентів і є критично важливими для інтерпретації антипатернів [33, 40]. До них належать churn, частота змін, кількість розробників, частка bug-fix-комітів і показники спільної еволюції файлів (change coupling). Формально change coupling між файлами f_i та f_j можна подати як:

$$CC(f_i, f_j) = \frac{|\{commit\ k: f_i \in k \wedge f_j \in k\}|}{|\{commit\ k: f_i \in k\}|}, \tag{1.13}$$

Високі значення CC свідчать про приховану логічну зв'язність і часто асоціюються з антипатернами «Shotgun Surgery» або нестабільними архітектурними залежностями [41]. Узагальнення ролі структурних, семантичних та історичних ознак у виявленні антипатернів наведено в таблиці 1.5.

Таблиця 1.5 – Структурні, семантичні та історичні ознаки антипатернів

Тип ознак	Джерело даних	Типові приклади показників / подань	Приклади антипатернів, де ознака особливо корисна
Структурні	Метрики, AST, CFG, PDG, графи залежностей	LOC, CC, WMC, CBO, LCOM; ступінь вузла в графі викликів; наявність циклів у графі залежностей	Long Method, God Class, Complex Class, Cyclic Dependency
Семантичні	Імена ідентифікаторів, сигнатури методів, коментарі, API-виклики	Частоти термінів; векторні подання (ембеддинги) коду й тексту; шаблони використання API	Feature Envy, Data Class, Mixed Responsibilities
Історичні (VCS)	Журнали комітів, повідомлення, мітки дефектів	Churn, частота змін, кількість розробників, частка bug-fix комітів, change coupling	Shotgun Surgery, Divergent Change, нестабільні «гарячі» модулі

1.2.3 Мультимодальність ознак і проблема невизначеності

Мультимодальність означає, що рішення про наявність антипатерну формується як результат взаємодії різнорідних каналів ознак. Формально це можна подати як:

$$y_i = f\left(\mathbf{x}_i^{str}, \mathbf{x}_i^{sem}, \mathbf{x}_i^{hist}\right). \quad (1.14)$$

де кожна підгрупа ознак описує різний аспект внутрішньої якості. На практиці це призводить до появи конфліктних сигналів, коли, наприклад, структурні метрики вказують на можливий антипатерн, а історичні ознаки демонструють стабільність і низьку дефектність компонента, або навпаки.

Такі ситуації породжують невизначеність, яку доцільно розглядати як поєднання випадкової (шум у даних, неконсистентні мітки) та епістемічної (атипові або слабо представлені в навчальних даних випадки) складових.

У термінах ймовірнісного моделювання це проявляється у розбіжностях умовних розподілів $p(y \mid x^{str})$ та $p(y \mid x^{hist})$. Типові мультимодальні сценарії та їх інтерпретація наведені в таблиці 1.6.

Таблиця 1.6 – Приклади мультимодальних сценаріїв та їх інтерпретація

Сценарій	Структурний сигнал	Семантичний сигнал	Історичний сигнал	Рекомендована інтерпретація
Великий клас із високими значеннями LOC, WMC, CBO	Сильний сигнал про можливий God Class / Long Method (високі метрики розміру та складності)	Змішана семантика: ідентифікатори та API з кількох різних піддоменів	Високий churn, часті зміни, значна частка bug-fix-комітів	Узгоджений мультимодальний сигнал про наявність антипатернів God Class / Divergent Change; високий пріоритет для рефакторингу
Великий клас із високими метриками, але стабільний у часі	Сильний сигнал за метриками (великий, складний клас)	Семантика відносно однорідна, ідентифікатори належать до одного домену	Низький churn, рідкі зміни, незначна частка bug-fix-комітів	Можливий «центральний сервіс» або фасад; результат детектора слід трактувати з обережністю, бажаний ручний перегляд перед рефакторингом

Продовження таблиці 1.6

Сценарій	Структурний сигнал	Семантичний сигнал	Історичний сигнал	Рекомендована інтерпретація
Невеликий клас з низькою складністю, але «прив'язаний» до багатьох інших файлів	Метрики в нормі: невеликий розмір, низька CC, помірна зв'язність	Семантика спеціалізована, без явних ознак змішаних відповідальностей	Дуже високий change coupling з багатьма модулями, часто змінюється разом з ними	Кандидат на архітектурний або процесний антипатерн (наприклад, прихований координатор або «бутилочне горлечко»); доцільно аналізувати на рівні архітектури та процесу
Клас із помірними метриками, але неоднорідною семантикою	Легкий або відсутній структурний сигнал (розмір і складність у межах порогів)	Ідентифікатори й API з кількох несумісних піддоменів, змішання ролей	Середній churn, епізодичні bug-fix-коміти в різних зонах класу	Потенційний антипатерн Divergent Change або прихований God Class на ранній стадії; доцільний превентивний рефакторинг із розділенням відповідальностей
Компонент, який часто змінюється, але має «чисті» метрики та семантику	Метрики в нормі, відсутні явні кодові запахи	Семантика однорідна, чітка роль компонента	Дуже високий churn через часті зміни вимог, релізні «гарячі точки»	Імовірний процесний або доменний фактор (часті зміни бізнес-вимог); автоматичний детектор антипатернів має позначати як «зону ризику», але не як класичний «запах»

У промислових проєктах додаткову складність становлять атипові компоненти – генератори коду, «каркасні» класи фреймворків або доменно-специфічні модулі, які виходять за межі стандартних розподілів [42].

Для таких випадків доцільним є використання open-set підходів і методів виявлення аномалій, що дозволяють позначати компоненти як кандидати на ручний перегляд, а не примусово відносити їх до відомих категорій антипатернів [43, 44].

Концептуальну схему мультимодальної інтеграції та роботи з невизначеністю наведено на рисунку 1.4.

Таким чином, класифікація антипатернів і відповідних ознак повинна враховувати не лише їхню наявність, а й взаємодію між структурою, семантикою та історією змін. Це безпосередньо мотивує використання в дисертації

багаторівневих мультимодальних моделей і механізмів оцінювання невизначеності, що розглядаються в наступних розділах.

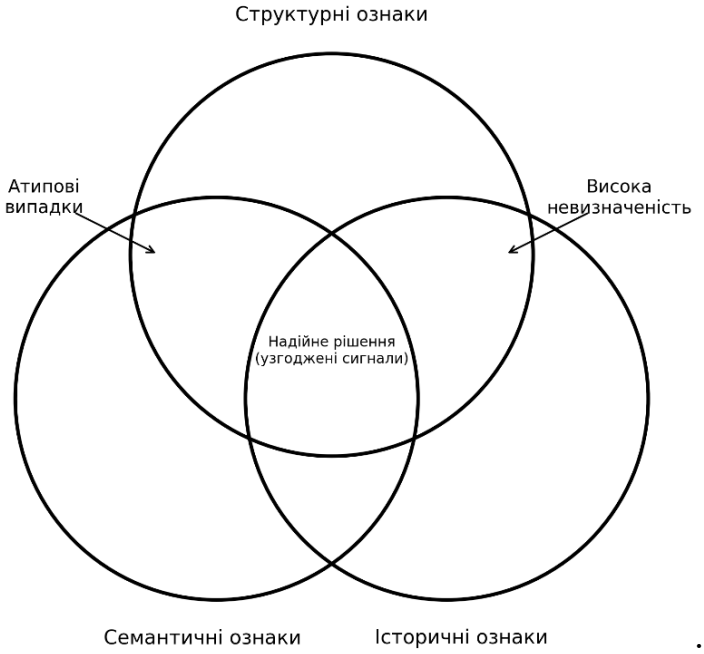

Рисунок 1.4 – Мультимодальна схема взаємодії структурних, семантичних та історичних ознак антипатернів

1.3 Традиційні методи виявлення та усунення антипатернів

У практиці забезпечення внутрішньої якості програмного забезпечення традиційні методи виявлення та усунення антипатернів історично ґрунтувалися на статичному аналізі коду, каталогах рефакторингів і вручну сформульованих евристиках.

Систематичні огляди показують, що впродовж тривалого часу домінували підходи, засновані на правилах та на метриках, до яких згодом додалися пошукові та машинно-навчальні методи, причому перші дві групи залишаються найбільш поширеними в промислових інструментах і зосереджені переважно на мовах сімейства Java (рис. 1.5).

У межах дисертаційного дослідження важливо узагальнити сильні й слабкі сторони класичних детекторів та засобів рефакторингу, проаналізувати їхню

чутливість до проєкту і мови програмування, а також виявити обмеження, які мотивують перехід до мультимодальних моделей і методів машинного навчання.

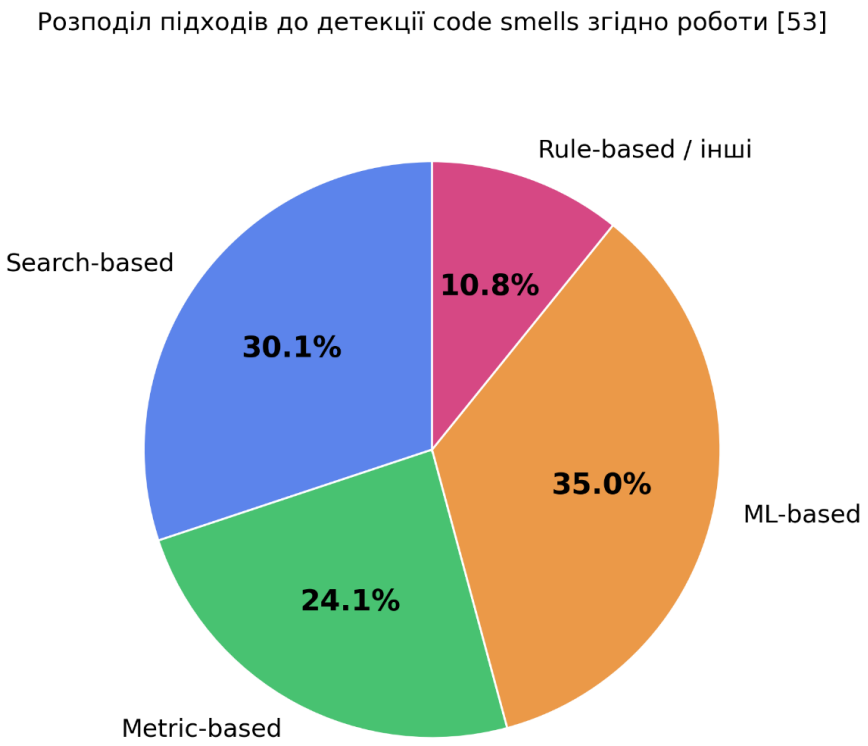

Рисунок 1.5 – Розподіл типів підходів до виявлення антипатернів

1.3.1 Детектори, засновані на метриках та правилах

Детектори, засновані на метриках та правилах, становлять базовий клас інструментів статичного аналізу й отримали найширше поширення в промисловій розробці ПЗ.

Їхня ідея полягає у формалізації неформальних «запахів» коду через набір вимірюваних метрик і евристичних правил, що застосовуються до синтаксичних дерев і графів викликів та залежностей [45–47]. У таких підходах антипатерн розглядається як відхилення від очікуваного профілю метрик розміру, складності, когезії та зв’язності.

У межах концепції стратегій виявлення правило для антипатерну S_k задається як булева функція над вектором метрик:

$$S_k(c) = \begin{cases} 1, \text{якщо } \varphi\big(\mathbf{m}(c)\big) = true \\ 0, \text{інакше} \end{cases}, \quad (1.15)$$

де $\mathbf{m}(c)$ – вектор метрик (LOC, WMC, CBO, LCOM тощо);

φ_k – композиція порівнянь з порогами та фільтрів.

Наприклад, спрощене правило для «God Class» може бути задане як комбінація перевищень порогів для WMC, CBO та LCOM [45, 48].

Такі підходи реалізовані в численних інструментах статичного аналізу – JDeodorant, iPlasma, InFusion, PMD, Checkstyle, SonarQube та їхніх розширеннях [46, 48, 49]. Їхніми ключовими перевагами є прозорість і пояснюваність правил, відсутність потреби в розмічених навчальних вибірках, добра масштабованість і проста інтеграція в IDE та CI/CD-процеси. Водночас численні емпіричні дослідження демонструють істотні обмеження детекторів, заснованих на правилах.

По-перше, вибір порогів метрик є значною мірою суб’єктивним і сильно залежить від конкретного проєкту, домену та стилю розробки [46, 50]. По-друге, різні інструменти, що декларують підтримку одного й того самого антипатерну, часто дають низькоузгоджені результати, що підтверджується невисокими значеннями коефіцієнтів узгодженості між ними [48, 50].

По-третє, спостерігається типовий компроміс між точністю та повнотою: підвищення чутливості детектора зазвичай призводить до зростання кількості хибнопозитивних спрацювань, що знижує довіру розробників до інструментів [46, 50].

Додатковою проблемою є висока чутливість детекторів, заснованих на правилах, до мови програмування та архітектурного стилю. Більшість досліджень і промислових інструментів орієнтовані на Java, тоді як для інших мов (Python, Kotlin, JavaScript) і платформ потрібна адаптація правил і повторне калібрування порогів [45, 49].

Узагальнення основних характеристик детекторів, заснованих на правилах та метриках, наведено в таблиці 1.7.

Таблиця 1.7 – Характеристики детекторів антипатернів, заснованих на правилах та метриках

Аспект	Переваги	Обмеження та чутливість
Інтерпретованість	Прості для пояснення правила (пороги метрик, логічні умови); відповідність відомим принципам	Суб'єктивність вибору порогів; неочевидні взаємодії між метриками
Вимоги до даних	Не потребують розмічених навчальних вибірок; працюють «із коробки»	Орієнтовані на ідеалізовані стилі та архітектури; слабка підтримка атипових і домен-специфічних випадків
Масштабованість	Ефективні на великих кодових базах; добре інтегруються у CI/CD	За надто агресивних конфігурацій створюють багато попереджень, що не масштабуються для review
Точність (precision/recall)	Прийнятні результати для окремих «запахів» у добре відкаліброваних проєктах	Низька узгодженість між різними інструментами; чутливість до конкретних систем і версій
Залежність від мови та проєкту	Добра підтримка для Java (та близьких мов), багатий інструментарій	Обмежена підтримка інших мов; правила потрібно адаптувати під архітектуру, домен і стиль команди

1.3.2 Структурні та текстові/LLM-підходи

Подальший розвиток методів виявлення антипатернів пов'язаний із використанням структурних і текстових подань коду. Структурні підходи базуються на графових моделях програми – AST, CFG, PDG та їх уніфікації у вигляді Code Property Graph (CPG), що поєднує синтаксис, керування та потоки даних у єдину багатошарову структуру [51–53]. Формально таке подання задається графом

$$G_{str} = (V, E, \tau), \tag{1.16}$$

де V – множина вузлів (інструкції, вирази, базові блоки);

E – множина ребер (синтаксичні, контрольні, дата-залежності);

τ – відображення, що позначає тип вузла/ребра (наприклад, «if-statement», «data-dependence», «call-edge»).

Структурні моделі забезпечують високу точність на рівні синтаксису й потоку виконання, дозволяють формулювати формальні запити до коду та слугують природною основою для графових нейронних мереж [54, 55]. Водночас вони погано враховують семантику ідентифікаторів, мають високу обчислювальну вартість (особливо для PDG) і суттєво залежать від мовної специфіки.

Паралельно розвивалися текстові та мовно-модельні підходи, які розглядають код як послідовність токенів і застосовують методи мовного моделювання, включно з великими мовними моделями (LLM) [56, 57]. Такі моделі здатні враховувати багатий семантичний контекст, працювати з кількома мовами та генерувати пояснення і пропозиції рефакторингу [58–60]. Проте вони мають обмежений контроль над точним потоком керування і потоком даних, чутливі до prompt-інжинірингу та пов’язані з високими обчислювальними витратами і ризиком «галюцинацій» [61, 62].

Порівняння структурних і текстових/LLM-підходів наведено в таблиці 1.8, з якої видно, що жоден із них окремо не забезпечує повного та надійного розв’язання задачі виявлення антипатернів у промислових системах.

Таким чином, традиційні методи виявлення антипатернів – від детекторів на базі правил до структурних і текстових моделей – відіграють важливу роль у практиці аналізу якості коду, однак кожен із них має суттєві обмеження. Відсутність інтегрованого подання, яке б одночасно враховувало структуру, семантику та історію змін, а також явного моделювання невизначеності рішень, обмежує їхню ефективність у складних промислових умовах. Саме ці обмеження слугують мотивацією для розроблення в подальших розділах дисертації мультимодальних моделей і методів машинного навчання для виявлення та усунення антипатернів.

Таблиця 1.8 – Порівняння структурних та текстових/LLM-підходів до виявлення антипатернів

Характеристика	Структурні (AST/CFG/PDG/CPG)	Текстові / LLM-підходи
Основне подання	Графи синтаксису, керування, залежностей	Послідовності токенів, ембеддинги коду та природної мови
Сильні сторони	Точність на рівні синтаксису й потоку; формальні запити; основа для GNN	Багатий семантичний контекст; мульти-мовність; пояснення та рефакторинг
Типові задачі	Вразливості, клонування, структурні й архітектурні антипатерни	Пошук коду, резюмування, детекція «запахів», рефакторинг, code review
Обмеження	Відсутність семантики імен; дорога побудова PDG; мовна специфіка	Нечіткий контроль потоку; залежність від prompt’ів; галюцинації
Масштабованість до великих систем	Хороша (за умови оптимізованої реалізації)	Обмежена; часто потрібен семплінг чи інкрементальний аналіз
Придатність до CI/CD	Висока для статичного аналізу	Поки що обмежена, зазвичай використовується точково

1.4 Сучасні підходи до виявлення антипатернів і підтримки рефакторингу на базі машинного навчання та глибинних моделей

Сучасний етап розвитку засобів виявлення антипатернів і підтримки рефакторингу характеризується переходом від жорстко фіксованих правил і метрик до моделей машинного та глибинного навчання, які орієнтовані на дані. На відміну від традиційних підходів, ML/DL-моделі навчаються безпосередньо на прикладах реального коду, його еволюції та супровідних артефактах (історія змін, дефекти, рефакторинги), що дозволяє автоматично виявляти складні нелінійні залежності між структурними, семантичними та історичними ознаками антипатернів. У цьому контексті застосовуються графові нейронні мережі над AST/CFG/PDG/CPG,

семантичні ембеддинги коду і природної мови, великі мовні моделі, а також гібридні підходи, які поєднують детекцію антипатернів із рекомендаціями або генерацією рефакторингів.

1.4.1 Класичні моделі машинного навчання над метриками і мітками «запахів» коду

Класичні ML-підходи до виявлення антипатернів ґрунтуються на поданні елементів коду у вигляді векторів метричних ознак

$$x_i \in \mathbb{R}^d, \tag{1.17}$$

та відповідних бінарних або мульти-лейблових міток

$$y_i^{(s)} \in \{0; 1\}, \tag{1.18}$$

що задають наявність певного антипатерну. Для кожного типу антипатерну формується датасет

$$D^{(s)} = \left\{\left(x_i, y_i^{(s)}\right)\right\}_{i=1}^{N}, \tag{1.19}$$

або, у мульти-лейбловій постановці, вектор міток

$$y_i \in \{0; 1\}^{|S|}, \tag{1.20}$$

На цих даних навчається відображення

$$f^{(s)}: \mathbb{R}^d \to \{0; 1\}, \tag{1.21}$$

яке мінімізує відповідну функцію втрат.

У ролі ознак зазвичай використовуються CK-метрики, розмірні та складнісні показники, інколи доповнені простими процесними метриками з VCS. На їхній основі будуються моделі Decision Tree, Random Forest, Gradient Boosting, SVM або логістичної регресії. Типовий конвеєр таких підходів узагальнено на рисунку 1.6. Основна перевага полягає в автоматичному навчанні нелінійних комбінацій метрик замість ручного підбору порогів, характерного для детекторів на базі правил.

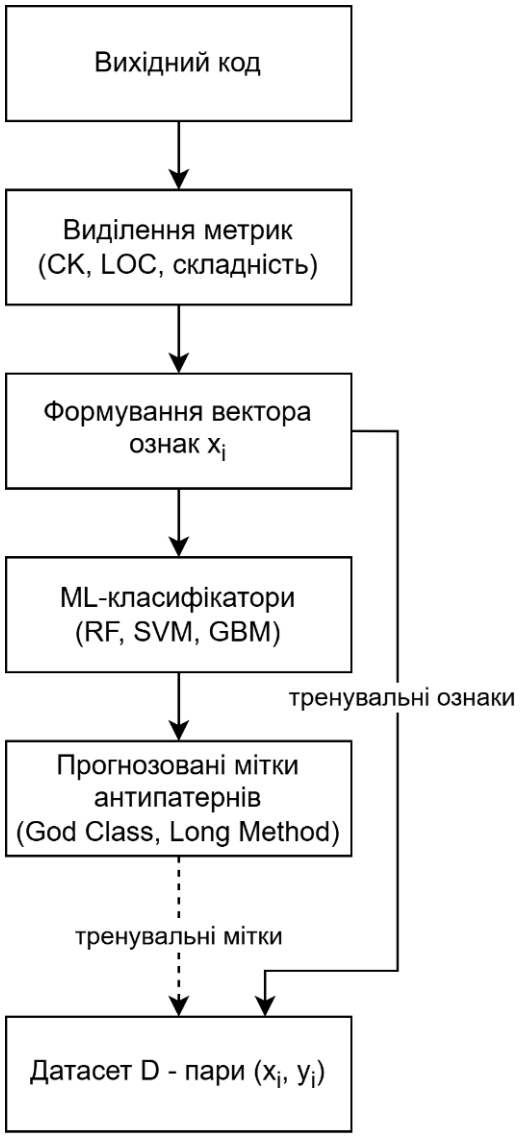

Рисунок 1.6 – Узагальнений конвеєр класичних ML-підходів до виявлення антипатернів

Однак емпіричні дослідження показують, що висока якість таких моделей зазвичай досягається у within-project сценаріях, тоді як у cross-project

налаштуваннях вона суттєво погіршується. Додатковою проблемою є якість датасетів із «запахами» коду: більшість із них обмежується кількома «популярними» антипатернами, переважно Java-кодом, і використовує мітки, отримані від існуючих інструментів. Як наслідок, істинні мітки спостерігаються з шумом

$$P\left(y_i^{(s)} \neq y_i^{*(s)}\right) = \varepsilon_s > 0, \tag{1.22}$$

що обмежує узагальнювальну здатність моделей. Узагальнення доступних датасетів наведено в таблиці 1.9.

Таблиця 1.9 – Приклади датасетів для класичних ML-моделей виявлення антипатернів

Датасет / джерело	Мова(и)	Підтримувані антипатерни	Тип міток	Джерело міток	Особливості використання
Набори в роботі [62]	Java, MVC-системи	God Class, Long Method, інші ОО та MVC-«запахи»	Single-label	Евристичні детектори + метрики	Аналіз впливу балансування даних [62]
Landfill	Java	Кілька класичних «запахів» дизайну	Переважно single	Комбінація інструментів та правил	Відкритий еталон для порівняння детекторів
SmellyCode++	Java	God Class, Data Class, Feature Envy, Long Method	Multi-label	Анотації на основі попередніх досліджень та очищення даних	Поєднання тексту коду та метрик; підтримка LLM + ML [63]
Ручно анотовані датасети з роботи [64]	Переважно Java	Обмежений набір запахів (3–5)	Single або soft labels	Експертні та/або краудсорсингові оцінки	Оцінка суб’єктивності та згодженості міток

1.4.2 Графові та репрезентаційні моделі коду

Подальший розвиток ML/DL-підходів пов’язаний із використанням структурних подань коду у вигляді графів

$$G = (V, E, X), \tag{1.23}$$

де $V -$ множина вузлів (оператори, вирази, базові блоки, інструкції);

E – множина ребер (синтаксичні, контрольні, дата-залежності);

$X \in \mathbb{R}^{|V| \times d_0}$ – матриця початкових ознак вузлів (типи, токени, прості метрики) [65, 66].

На таких поданнях застосовуються графові нейронні мережі, які виконують ітеративне поширення інформації між вузлами

$$h_v^{(k+1)} = \sigma\left(W_1 h_v^{(k)} + \sum_{u \in N(v)} W_2 h_u^{(k)} \right), \tag{1.24}$$

де $N(v)$– множина сусідів вузла v;

W_1, W_2 – матриці ваг;

$\sigma(\cdot)$ – нелінійна функція активації.

Далі відбувається агрегація у графовий ембеддинг

$$z_G = READOUT\left(\left\{ h_v^{(K)} | v \in V \right\} \right), \tag{1.25}$$

де $READOUT$ – інваріантний до перестановки оператор (сума, середнє, attention-пулінг тощо).

Такі моделі дозволяють явно враховувати структурні патерни, релевантні для антипатернів типу «Long Method», «God Class» або архітектурних залежностей. Узагальнений конвеєр графового аналізу показано на рисунку 1.7.

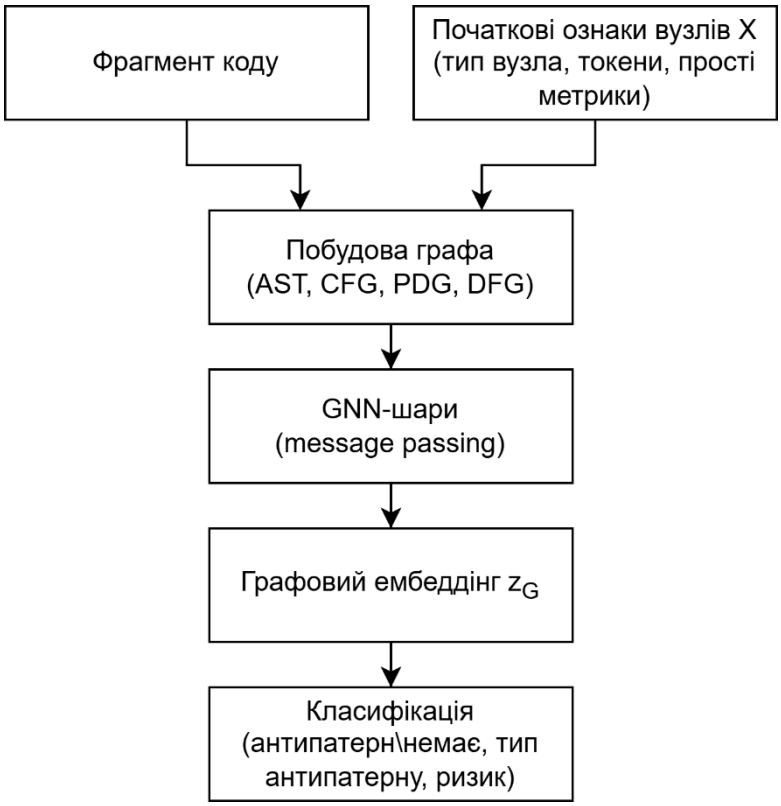

Рисунок 1.7 – Узагальнена схема графової моделі коду

Уніфікованим розвитком цього напряму є Code Property Graph, який об’єднує AST, CFG та PDG у єдине подання. CPG дозволяє описувати антипатерни як підграфи або повторювані структурні конфігурації, проте його практичне застосування у детекції «запахів» обмежується високою вартістю побудови, мовною специфікою та відсутністю стандартних еталонних даних.

Семантичні ембеддинги та попередньо натреновані моделі. Паралельно розвиваються репрезентаційні підходи, що будують семантичні ембеддинги коду

$$\varphi: \text{код} \to z \in \mathbb{R}^m, \tag{1.26}$$

Моделі типу CodeBERT і GraphCodeBERT навчаються на великих мультимовних корпусах і дозволяють враховувати семантику імен, коментарів і контекст використання API. Такі ембеддинги добре підходять для інтеграції у гібридні моделі, однак залишаються обмеженими з точки зору явного аналізу потоку керування та еволюції коду.

Узагальнена характеристика основних типів графових та репрезентаційних моделей наведена в таблиці 1.10, яка демонструє, що жоден із підходів не охоплює повною мірою структурний, семантичний та еволюційний виміри антипатернів.

Таблиця 1.10 – Основні типи графових та репрезентаційних моделей коду і їхні властивості

Тип моделі	Базове подання	Типові задачі	Переваги	Обмеження
GNN над AST/CFG/PDG	Локальні/глобальні графи коду	Дублікати, уразливості, дефекти	Неявне моделювання структурних патернів, потоку керування і даних	Обмежене охоплення архітектурних та еволюційних аспектів; висока вартість побудови
Code Property Graph + запити/GNN	Об’єднаний граф (AST+CFG+PDG)	Уразливості, патерни доступу до ресурсів	Уніфіковане подання коду, формальні graph-запити	Відсутність еталонів «запахів» коду; складність мультимовної підтримки
AST-патерни та code2vec-подібні	Шляхи/послідовності в AST	Іменування, класифікація фрагментів	Компресія структури у вектор; transfer learning	Обмежена чутливість до архітектури, відсутність історії змін
PLM (CodeBERT, GraphCodeBERT тощо)	Токени + (частково) графи/потік даних	Пошук, резюмування, вразливості, якість коду	Потужні універсальні ембеддинги, мульти-мовність, урахування тексту	Високі вимоги до ресурсів; нестача датасетів із «запахами» коду; непрозорість рішень

1.4.3 Гібридні та uncertainty-aware підходи

Гібридні ML/DL-моделі поєднують структурні, семантичні та історичні ознаки:

$$z_i = \left[x_i^{(str)} || x_i^{(txt)} || x_i^{(hist)}\right] \in \mathbb{R}^{d_s+d_t+d_h}, \tag{1.27}$$

після чого над інтегрованим поданням навчається класифікатор або регресійна модель. У більш розвинених архітектурах використовуються attention- або gating-механізми для адаптивного злиття модальностей. Узагальнена архітектура такого підходу наведена на рисунку 1.8.

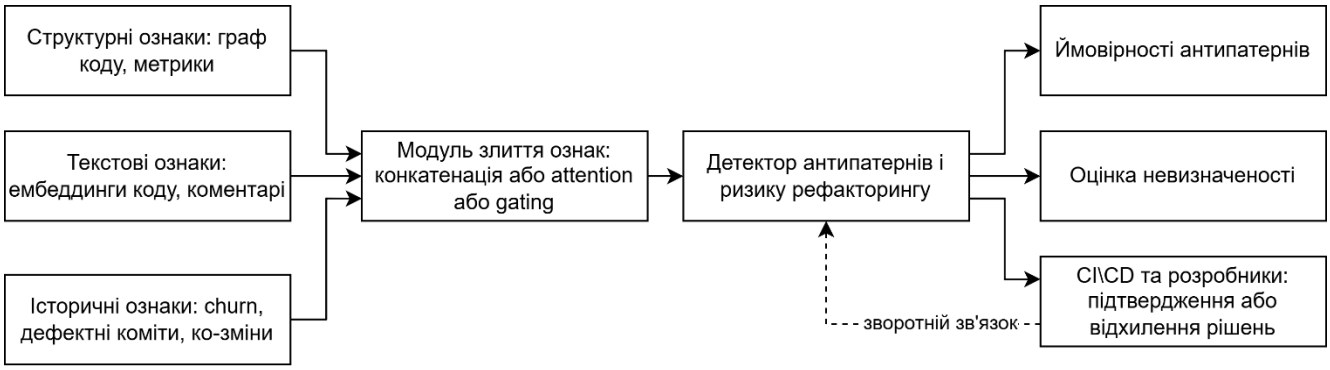

Рисунок 1.8 – Узагальнена архітектура гібридної моделі виявлення антипатернів

Окремим напрямом є uncertainty-aware та open-set-детекція, де модель, окрім прогнозу класу, оцінює власну впевненість і може утримуватися від рішення:

$$S_i = \begin{cases} \arg\max\limits_{s \in S} p_i(s), & \text{якщо } \max\limits_{s \in S} p_i(s) \geq \tau \\ \text{"невідомий/потребує ручної перевірки"}, & \text{інакше} \end{cases}, \tag{1.28}$$

де $\tau \in (0,1)$ – поріг впевненості.

Такі механізми дозволяють зменшити кількість хибних спрацювань у нетипових або домен-специфічних випадках.

1.4.4 Підтримка рефакторингу та роботи з дублікатами коду моделями на базі машинного навчання та глибинних моделей

Сучасні ML/DL-системи підтримки рефакторингу розглядають задачу рекомендації окремих рефакторингів або їхнього ранжування для конкретного фрагмента коду. Для кожного кандидата формується вектор ознак, а модель оцінює очікувану корисність і ризик змін. Узагальнений конвеєр таких підходів показано на рисунку 1.9.

Більшість робіт фокусується на локальних сценаріях («Extract Method», «Move Method», окремі групи дублікатів) і не моделює узгоджені послідовності кроків у масштабі всієї системи. Аналогічні обмеження характерні і для рекомендацій щодо дубльованого коду, де історичні ознаки з VCS відіграють ключову роль, але ризики та взаємодія змін часто залишаються неявними.

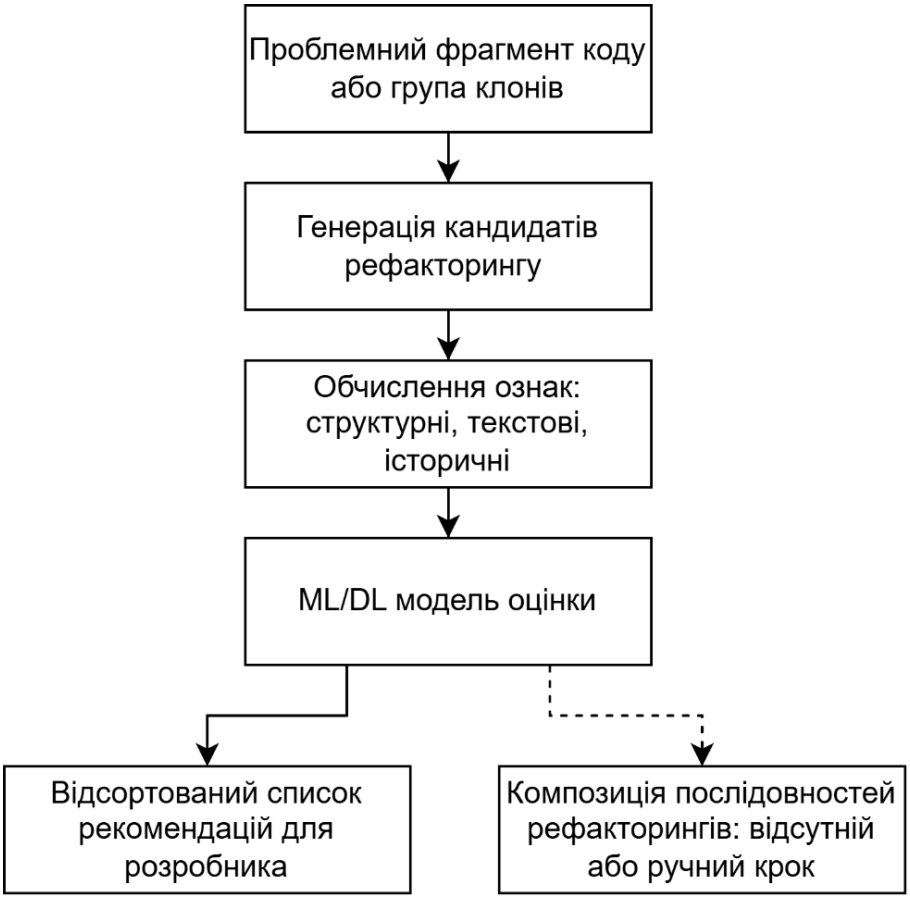

Рисунок 1.9 – Узагальнений конвеєр ML/DL-підтримки рефакторингу проблемних фрагментів та клонів коду

Узагальнення класів підходів наведено в таблиці 1.11.

Таблиця 1.11 – Узагальнення класів ML/DL-підходів до підтримки рефакторингу та роботи з дубльованим кодом

Тип підходу	Статті	Вхідні дані	Ціль / вихід	Контекст (проєкт, історія)	Послідовності кроків	Оцінка ризику
ML-рейтинг окремих рефакторингів (Extract/Move)	[68-69]	Локальні метрики, структура методу/класу, іноді прості історичні ознаки	Рейтинг кандидатів рефакторингу для конкретного фрагмента	Переважно локальний (один проєкт, один модуль)	Не моделюється; кожен крок розглядається незалежно	Евристична (неявна через метрики) або відсутня
Передбачення факту рефакторингу / потреби в ньому	[70]	Історія комітів, метрики, наявність запахів	Бінарний прогноз: «буде рефакторинг» / «не буде»	Враховується історія змін елемента коду	Не моделюється; прогнозуються поодинокі події	Опосередковано (через ймовірність події), без явного розподілу ризику
Рекомендація рефакторингу клонів	[71, 72]	Групи клонів, їхній розмір та розташування, еволюція в VCS, co-change	Класифікація: «рефакторити / не рефакторити», пріоритизація груп клонів	Враховується історія проєкту, зв’язки між клонованими фрагментами	Послідовність кроків для всієї групи зазвичай не оптимізується; пропонується одна стратегія	Неявна; ризик пов’язується з розміром/складністю клонів
DL-оцінка наслідків рефакторингу	[73]	Деталізовані характеристики коду, конфігурації виконання	Прогноз зміни продуктивності після Move Method	Локальний контекст застосування рефакторингу, без повного проєктного огляду	Не моделюється; розглядається один рефакторинг	Явна, але фокусується тільки на продуктивності

У підсумку, сучасні ML/DL-підходи демонструють значний прогрес у порівнянні з традиційними методами, проте залишаються фрагментарними: вони переважно працюють у закритій постановці, обмежено інтегрують багаторівневий контекст і рідко моделюють невизначеність.

Це обґрунтовує необхідність розробки інтегрованих, багаторівневих і uncertainty-aware моделей, здатних поєднувати структурні, семантичні та еволюційні аспекти коду, що і становить предмет подальших розділів дисертації.

1.5 Ключові виклики сучасних систем виявлення антипатернів

Попри значний прогрес у розвитку підходів на базі правил та метрик, в також ML/DL-підходів, сучасні системи виявлення антипатернів залишаються вразливими до низки фундаментальних викликів.

Вони зумовлені як природою вхідних даних і обмеженнями моделей, так і організаційним контекстом їх застосування та вимогами інженерних процесів. В межах дисертаційного дослідження виділено шість ключових груп проблем: шумність даних і нечіткість ознак, взаємозалежність ознак та нелінійність рішень, проєктна специфічність, мультимовність і слабка переносимість моделей, відсутність механізмів роботи з новими або атиповими антипатернами, а також потреба у пояснюваності рішень та інтеграції в CI/CD.

Надалі слід вважати, що система виявлення антипатернів працює з множиною об’єктів (класів, методів, пакетів)

$$D = \{(x_i, y_i)\}_{i=1}^{N}, \tag{1.29}$$

де $x_i \in \mathbb{R}^d -$ вектор ознак, сформований на основі метрик, структурних, текстових та історичних характеристик;

$y_i \in \{0,1, \dots, K\}$ – мітка відсутності антипатерну (0) або одного з K типів антипатернів.

У реальних умовах як ознаки, так і мітки є результатом наближених вимірювань та експертних суджень, що породжує низку системних проблем.

1.5.1 Шумність даних і нечіткість ознак

Одним із базових джерел невизначеності є шумність у просторі ознак і в мітках. Більшість метрик коду (LOC, WMC, CBO, LAA тощо) є лише непрямими індикаторами складності чи якості і не мають однозначної інтерпретації. Різні стилі програмування, фреймворки та наявність згенерованого коду призводять до того, що однакові значення метрик можуть відповідати як коректним, так і проблемним фрагментам.

Додатковим чинником є шумність міток, що формуються на основі інструментів статичного аналізу або експертної розмітки. Якщо позначити латентну «істинну» мітку як y_i^*, а спостережувану – як $\tilde{y}_i$, то ймовірність помилки розмітки можна подати у вигляді

$$\varepsilon = \mathbb{P}(\tilde{y}_i \neq y_i^*), \tag{1.30}$$

де ε залежить від типу антипатерну, складності прикладу та застосованого інструмента. Для деяких запахів узгодженість є відносно високою, тоді як для більш контекстних антипатернів вона суттєво знижується.

Це призводить до того, що навчена модель часто відображає особливості конкретного датасету, а не узагальнені закономірності, знижуючи її стабільність і переносимість.

1.5.2 Взаємозалежність ознак і нелінійність рішень

Ознаки внутрішньої якості ПЗ є суттєво корельованими, що створює ефект мультиколінеарності. У загальному випадку рішення ML-моделі можна подати як

$$\hat{y}_i = f(x_i; \theta), \tag{1.31}$$

де $f(\cdot)$ – нелінійна функція (наприклад, ансамбль дерев, нейронна мережа або GNN);

θ – параметри, навчений на історичних прикладах [19, 62, 74].

Важливу роль відіграють не лише окремі значення ознак, а й їхні взаємодії вищих порядків:

$$f(x_i;\theta) \approx g(x_i, x_i^{(j)}x_i^{(k)}, x_i^{(j)}x_i^{(k)}x_i^{(l)}, \dots), \tag{1.32}$$

де $x_i^{(j)}$ – j-та компонента вектору ознак.

Це означає, що проблемність фрагмента коду визначається складною комбінацією факторів, а не окремою метрикою. Хоча DL-моделі здатні автоматично виявляти такі взаємодії, вони значно ускладнюють інтерпретацію результатів і створюють проблему «чорної скриньки».

1.5.3 Проєктна специфічність і неможливість універсальних порогів

Класичні підходи часто використовують фіксовані порогові значення метрик, однак емпіричні дослідження показують, що розподіли метрик істотно відрізняються між проєктами. Формально це можна описати як зміну розподілу

$$\mathcal{P}_{\text{train}}(X,Y) \neq \mathcal{P}_{\text{proj}}(X,Y), \tag{1.33}$$

де $\mathcal{P}_{\text{train}}$ – розподіл у навчальному наборі;

$\mathcal{P}_{\text{proj}}$ – реальний розподіл у цільовому проєкті.

Внаслідок цього універсальні пороги або глобально навчені моделі призводять до систематичних похибок. Цю проблему ілюструє рисунок 1.10, де показано, що одна межа для різних проєктів неминуче буде або надто м’якою, або надто жорсткою. Це обґрунтовує потребу в адаптивних, проєкт-орієнтованих підходах та методах доменної адаптації.

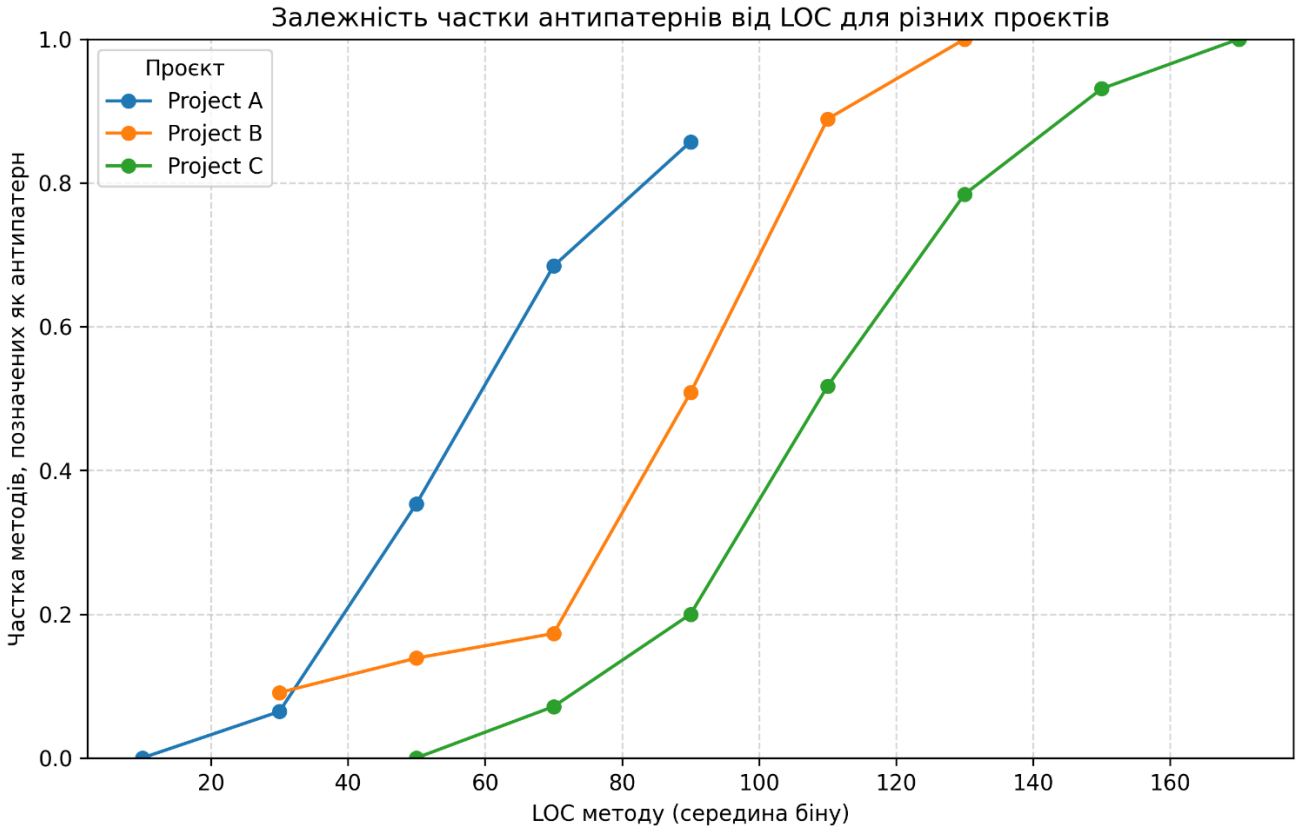

Рисунок 1.10 – Приклад проєктної специфічності порогових значень метрик для виявлення антипатернів

Узагальнення ключових викликів і їхніх проявів наведено в таблиці 1.12.

Таблиця 1.12 – Основні виклики виявлення антипатернів та їхні прояви

Виклик	Типові прояви в практиці	Наслідки для систем виявлення антипатернів
Шумність даних, нечіткість ознак	Суперечливі мітки, різні інструменти дають різні результати	Зниження точності, нестабільність моделей
Взаємозалежність ознак, нелінійність	Корельовані метрики, складні взаємодії ознак	Перенавчання, складність інтерпретації результатів
Проєктна специфічність	Різні «норми» метрик для різних систем	Неможливість універсальних порогів, потреба в калібруванні

Продовження таблиці 1.12

Виклик	Типові прояви в практиці	Наслідки для систем виявлення антипатернів
Мультимовність, слабка переносимість	Моделі навчені для однієї мови/стеку	Падіння якості при застосуванні до інших мов/платформ
Нові / атипові антипатерни	Зустріч невідомих патернів, яких немає в навчанні	Небезпека «насильного» віднесення до відомих класів
Пояснюваність, інтеграція в CI/CD	Відсутність прозорих пояснень, шум у пайплайні	Недовіра розробників, ігнорування рекомендацій

1.5.4 Мультимовність і слабка переносимість моделей

Більшість наявних систем орієнтована на одну мову програмування, тоді як промислові репозиторії часто є мультимовними. Перенесення правил, порогів і моделей між мовами ускладнюється різними ідіомами, архітектурними стилями, набором доступних метрик і мовно-специфічними антипатернами. Навіть репрезентаційні моделі коду лише частково знижують цю проблему, оскільки потребують достатнього обсягу мультимовних навчальних даних, які для задач виявлення «запахів» коду» наразі обмежені.

1.5.5 Відсутність механізмів роботи з новими та атиповими антипатернами

Більшість систем працює у закритій постановці, припускаючи, що всі типи антипатернів відомі заздалегідь. Формально класичний класифікатор

$$f: \mathcal{X} \to \mathcal{Y}_{\text{known}} \tag{1.34}$$

не здатний коректно обробляти приклади з множини невідомих антипатернів. Для роботи в реальних умовах необхідне розширення до

$$f' : \mathcal{X} \to \mathcal{Y}_{\text{known}} \cup \{\perp\}, \tag{1.35}$$

де $\perp$ – «невідомий або атиповий антипатерн, потребує уваги експерта».

Це потребує механізмів оцінки невизначеності, виявлення out-of-distribution-прикладів і використання активного навчання, які поки що лише частково застосовуються у задачах детекції «запахів» коду.

1.5.6 Пояснюваність рішень та інтеграція в CI/CD

Ефективність систем виявлення антипатернів значною мірою визначається тим, наскільки їхні результати приймаються розробниками.

Для цього необхідні зрозумілі пояснення причин класифікації та можливих наслідків. Формально для кожного прогнозу бажано мати пояснювальне відображення

$$e_i = g(x_i, \hat{y}_i, \theta), \tag{1.36}$$

де e_i – інтерпретація, придатна для сприйняття людиною (список найважливіших метрик, приклади подібних фрагментів, пропозиції рефакторингу).

Відсутність пояснюваності та надлишок попереджень у CI/CD-пайплайнах призводять до зниження довіри та ігнорування рекомендацій. Тому сучасні системи мають поєднувати контроль рівня шуму, ранжування знахідок за ризиком і замкнений цикл зворотного зв’язку для донавчання моделей.

Таким чином, ключові виклики сучасних систем виявлення антипатернів мають комплексний характер і охоплюють як алгоритмічні, так і процесні аспекти. Подолання цих обмежень вимагає інтегрованих, адаптивних і uncertainty-aware підходів, що безпосередньо мотивує запропоновані в подальших розділах дисертації моделі та методи.

1.6 Узагальнення обмежень існуючих підходів та формування вимог до сучасних систем виявлення антипатернів

Проведений у попередніх підрозділах аналіз засвідчив, що попри значний прогрес у дослідженнях антипатернів і широке впровадження інструментів статичного аналізу та ML-моделей у промислові процеси, наявні підходи залишаються принципово обмеженими з позицій сучасної розробки програмного забезпечення. Більшість із них є одноканальними за типом ознак, орієнтованими переважно на локальний рівень аналізу, побудованими в закритій постановці та слабо інтегрованими в інженерні CI/CD-пайплайни, що ускладнює їх використання для керування довготривалою еволюцією систем.

1.6.1 Недостатність одноканальних моделей

Типові системи виявлення антипатернів спираються на один домінуючий канал ознак. Підходи на базі метрик використовують лише структурні характеристики коду, текстові моделі аналізують послідовності токенів або семантичні ембеддинги, а історично орієнтовані методи зосереджуються на еволюційних показниках репозиторію. Водночас антипатерни є комплексними феноменами, що формуються на перетині структури, семантики реалізованої функціональності та історії змін.

Один і той самий структурно «підозрілий» фрагмент може бути прийнятним у стабільному компоненті та критичним у модулі з інтенсивним churn і дефектною історією, тоді як одноканальні моделі не здатні коректно розрізняти такі ситуації. Це призводить або до надмірної кількості хибних спрацювань, або до пропуску дійсно ризикованих фрагментів коду.

Для подолання цих обмежень у дисертації запропоновано багаторівневу модель виявлення антипатернів на основі гібридного графового подання коду, яка поєднує структурні (AST/CFG/PDG), семантичні та історичні ознаки в єдиному

інформаційному просторі. Такий підхід дозволяє описувати антипатерни як багатовимірні патерни поведінки, а не як ізольовані відхилення окремих метрик.

1.6.2 Потреба у багаторівневому аналізі

Більшість існуючих детекторів приймає рішення на рівні окремих методів або класів, тоді як значна частина антипатернів має системний характер і проявляється лише на компонентному або проєктному рівнях. Архітектурні «запахи», накопичення технічного боргу та ефекти взаємодії між модулями стають помітними лише при узгодженому аналізі декількох рівнів абстракції.

Сучасні DevOps-практики вимагають оцінювання якості не лише локально, а й у часовій динаміці – як здатності системи зберігати прийнятний рівень підтримуваності та стабільності протягом багатьох релізних циклів. Проте наявні інструменти або не підтримують такої вертикальної інтеграції, або обмежуються простими агрегатами, що ігнорують взаємозалежності між компонентами.

У дисертації ці обмеження долаються завдяки багаторівневому проєктуванню гібридного подання коду та методу композиції «мінімально інвазивних» послідовностей рефакторингів, який явно враховує причинно-наслідкові зв’язки між змінами на різних рівнях системи.

1.6.3 Необхідність інтеграції різних джерел ознак

Як показано в підрозділах 1.4–1.5, сучасні дослідження розвивають метрик-орієнтовані, текстові, історичні та графові підходи переважно паралельно, без їх формального об’єднання. У результаті різні моделі можуть давати суперечливі рекомендації, оскільки ґрунтуються на несумісних інформаційних базисах.

Сучасні вимоги розробки обумовлюють перехід до гібридних подань, у яких структурні, семантичні та еволюційні ознаки інтегруються в єдиний вектор або графову репрезентацію. Лише на основі такого узгодженого подання можливо

будувати моделі виявлення антипатернів і рекомендації рефакторингів, які не суперечать одна одній та працюють у спільному контексті.

У дисертації ця вимога реалізована через гібридну графову модель, що поєднує Code Property Graph із семантичними ембеддингами та історичними ознаками з VCS, і слугує спільною основою для детекції антипатернів та оптимізації планів рефакторингу.

1.6.4 Вимога до open-set-детекції та моделювання невизначеності

Класичні детектори та більшість ML-моделей працюють у закритій постановці, припускаючи, що всі типи антипатернів відомі заздалегідь. На практиці ж кодові бази постійно еволюціонують, породжуючи нові або проєкт-специфічні патерни деградації якості. Крім того, більшість моделей не надає каліброваної оцінки невизначеності прогнозів, що підвищує ризик агресивних або, навпаки, надто обережних рішень.

У дисертації ця проблема розв’язується шляхом введення open-set/low-confidence режиму в моделі виявлення антипатернів та явного моделювання невизначеності при рекомендації рефакторингів. Це дозволяє відмовлятися від автоматичних рішень у зонах високого ризику та залучати експертів лише там, де це дійсно необхідно.

1.6.5 Обмеження пояснюваності та інтеграції в інженерні пайплайни

Навіть точні з погляду офлайнових метрик моделі залишаються малокорисними, якщо їх результати погано інтегровані в повсякденну роботу команд. Плоскі списки попереджень, відсутність пояснень та формалізованої оцінки ефекту рефакторингів призводять до втрати довіри й ігнорування рекомендацій.

Сучасні CI/CD-практики вимагають замкненого контуру керування якістю, у якому рішення моделі пояснюються через ключові ознаки та історичний контекст,

рекомендації узгоджуються з обмеженнями пайплайну, а ефект змін оцінюється у часовій динаміці.

У дисертації ці вимоги реалізовані через поєднання рекомендера рефакторингів, методу композиції послідовностей змін та модуля статистичного контролю процесів, що дозволяє формалізовано оцінювати результативність втручань і знижувати ризик хибних рішень.

1.7 Висновки до першого розділу

У першому розділі дисертації здійснено системний аналіз проблеми внутрішньої якості програмного забезпечення та ролі антипатернів як ключових індикаторів її деградації. Показано, що внутрішня якість є багатовимірною характеристикою, тісно пов’язаною зі структурою коду, семантикою реалізованої функціональності та еволюційною історією програмних компонентів, а антипатерни відображають не поодинокі дефекти, а стійкі дисбаланси в архітектурі й дизайні систем, що накопичуються в процесі розвитку проєкту.

Аналіз традиційних методів виявлення антипатернів на базі правил та метрик показав їх практичну цінність з огляду на інтерпретованість, масштабованість та простоту інтеграції у промислові процеси. Водночас встановлено, що такі підходи є чутливими до вибору порогових значень, демонструють низьку узгодженість між різними інструментами та істотно залежать від мови програмування, архітектурного стилю і специфіки конкретного проєкту, що обмежує їх здатність до узагальнення та автоматизованого прийняття рішень.

Розглянуті структурні, текстові та мовно-модельні підходи до аналізу коду підтвердили потенціал використання графових подань, семантичних ембеддингів і великих мовних моделей для виявлення складніших і більш контекстних антипатернів. Разом із тим встановлено, що більшість сучасних рішень застосовує ці підходи ізольовано або поєднує їх лише на рівні простого агрегування результатів, що не дозволяє повною мірою узгодити структурні, семантичні та історичні аспекти якості коду в єдиній моделі.

У підрозділі, присвяченому ключовим викликам сучасних систем виявлення антипатернів, показано, що шумність даних, корельованість ознак і нелінійність рішень, проєктна специфічність розподілів метрик, мультимовність, відсутність механізмів роботи з новими та атиповими антипатернами, а також обмежена пояснюваність і слабка інтеграція в CI/CD-процеси суттєво знижують практичну ефективність існуючих підходів. Наголошено, що ці проблеми мають комплексний характер і не можуть бути усунені шляхом локального вдосконалення окремих детекторів або підбору більш «вдалих» порогів.

Аналітичне узагальнення результатів огляду дозволило сформулювати ключові вимоги до сучасних систем виявлення антипатернів, а саме необхідність переходу від одноканальних до інтегрованих гібридних подань коду, підтримки багаторівневого аналізу на локальному, компонентному та проєктному рівнях, явного моделювання невизначеності та open-set-детекції, а також тісної інтеграції з інженерними CI/CD-пайплайнами із забезпеченням пояснюваності рішень і формалізованого оцінювання ефекту рефакторингів у часовій динаміці.

2 БАГАТОРІВНЕВА МОДЕЛЬ ВИЯВЛЕННЯ АНТИПАТЕРНІВ У ПРОГРАМНИХ КОМПОНЕНТАХ НА ОСНОВІ ГІБРИДНОГО ГРАФОВОГО ПОДАННЯ КОДУ

2.1 Постановка задачі та обґрунтування вимог до моделі виявлення антипатернів

У першому розділі дисертації було показано, що антипатерни («запахи» коду, «запахи» дизайну/архітектури, дублікати коду та інші дефекти структури) є стабільним джерелом технічного боргу та погіршення підтримуваності програмних систем, що узгоджується з моделлю внутрішньої якості ISO/IEC 25010 [75] та сучасними підходами до технічного боргу [76, 77]. У цьому підрозділі базову постановку задачі автоматичного виявлення антипатернів уточнено з урахуванням багатовимірної природи явища, вимог до багаторівневого аналізу («метод – компонент – проєкт») та потреби у відкритому (open-set) режимі з урахуванням невизначеності. Надалі під терміном *програмний компонент* буде розумітися сутність, що може відповідати методу, класу/модулю або логічно виділеному підкомпоненту, для якого можна зібрати структурні, семантичні, метричні та еволюційні ознаки.

2.1.1 Багатовимірна природа антипатернів у програмних системах

Емпіричні дослідження показують, що антипатерн не є властивістю лише структурного рівня коду: його наявність та небезпечність визначаються поєднанням декількох типів сигналів – топології коду, семантики обчислень, метричних характеристик та еволюції в часі [77, 78]. Наприклад, довгий метод може бути прийнятним, якщо його логіка проста і стабільна, тоді як відносно короткий метод із часто змінюваною, умовно насиченою логікою утворює більш критичний антипатерн. Формально вважатимемо, що кожному компоненту $c \in C$ ставиться у відповідність вектор ознак

$$\mathbf{x}_c = (\mathbf{x}_c^{(str)}, \mathbf{x}_c^{(sem)}, \mathbf{x}_c^{(met)}, \mathbf{x}_c^{(evo)}) \in \mathbb{R}^d, \quad (2.1)$$

де $\mathbf{x}_c^{(str)}$ – описує структурні характеристики (AST/CFG/граф викликів);

$\mathbf{x}_c^{(sem)}$ – семантичні представлення коду;

$\mathbf{x}_c^{(met)}$ – класичні метрики;

$\mathbf{x}_c^{(evo)}$ – еволюційні показники (churn, частота змін, «вік», ко-зміни тощо).

Нехай $A = \{a_1, \dots, a_K\}$ – множина типів антипатернів («запахи» коду, «запахи» дизайну, дублікати коду та інші дефекти). Оскільки один компонент може одночасно містити кілька антипатернів, вихідна змінна має мультиміткову природу:

$$\mathbf{y}_c = (y_{c,1}, \dots, y_{c,K}) \in \{0,1\}^K, \quad (2.2)$$

де $y_{c,k} = 1$ – наявність антипатерну типу a_k у компоненті c.

Таким чином, задача виявлення антипатернів на рівні компонентів у найзагальнішому вигляді формулюється як побудова відображення

$$f: \mathbb{R}^d \to [0,1]^K, \quad (2.3)$$

яке для кожного компонента c за його вектором ознак $\mathbf{x}_c$ повертає оцінку ймовірностей (або ступеня впевненості) щодо наявності кожного з антипатернів a_k. Подальші вимоги до моделі уточнюють структуру f, типи ознак та поведінку моделі в умовах невизначеності.

2.1.2 Вплив антипатернів на внутрішню якість та підтримуваність

Модель якості ISO/IEC 25010 виділяє підтримуваність як одну з ключових характеристик внутрішньої якості ПЗ [75]. Внутрішня якість включає модульність, зрозумілість, змінюваність, тестованість та повторне використання. Антипатерни безпосередньо впливають на ці підхарактеристики, погіршуючи прозорість

структури, збільшуючи зв’язаність та зменшуючи когезію компонентів. У сучасній теорії технічного боргу антипатерни розглядаються як конкретні «носії» боргу, що збільшують поточну й майбутню вартість змін [76, 77]. Для компоненту cможна ввести узагальнену оцінку технічного боргу

$$D(c) = \sum_{k=1}^{K} w_k \cdot s_{c,k}, \tag{2.4}$$

де $s_{c,k}$ – показник «інтенсивності» антипатерну a_k (наприклад, кількість локалізацій або нормалізована міра тяжкості);

w_k – ваговий коефіцієнт, що відображає усереднену трудомісткість його усунення та ризики для якості.

Емпіричні роботи демонструють статистично значущий зв’язок між наявністю «запахів» коду, збільшенням витрат на супровід і дефектністю [77, 78].

Отже, модель виявлення антипатернів, яка спирається лише на бінарне рішення «є/немає», є недостатньою: результат її роботи повинен бути інтегрованим у більш загальну схему оцінювання внутрішньої якості та технічного боргу на різних рівнях. Це обумовлює вимогу до моделі бути сумісною з метричними підходами до оцінки підтримуваності та технічного боргу.

2.1.3 Обмеження традиційних правил, метрик та існуючих моделей машинного навчання

Аналіз існуючих інструментів і досліджень, узагальнених у низці систематичних оглядів [79-81], показує, що переважна більшість підходів до виявлення антипатернів належить до однієї з таких груп:

– підходи на базі правил та метрик – класичні стратегії на основі порогових значень метрик (наприклад, CK, LOC, глибина вкладеності) та евристичних правил добре інтерпретуються, але чутливі до вибору порогів, погано узагальнюються між проєктами та часто дають високий рівень хибних спрацювань у системах [77, 80];

– AST-only та структурні ML-моделі – моделі, що працюють лише з деревами синтаксичного розбору, ігнорують семантичний контекст (наприклад, значення ідентифікаторів, контракт методів, доменну термінологію) та історичні ознаки. Це обмежує їх здатність відрізняти «зовні схожі», але семантично різні структури;

– текстові та PLM-підходи без явної структури – моделі, що використовують тільки послідовності токенів або вбудовування на кшталт code2vec або CodeBERT без явного графового подання, недостатньо враховують контрольну і дата-флоу структуру коду [80];

– історичні та процесні моделі – підходи, що базуються на даних з репозиторіїв (churn, ко-зміни, дефектність) добре відображають еволюційний аспект, але не дають повної картини локальної структури та семантики компонентів [82].

У таблиці 2.1 узагальнено основні типи підходів, використані джерела ознак (структура, семантика, метрики, еволюція) та їх характерні недоліки.

Таблиця 2.1 – Порівняння класів підходів до виявлення антипатернів

Клас підходів	Використовувані сигнали	Основні обмеження
На базі правил та метрик (евристики, порогові значення метрик)	Структура: опосередковано через прості метрики (LOC, складність, CK); семантика: не використовується; метрики: основний сигнал; еволюція: не використовується	Чутливість до вручну заданих порогів; слабка переносимість між проєктами й мовами; погане врахування контексту; високий рівень хибних спрацювань та пропусків складних антипатернів.
AST-only та інші суто структурні ML-моделі	Структура: AST/CFG, іноді граф викликів; семантика: майже не враховується (токени без глибоких ембеддингів); метрики: використовуються обмежено або взагалі ні; еволюція: не враховується	Ігнорування семантичного змісту ідентифікаторів і доменної термінології; складність узагальнення на інші стилі коду; відсутність інформації про історію змін і технічний борг.

Продовження таблиці 2.1

Клас підходів	Використовувані сигнали	Основні обмеження
Текстові / PLM-підходи без явної структурної компоненти (послідовності токенів, CodeBERT-подібні моделі «як є»)	Структура: враховується неявно через позицію токенів; семантика: основний сигнал (контекстні ембеддинги токенів, коментарів); метрики: майже не інтегруються; еволюція: не використовується	Обмежена здатність моделювати контрольні та дата-флоу залежності; складність інтерпретації; відсутність явного зв'язку з класичними метриками й еволюційними показниками; ризик «галюцинацій» на нетипових структурах.
Історичні / процесні моделі (MSR, дефект-предикція, JIT-аналітика)	Структура: враховується опосередковано або зовсім ні; семантика: здебільшого ігнорується; метрики: використовуються процесні (кількість змін, автори, дефекти); еволюція: основний сигнал (churn, ко-зміни, стабільність)	Не дають локальної діагностики конкретних антипатернів у коді; придатні радше для виявлення «гарячих точок» і ризикових файлів; потребують довгої історії змін; чутливі до шуму в репозиторіях.
Існуючі гібридні та мультимодальні підходи (структура + текст + метрики / процесні ознаки)	Структура: AST/графи; семантика: ембеддинги токенів/ідентифікаторів; метрики: частково інтегруються; еволюція: іноді враховується (churn, ко-зміни)	Немає систематичної підтримки багаторівневої архітектури; практично відсутній open-set / low-confidence режим; часто складні для інкрементального оновлення й інтеграції в CI/CD; обмежена переносимість між мовами та організаціями.

Як видно з таблиці, жоден з класів моделей не забезпечує одночасно багатовимірного подання компонентів, багаторівневого аналізу та явної роботи з невизначеністю. Крім того, більшість існуючих ML-рішень припускають закриту постановку задачі (closed set): вважається, що всі можливі типи антипатернів відомі наперед, а всі вхідні приклади належать до цього фіксованого набору класів [81]. Така постановка неадекватна умовам промислової розробки, де постійно

з’являються нові стилі кодування, фреймворки й архітектурні практики, а також раніше не описані антипатерни.

2.1.4 Необхідність інтегрованого подання «структура + семантика + метрики + еволюція» та багаторівневого аналізу

Сучасні тенденції в прогнозуванні дефектів і якості ПЗ демонструють перехід до мультимодальних моделей, що одночасно враховують різнорідні ознаки – структурні, текстові, процесні, операційні [80, 82, 83]. Для задачі виявлення антипатернів це означає потребу в інтегрованому поданні, яке:

– об’єднує структуру коду (AST, графи керування й потоків даних);

– враховує семантику (вбудовування токенів, ідентифікаторів, коментарів із моделей для коду);

– включає метрики якості (CK-подібні метрики, розмір, глибину вкладеності тощо);

– доповнюється еволюційними ознаками (churn, частота змін, ко-зміни, стабільність);

Узагальнено таке подання можна розглядати як відображення

$$\Phi: \mathcal{C} \to \mathbb{R}^d, \qquad \Phi(\mathrm{c}) = \mathbf{x}_c, \tag{2.5}$$

де C – множина всіх компонентів проєкту;

$\mathbf{x}_c$ – згаданий раніше гібридний вектор ознак.

На рисунку 2.1 концептуально показано, як гібридне подання на рівні методів агрегується до рівня компонентів і проєкту, забезпечуючи узгоджений багаторівневий аналіз.

Крім інтеграції сигналів на рівні окремого компонента, важливим є багаторівневий аналіз:

– аналіз на рівні методів – виявлення локальних антипатернів (Long Method, Feature Envy тощо) з урахуванням локальної структури й семантики;

– аналіз на рівні компонентів – агрегація відомостей про методи, внутрішні залежності, інтерфейси та історію зміни класу/модуля;

– аналіз на рівні проєкту – аналіз розподілу антипатернів по модулях, підсистемах, взаємозв'язок з архітектурними рішеннями та процесними метриками (наприклад, частота релізів, стабільність CI/CD-пайплайна [82]).

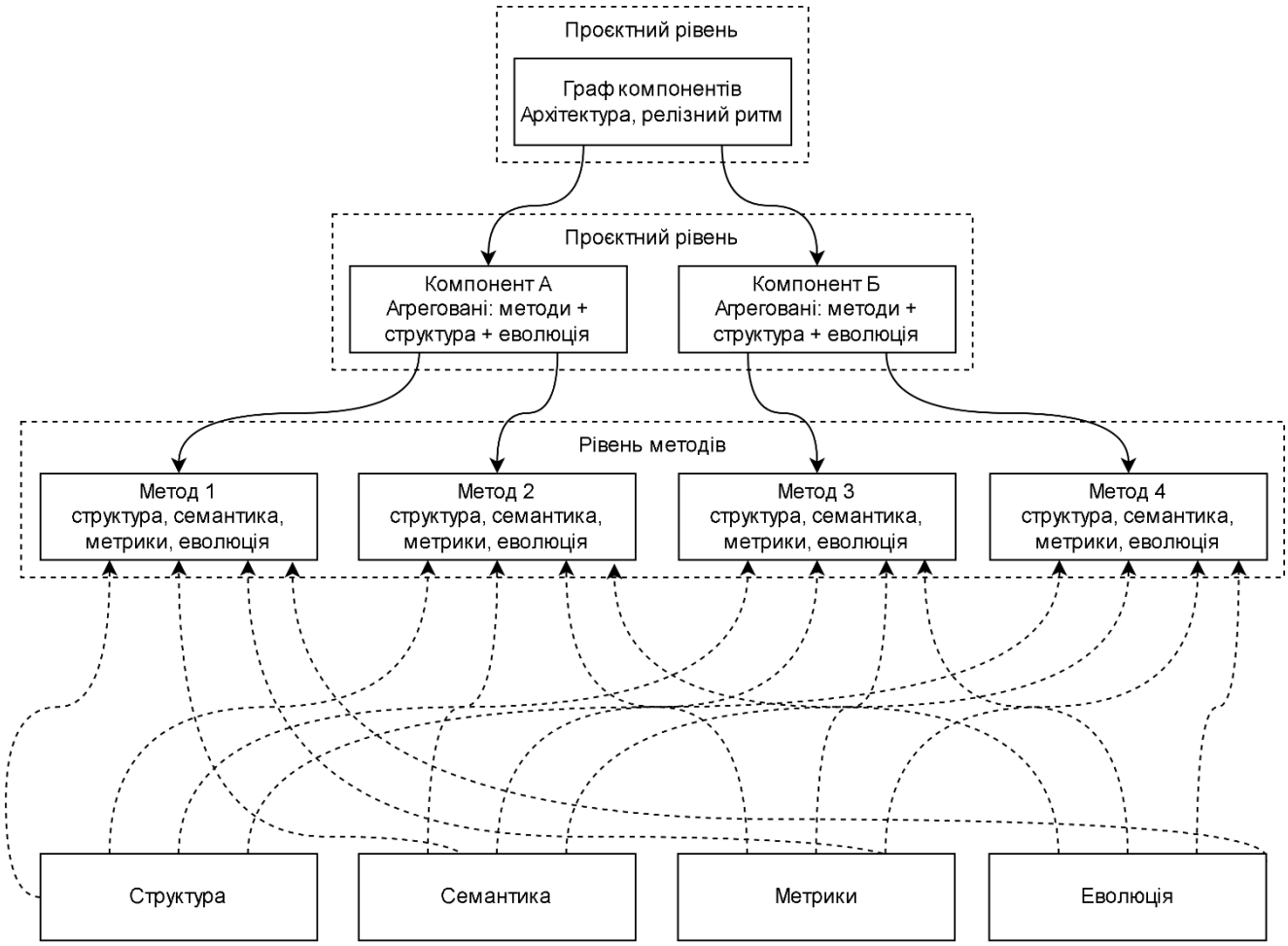

Рисунок 2.1 – Схематичне зображення багаторівневої моделі

2.1.5 Вимоги до запропонованої моделі виявлення антипатернів

З урахуванням викладеного вище, а також вимог до безперервного контролю якості в DevOps-середовищах [82] необхідно сформулювати узагальнену постановку задачі та вимоги до моделі виявлення антипатернів.

1. Багатовимірність ознак. Модель повинна працювати з інтегрованим вектором ознак

$$\mathbf{x}_c = (\mathbf{x}_c^{(str)}, \mathbf{x}_c^{(sem)}, \mathbf{x}_c^{(met)}, \mathbf{x}_c^{(evo)}), \tag{2.6}$$

тобто одночасно враховувати структуру, семантику, метрики та еволюцію. Це означає підтримку гібридного подання (зокрема, у вигляді графів властивостей коду), але конкретні деталі подання розкриваються у наступних підрозділах.

2. Багаторівнева архітектура («метод – компонент – проєкт»). Модель має містити узгоджений набір підмоделей

$$f^{(m)}, f^{(c)}, f^{(p)}, \tag{2.7}$$

що відповідають рівням «метод», «компонент» та «проєкт» відповідно. Для метода m визначається локальний вектор ознак $\mathbf{x}_m$ та оцінка $\mathbf{y}_m$; для компонента c – агреговані представлення $\mathbf{z}_c = \rho(\{\mathbf{y}_m\}_{m \in c})$; для проєкту P – $\mathbf{z}_P = \pi(\{\mathbf{z}_c\}_{c \in P})$, де ρ та π – оператори агрегації. Така багаторівнева схема узгоджується з підходами до багаторівневого оцінювання якості [83].

3. Open-set або low-confidence режим. Модель повинна вміти виявляти випадки, коли наявні ознаки не відповідають жодному з відомих типів антипатернів або виходять за межі тренувального розподілу. У спрощеному вигляді це реалізується через введення «відмови від рішення»:

$$\gamma_c = \max_{k=1,\dots,K} f_k(\mathbf{x}_c), \quad \gamma_c < \tau \;\Rightarrow\; \text{позначити компонент } c \text{ як невідомий/нетиповий і передати експерту}, \tag{2.8}$$

де τ – порогове значення впевненості.

Потреба у такому режимі обґрунтовується результатами досліджень з open-set recognition та невизначеності в процесі аналізу ПЗ. [81, 84].

4. Переносимість між мовами та проєктами. Модель повинна зберігати здатність до узагальнення між різними мовами програмування, фреймворками й організаціями. Це досягається завдяки:

– виділенню мово-агностичних типів вузлів і ребер у гібридному поданні коду;

– використанню семантичних вбудовувань, навчених на багатомовних корпусах коду;

– нормалізації та масштабуванню метричних і еволюційних ознак на рівні проєкту.

5. Інкрементальність і сумісність з CI/CD. Модель має підтримувати інкрементальний режим роботи: при переході від ревізії t до $t+1$ повинні оновлюватися лише ознаки й рішення для змінених компонентів ΔC_t, а часові та ресурсні витрати мають масштабуватися як $O(|\Delta C_t|)$, а не $O(|C|)$. Це є критично важливим для інтеграції у пайплайни CI/CD та реалізації безперервного контролю якості [82].

2.2 Гібридне графове подання програмних компонентів

2.2.1 Формальне визначення Code Property Graph та уніфікація AST, CFG і PDG

Нехай програмний компонент (клас, модуль) c реалізує множину методів $M(c)$ у версії репозиторію з індексом часу t. Для кожного компонента будується абстрактне синтаксичне дерево $G_c^{AST} = (V_c^{AST}, E_c^{AST})$ [85], де:

– $G_c^{CFG} = (V_c^{CFG}, E_c^{CFG})$ – орієнтований граф потоку керування;

– $G_c^{PDG} = (V_c^{PDG}, E_c^{PDG})$ – граф потоку даних або залежностей;

Code Property Graph (CPG) для компонента c у версії t визначається як трійка:

$$G_{c,t}^{CPG} = (V_{c,t}, E_{c,t}, \varphi_{c,t}), \tag{2.9}$$

де $V_{c,t} = V_c^{AST} \cup V_c^{CFG} \cup V_c^{PDG}$ – уніфікована множина вузлів;

$E_{c,t} = E_c^{AST} \cup E_c^{CFG} \cup E_c^{PDG} \cup E_c^{SEM} \cup E_c^{REL}$ – множина ребер, що включає синтаксичні, керуючі, залежні, семантичні (наприклад, *CALL*, *OVERRIDES*) та структурно-архітектурні зв'язки (*CONTAINS*, *IMPLEMENTS* тощо);

$\varphi_{c,t}: V_{c,t} \to \mathcal{P}$ – відображення, яке зіставляє кожному вузлу вектор властивостей.

Таким чином, CPG виступає надграфом, що одночасно кодує синтаксис, потік керування, потік даних та обрані семантичні й еволюційні властивості коду [86]. Це дозволяє застосовувати єдиний графовий апарат (наприклад, графові нейронні мережі) до всіх рівнів аналізу – від окремих операторів до меж компонентів та проєкту. На рисунку 2.2 наведена базова схема організації гібридного CPG, що використовується в дисертаційній роботі.

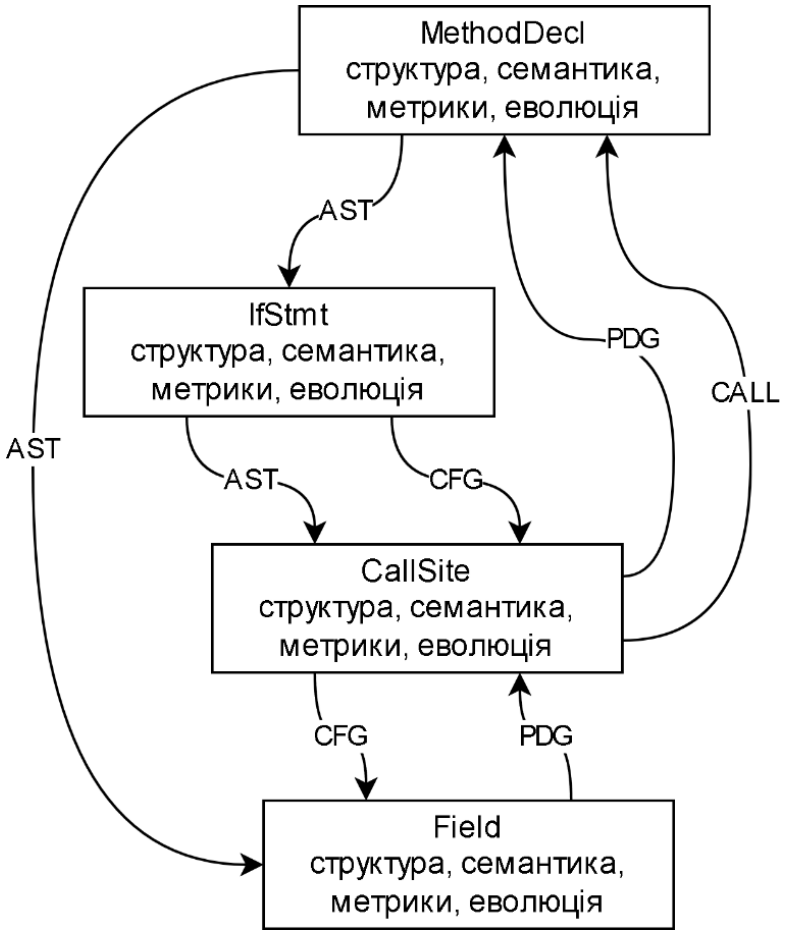

Рисунок 2.2 – Схематичне зображення гібридного Code Property Graph

2.2.2 Типи вузлів і ребер CPG для компонентів об’єктно-орієнтованого коду

Для об’єктно-орієнтованих систем множина вузлів $V_{c,t}$ подається як об’єднання кількох підтипів:

– структурні вузли класів та інтерфейсів: *ClassDecl*, *InterfaceDecl*, *EnumDecl*;

– вузли методів та полів: *MethodDecl*, *ConstructorDecl*, *FieldDecl*, *Parameter*;

– операторні та виразні вузли: *IfStmt*, *ForStmt*, *ReturnStmt*, *AssignExpr*, *CallExpr*;

– вузли псевдоінструкцій потоку керування: *Entry*, *Exit*, *Branch*, *Join*;

– вузли залежностей: *Def*, *Use*, *Phi* тощо [87].

Множина ребер $E_{c,t}$ включає принаймні такі класи зв’язків:

– AST-ребра (*AST_CHILD*, *NEXT_SIBLING*), які відтворюють вкладеність синтаксичних конструкцій;

– CFG-ребра (*FLOWS_TO*, *TRUE_EDGE*, *FALSE_EDGE*, *EXCEPTION_EDGE*), які описують можливі шляхи виконання;

– PDG-ребра (*DATA_DEP*, *CONTROL_DEP*), що фіксують залежності за даними та керуванням;

– семантичні ребра (*CALL*, *RETURNS*, *OVERRIDES*, *IMPLEMENTS*), які пов’язують виклики з деклараціями та моделюють поліморфізм;

– архітектурні та контейнерні зв’язки (*CONTAINS*, *USES*, *IMPORTS*), що відображають належність методів та класів до компонента й проєкту [88].

Для зручності подальшого аналізу на рівні компонентів вводиться індукований підграф

$$G_{C,t}^{CPG} = (V_{C,t}, E_{C,t}, \varphi_{C,t}), \tag{2.10}$$

де C – множина класів (компонентів);

$V_{C,t}$ – включає всі вузли методів та операторів, що належать цим класам, разом із відповідними ребрами.

Саме цей підграф використовується як вхід до моделі виявлення антипатернів у наступних підрозділах.

2.2.3 Каналізовані ознаки вузлів

Кожному вузлу $v \in V_{c,t}$ ставиться у відповідність чотириканальний вектор ознак

$$x_v = [x_v^{str} \parallel x_v^{sem} \parallel x_v^{met} \parallel x_v^{evo}], \tag{2.11}$$

де $\parallel$ – конкатенація векторів.

Структурний канал x_v^{str} містить дискретні та числові характеристики, пов'язані з позицією вузла в графі: тип вузла (one-hot або ембеддинг типу), глибина в AST, домінування в CFG (наприклад, бітові індикатори post-dominator), вхідний та вихідний ступені за різними типами ребер

Семантичний канал x_v^{sem} відображає вміст токенів, ідентифікаторів та коментарів, пов'язаних з вузлом. Для цього використовується попередньо натренована модель представлення коду (типу code2vec або GraphCodeBERT), що перетворює локальний контекст $ctx(v)$ у вектор

$$x_v^{sem} = f_{\text{LLM}}(ctx(v)) \in \mathbb{R}^{d_{sem}}, \tag{2.12}$$

де f_{LLM} – фіксований енкодер;

d_{sem}– розмірність семантичного простору [89].

Метричний канал x_v^{met} агрегує традиційні метрики на рівні методів і класів: показники CK-метрик (WMC, CBO, LCOM, RFC), метрики Halstead, базові структурні показники (LOC, кількість гілок, максимальна глибина вкладеності, число параметрів тощо) [90].

Для вузлів нижчого рівня (операторів) відповідні значення наслідуються від методу/класу або агрегуються локально.

Еволюційний канал x_v^{evo} кодує історію змін компонента на основі даних системи контролю версій [91].

До нього входять показники churn (сукупна кількість доданих/видалених рядків за останні h комітів), частота змін, вік компонента, частота ко-змін з іншими файлами, оцінки стабільності (наприклад, частка комітів, у яких компонент змінювався разом із тестами).

У таблиці 2.2 узагальнено основні групи ознак, що дозволяє чітко відокремити структурну, семантичну, метричну та еволюційну інформацію при подальшому проєктуванні моделі машинного навчання.

Таблиця 2.2 – Каналізовані ознаки вузлів гібридного CPG та приклади показників для кожного каналу

Канал ознак	Коротка характеристика	Приклади показників для вузлів CPG
Структурний	Формальна структура коду та позиція вузла в AST/CFG/PDG, локальний контекст керування.	тип вузла (*MethodDecl*, *IfStmt*, *CallSite*, *Field* тощо); глибина вузла в AST; in/out-ступінь у CFG/PDG; роль у блоці (вхідний/вихідний); наявність виняткових гілок; кількість дочірніх вузлів.
Семантичний	Змістовне наповнення коду, ідентифікаторів та викликів API, прихована семантика.	токенізовані ідентифікатори та літерали; bag-of-subtokens; ембеддинги оператора / методу з моделей типу code2vec / CodeBERT; сигнатура методу; імена викликаних API; типи параметрів і результату.
Метричний	Локальні кількісні характеристики якості та складності для вузла / пов’язаного методу.	LOC методу; цикломатична складність; глибина вкладеності умов і циклів; кількість параметрів; кількість звернень до полів; fan-in/fan-out для методу; співвідношення коду й коментарів.
Еволюційний	Динаміка змін вузла та пов’язаного компонента в історії репозиторію.	churn (кількість змінених рядків у комітах); вік вузла (час від першої появи); кількість комітів, що торкаються вузла/методу; число баг-фікс-комітів; частота ко-змін з іншими компонентами; кількість різних авторів змін.

2.2.4 Нормалізація та масштабування ознак на рівні проєкту

Через різні шкали та розподіли ознак (наприклад, LOC та churn мають важкі «хвости», а ембеддинги – майже нормальний розподіл) перед навчанням моделі виконується нормалізація на рівні всього проєкту. Для кожної числової ознаки f застосовується робастна стандартизація:

$$\hat{f} = \frac{f - \mathrm{med}\,(f)}{\mathrm{iqr}\,(f) + \varepsilon}, \tag{2.13}$$

де $\mathrm{med}\,(f)$ – медіана;

$\mathrm{iqr}\,(f)$ – міжквартильний розмах;

$\varepsilon > 0$ – мале число для запобігання діленню на нуль [92].

Такий підхід зменшує вплив викидів, характерних для великих промислових репозиторіїв. Семантичні ембеддинги x_v^{sem} додатково приводяться до одиничної норми

$$\tilde{x}_v^{sem} = \frac{x_v^{sem}}{\|x_v^{sem}\|_2 + \varepsilon}, \tag{2.14}$$

що полегшує подальшу агрегацію на рівні компонентів та проєкту.

Для забезпечення мовної агностичності вводиться єдина таксономія типів вузлів (наприклад, *Function*, *Loop*, *Conditional*, *MemberAccess*), до якої відображаються конкретні типи різних мов (Java, C#, Kotlin тощо). Це дає змогу застосовувати одну й ту ж модель до багатомовних проєктів без перенавчання на кожну мову окремо [93].

2.2.5 Інкрементальне оновлення гібридного графа при змінах у репозиторії

В умовах сучасних CI/CD-процесів оновлення CPG «з нуля» для кожного коміту є непрактичним.

Тому вводиться інкрементальна схема, за якої для кожного коміту Δ_t будується лише різниця графів

$$\Delta G_t^{CPG} = G_t^{CPG} \ominus G_{t-1}^{CPG}, \tag{2.15}$$

а повний граф оновлюється як

$$G_t^{CPG} = G_{t-1}^{CPG} \oplus \Delta G_t^{CPG}. \tag{2.16}$$

На практиці це означає, що:

– виконується аналіз *diff* репозиторію для визначення множини змінених файлів та методів;

– для них локально реконструюються відповідні фрагменти AST/CFG/PDG та оновлюються вузли і ребра;

– повторно обчислюються метричні ознаки для лише тих компонентів, які потрапили в область впливу (наприклад, клас і його безпосередні сусіди у графі викликів);

– еволюційні ознаки x_v^{evo} оновлюються на основі агрегованих показників по вікну останніх h комітів (sliding window).

Таким чином, гібридний граф залишається синхронізованим з реальним станом репозиторію при прийнятних витратах часу, що є критичним для інтеграції моделі виявлення антипатернів у конвеєр безперервної інтеграції.

2.3 Локальний рівень моделі: представлення методів та базових блоків

Локальний рівень моделі відповідає за детальне подання поведінки окремого методу та його базових блоків, що є носіями елементарних реалізаційних антипатернів (надмірна вкладеність, «шотган-модифікації» логіки, дублювання фрагментів тощо). На рисунку 2.3 наведена узагальнена схема локального рівня моделі.

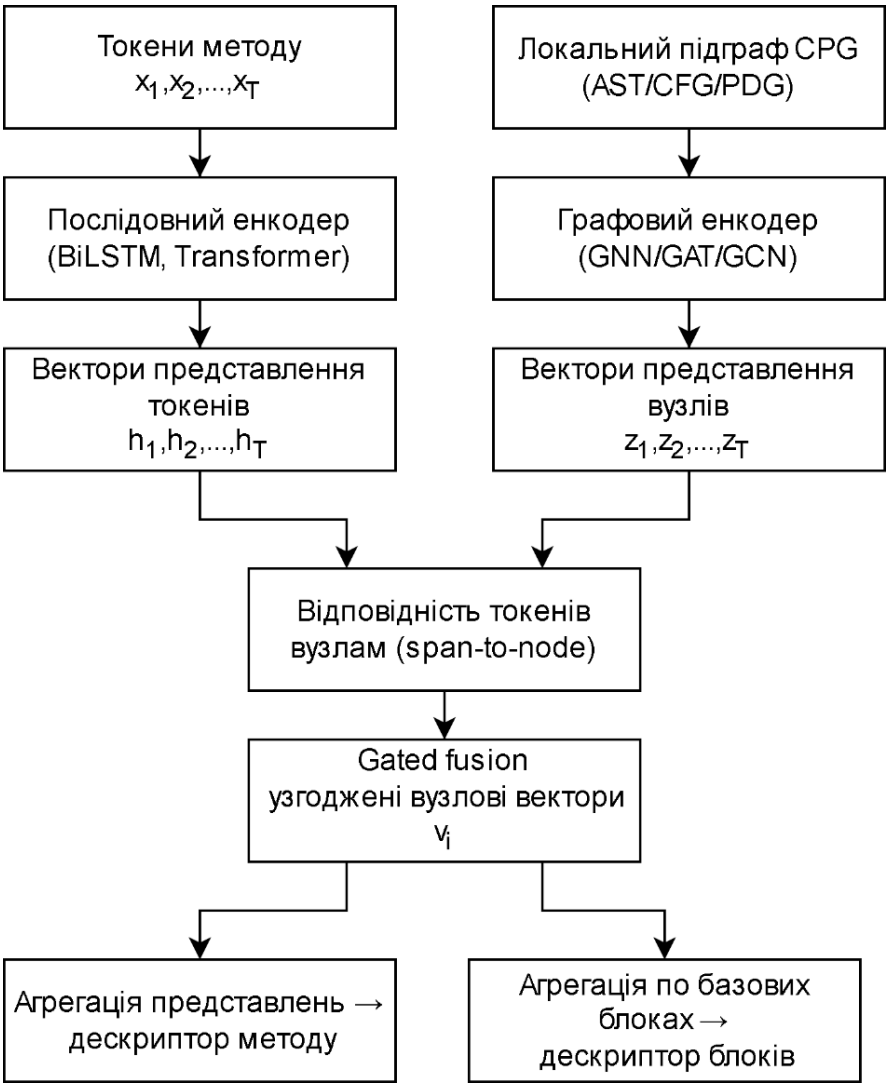

Рисунок 2.3 – Схематичне зображення конвеєру локального рівня

На цьому рівні поєднуються два типи представлень: послідовнісне (над токенами вихідного коду) та графове (над локальним підграфом CPG, індукованим методом). Подальша інтеграція цих представлень дає змогу сформувати інформативні локальні дескриптори, які використовуються в наступних рівнях моделі.

2.3.1 Послідовнісний енкодер вихідного коду для токенів і ідентифікаторів

Нехай метод m після лексичного аналізу задається послідовністю токенів

$$(w_1, w_2, \dots, w_T), \tag{2.17}$$

де кожний токен відповідає ключовому слову, оператору, ідентифікатору або спеціальному символу. Кожному токену ставиться у відповідність вектор

$$e_i = E(w_i) \in \mathbb{R}^{d_{emb}}, \tag{2.18}$$

де E – матриця вбудовувань, що включає окремі підпростори для типів токенів, лексем і, за потреби, розбитих субтокенів ідентифікаторів (*camelCase*, *snake_case*) [94].

Контекстуалізація цієї послідовності здійснюється за допомогою послідовнісного енкодера – бінапрямленої GRU або трансформера. У випадку бі-GRU отримуємо представлення

$$h_i = \overrightarrow{GRU}\left(e_i, \vec{h}_{i-1}\right) \| \overleftarrow{GRU}\left(e_i, \overleftarrow{h}_{i+1}\right), i = 1, \dots, T, \tag{2.19}$$

де $h_i \in \mathbb{R}^{d_{seq}}$ – контекстуальний вектор токена;

$\|$ – конкатенація прямого та зворотного станів [95].

Альтернативно може бути використаний трансформерний енкодер, у якому контекст формується за рахунок механізму самоуваги:

$$H = \text{TransformerEncoder}(E(W)), \tag{2.20}$$

де $H = (h_1, \dots, h_T)$ – послідовність вихідних векторів, що кодує як локальні, так і далекі залежності між токенами [94].

Обидва варіанти допускаються в рамках дисертаційної моделі; конкретна архітектура обирається експериментально.

2.3.2 Відповідність між токенами та вузлами CPG, механізм span-to-node

Щоб повернути багатий послідовнісний контекст у графове подання CPG, для кожного вузла локального підграфа G_m^{CPG} визначається відображення *span-to-*

node. Нехай $v \in V_m$ – вузол, що відповідає, наприклад, оператору *if*, виклику методу або блоку інструкцій. На етапі побудови CPG для нього зберігається проміжок токенів

$$\text{span}(v) = [s_v, e_v], 1 \leq s_v \leq e_v \leq T, \tag{2.21}$$

який відповідає фрагменту вихідного коду, з якого цей вузол було породжено. Тоді базове семантичне представлення вузла з послідовнісної модальності визначається як агрегат контексту токенів на відповідному проміжку, наприклад

$$z_v^{seq} = \frac{1}{e_v - s_v + 1} \sum_{i=s_v}^{e_v} h_i. \tag{2.22}$$

Для вузлів узагальненого типу (базові блоки, віртуальні вузли *Entry* або *Exit*) span може складатися з кількох неперервних інтервалів, які об'єднуються за допомогою зваженої або багатоголової уваги. Таким чином, послідовнісне подання коду не існує окремо, а безпосередньо «прив'язується» до вузлів CPG через механізм *span-to-node*, що забезпечує узгодженість двох представлень [96]. Для ілюстрації на прикладі Java-методу:

```
public int sum(int[] arr) {
    int result = 0;
    for (int value : arr) {
        result += value;
    }
    return result;
}
```

вузол, що відповідає циклу «for», отримає span, який охоплює токени «for», «(» , «int», «value», «:», «arr», «)» і тіло циклу; це дозволяє відобразити в одному векторі семантику як ітераційного патерну, так і операцій над колекцією.

2.3.3 Гетерогенна GNN на локальному підграфі з поширенням повідомлень з урахуванням семантики ребер

Локальний підграф $G_m^{CPG} = (V_m, E_m)$, індукований методом m, включає вузли різних типів та ребра кількох семантичних класів (AST, CFG, PDG тощо). Для обробки такої структури використовується гетерогенна графова нейронна мережа з відношеннєво-обізнаним поширенням повідомлень [97].

Нехай $\mathcal{R}$ – множина типів ребер (наприклад, *AST_CHILD*, *FLOWS_TO*, *DATA_DEP*). На кожному шарі ℓ представлення вузла оновлюється за правилом:

$$h_v^{(\ell+1)} = \sigma\left(W_0^{(\ell)} h_v^{(\ell)} + \sum_{r\in\mathcal{R}} \sum_{u\in\mathcal{N}_r(v)} \frac{1}{|\mathcal{N}_r(v)|} W_r^{(\ell)} h_u^{(\ell)}\right), \tag{2.23}$$

де $h_v^{(0)} = z_v^{seq} \parallel x_v^{str,met,evo}$ – початковий вектор вузла, який поєднує послідовнісні, структурні, метричні та еволюційні ознаки;

$\mathcal{N}_r(v)$ – сусіди вузла v за типом ребра r;

$W_r^{(\ell)}$– матриця перетворення для типу зв'язку r;

σ– нелінійність (наприклад, ReLU).

За потреби у модель вводяться механізми уваги над ребрами (подібні до R-GCN або GAT), коли коефіцієнти агрегації для сусідів вагуються залежно від типу зв'язку і поточного стану вузлів [98]. Це дозволяє, наприклад, сильніше підкреслювати залежності за керуванням при виявленні антипатернів, пов'язаних з надмірною складністю логіки, та залежності за даними – для антипатернів, що стосуються надлишкового стану [99].

2.3.4 Узгодження послідовнісного та графового подань на рівні вузлів

Після кількох шарів GNN для кожного вузла vмаємо графове представлення $h_v^{(L)}$, яке кодує контекст у локальному підграфі, та послідовнісне представлення

z_v^{seq}, що відображає лінійну структуру вихідного коду. Для їх узгодження використовується gated fusion:

$$g_v = \sigma\left(W_g[h_v^{(L)} \parallel z_v^{seq}] + b_g\right),$$
$$z_v = g_v \odot h_v^{(L)} + (1 - g_v) \odot z_v^{seq}, \tag{2.24}$$

де $g_v \in (0,1)^d$ – вектор «воріт»;

$\odot$ – поелементне множення;

z_v – підсумкове локальне представлення вузла [100].

У такий спосіб модель здатна автоматично балансувати внесок графового та послідовнісного контексту для різних типів вузлів: наприклад, для вузлів, що відповідають довгим виразам, більшу вагу може отримати послідовнісний канал, тоді як для вузлів *Entry* або *Exit* або базових блоків – графовий. Додатково може застосовуватися увага над токенами всередині span v, яка дозволяє фокусуватися на семантично важливих елементах (ідентифікатори, виклики API), не змінюючи загальної схеми узгодження.

2.3.5 Побудова локальних дескрипторів антипатернів на рівні методу та базового блока

На основі вузлових представлень z_v формується компактний дескриптор методу d_m та дескриптори базових блоків d_b, які надалі подаються до вищих рівнів моделі (компонентного та проєктного). Нехай V_m – множина вузлів, що належать методу m, а $V_b \subset V_m$ – вузли, які входять до базового блока b. Тоді дескриптори визначаються як

$$d_m = \text{ATTN}_m(\{z_v \mid v \in V_m\}), d_b = \text{ATTN}_b(\{z_v \mid v \in V_b\}), \tag{2.25}$$

де ATTN_m, ATTN_b – механізми вагового усереднення на основі уваги або, в простішому випадку, max/mean-pooling.

Отримані вектори d_m та d_b інтерпретуються як локальні багатовимірні дескриптори потенційних антипатернів: вони узагальнюють інформацію про структуру й семантику методу, його внутрішню складність, а також локальні патерни потоків керування і даних.

На наступних рівнях моделі ці дескриптори використовуються як вхідні ознаки для багаторівневого класифікатора антипатернів та механізмів оцінки невизначеності.

2.4 Рівень програмного компонента та проєктного контексту

Запропонована модель виходить за межі локального аналізу окремих методів і вводить рівень представлення *програмного компонента* (класу, модуля) та його *проєктного контексту*. На цьому рівні CPG використовується для побудови індукованих підграфів компонентів, подальшої агрегації локальних дескрипторів у вектори компонентів та моделювання взаємодії між ними у вигляді графа компонентів, над яким виконується багатокрокове поширення повідомлень [101, 102].

2.4.1 Індуковані підграфи компонентів у CPG

Нехай $G = (V, E)$ – гібридний Code Property Graph проєкту, де вершини $v \in V$ відповідають вузлам AST/CFG/PDG, а ребра $e \in E$ – синтаксичним, контрольним, потоко-даним та викличним зв’язкам. Для кожного компонента c (класу або модуля) визначається множина вузлів

$$V_c = \{v \in V \mid v \text{ належить до оголошення, тіла або поля компонента } c\} \quad (2.26)$$

та індукований підграф

$$G_c = G[V_c] = (V_c, E_c), E_c = \{(u, v) \in E \mid u, v \in V_c\}. \quad (2.27)$$

Таким чином, G_c зберігає усі внутрішні залежності між методами, полями та контролем потоку всередині компонента, але ізолює його від решти проєкту. Це дозволяє навчати локальні графові представлення, які природно узагальнюються на компоненти різного розміру [86].

2.4.2 Агрегація локальних представлень у вектор компонента

На локальному рівні для кожного методу m та базового блока b вже побудовані дескриптори h_m та h_b.

Для компонента c множини методів $M(c)$ та базових блоків $B(c)$ агрегуються за допомогою увагою-орієнтованого пулінгу [94, 97]. Спочатку обчислюється зважене середнє локальних представлень:

$$h_c^{\text{loc}} = \sum_{m \in M(c)} \alpha_m \, h_m + \sum_{b \in B(c)} \beta_b \, h_b, \tag{2.28}$$

де ваги α_m, β_b визначаються механізмом уваги

$$\alpha_m = \frac{\exp\left(s(h_m)\right)}{\sum_{m' \in M(c)} \exp\left(s(h_{m'})\right)}, \qquad \beta_b = \frac{\exp\left(s(h_b)\right)}{\sum_{b' \in B(c)} \exp\left(s(h_{b'})\right)}, \tag{2.29}$$

а функція оцінки $s(\cdot)$ моделюється невеликою нейронною мережею.

До локального представлення компонента додаються агреговані метричні та еволюційні ознаки:

$$\phi_c^{\text{metr}} \in \mathbb{R}^{d_m}, \phi_c^{\text{evol}} \in \mathbb{R}^{d_e}, \tag{2.30}$$

що включають, наприклад, сумарний та середній LOC, середню глибину вкладеності, показники зв’язності, churn, частоту змін та інші характеристики, отримані з історії репозиторію [91].

Фінальне векторне подання компонента задається як

$$z_c = W_c\,[\,h_c^{\text{loc}} \;\|\; \phi_c^{\text{metr}} \;\|\; \phi_c^{\text{evol}}\,], \tag{2.31}$$

де W_c – матриця перетворення;

$\|$ – операція конкатенації.

2.4.3 Граф взаємодії компонентів

Для моделювання архітектурного контексту формується граф взаємодії компонентів

$$G^{\text{proj}} = (C, E^{\text{proj}}), \tag{2.32}$$

де кожна вершина $c \in C$ – програмний компонент;

ребра $e = (c_i, c_j)$ – різні типи взаємодій між ними [103].

Типові канали включають:

– виклики (call edges) – наявність викликів методів компонента c_j із компонентів c_i;

– імпорти та залежності (import / dependency edges) – використання типів, інтерфейсів або пакетів іншого компонента;

– ко-зміни (co-change edges) – часте спільне змінювання файлів компонентів c_i та c_j в одному коміті;

– спільна власність (co-ownership edges) – перетин авторів або команд, що супроводжують компоненти.

Кожне ребро має вектор атрибутів $\psi_{ij} \in \mathbb{R}^{d_e}$, який включає частоту викликів, інтенсивність ко-змін, середній інтервал між змінами тощо. На рисунку 2.4 узагальнено показано приклад такого графа для невеликої системи.

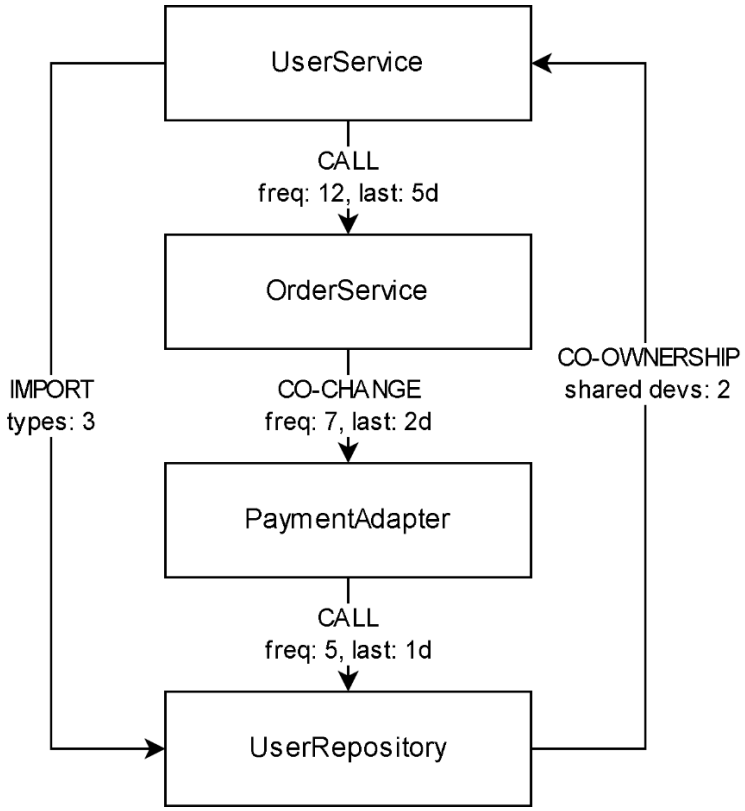

Рисунок 2.4 – Структура компонентного графа з еволюційними атрибутами ребер, яка використовується як вхід для проєктного рівня GNN

2.4.4 Проєктний рівень: передача повідомлень між компонентами з еволюційними атрибутами

Над графом G^{proj} застосовується багатошарова графова нейронна мережа, яка оновлює представлення компонентів з урахуванням їхніх взаємозв'язків та еволюційних атрибутів ребер [101–103]. На t-му шарі оновлення для компонента c має вигляд

$$h_c^{(t+1)} = \sigma(W_{\text{self}} h_c^{(t)} + \sum_{c' \in \mathcal{N}(c)} g(\psi_{cc'}) \odot W_{\text{neigh}} h_{c'}^{(t)}), \tag{2.33}$$

де $h_c^{(0)} = z_c$, $\mathcal{N}(c)$ – множина сусідніх компонентів;

$\sigma(\cdot)$ – нелінійність;

W_{self}, W_{neigh} – навчувані матриці.

Функція $g(\psi_{cc'})$ реалізує ваги ребра (наприклад, через невелику мережу або нормалізовану лінійну функцію), що дозволяє посилювати вплив стабільних і часто взаємодіючих пар компонентів та послаблювати шумові взаємозв'язки.

Таким чином модель явно враховує проєктний контекст компонента: компоненти, що формують щільно пов'язаний підграф з високою інтенсивністю козмін, можуть бути ідентифіковані як кандидати на комплексний антипатерн.

2.4.5 Ієрархічна увага між локальним, компонентним та проєктним рівнями

Після декількох шарів передачі повідомлень отримується проєктно-контекстуалізоване представлення h_c^{proj}. Для кожного компонента доступні три рівні подання:

– локальне h_c^{loc} (агрегація методів і блоків);

– внутрішньокомпонентне h_c^{comp} (результат GNN на G_c);

– проєктне h_c^{proj} (результат GNN на G^{proj}).

Щоб сформувати узгоджений дескриптор компонента для задачі виявлення антипатернів, використовується ієрархічна увага [100, 102]. Нехай

$$H_c = \{h_c^{\text{loc}}, h_c^{\text{comp}}, h_c^{\text{proj}}\}. \tag{2.34}$$

Для кожного рівня $r \in \{\text{loc,comp,proj}\}$ обчислюється вага

$$\gamma_r = \frac{\exp\left(u^{\top}\tanh\left(W_r h_c^r\right)\right)}{\sum_{r'}\exp\left(u^{\top}\tanh\left(W_{r'} h_c^{r'}\right)\right)}, \tag{2.35}$$

де u та W_r – параметри уваги.

Фінальне представлення компонента задається як

$$\tilde{h}_c = \sum_{r\in\{\text{loc,comp,proj}\}} \gamma_r\, h_c^r. \tag{2.36}$$

Таким чином, для антипатернів, які проявляються переважно у локальній структурі (наприклад, довгі методи), модель може збільшувати γ_{loc}, тоді як для архітектурних та еволюційних антипатернів (надмірні ко-зміни, приховані залежності) – посилювати внесок γ_{proj}. Така ієрархічна схема дозволяє адаптивно комбінувати інформацію з різних рівнів абстракції без ручного налаштування ваг, що є критично важливим для узагальнення моделі між проєктами з різною структурою та еволюційними характеристиками [104].

2.5 Класифікаційна підсистема та open-set обробка невизначеності

На попередніх етапах багаторівнева модель формує узгоджене векторне представлення компонента $\tilde{h}_c$, яке відображає локальні, компонентні та проєктні характеристики. Класифікаційна підсистема накладає на цей простір подань задачу мультиміткової класифікації з явним урахуванням невизначеності та open-set сценаріїв, коли в потоці аналізу з'являються компоненти з невідомими або нетиповими антипатернами [105, 106].

2.5.1 Формулювання задачі як мультиміткової класифікації компонентів

Нехай $\mathcal{C}$ – множина програмних компонентів, $\mathcal{K} = \{1, \dots, K\}$ – множина розглядуваних класів антипатернів («запахи» коду, «запахи» дизайну, архітектурні антипатерни). Для кожного компонента $c \in \mathcal{C}$ задається бінарний вектор міток

$$y_c = (y_{c,1}, \dots, y_{c,K}, y_{c,0}) \in \{0,1\}^{K+1}, \tag{2.37}$$

де $y_{c,k} = 1$ – позначає наявність k-го антипатерну;

$y_{c,0} = 1$ – інтерпретується як клас «без антипатерну».

Для навчальних прикладів виконується додаткове обмеження

$$y_{c,0} = 1 \;\Rightarrow\; \forall k \in \mathcal{K}: y_{c,k} = 0, \tag{2.38}$$

проте для компонента з декількома антипатернами допускається $y_{c,k} = 1$ для кількох k. Таким чином, задача є мультимітковою (компонент може одночасно належати до кількох класів антипатернів), але з одним спеціальним «чистим» класом. Класифікаційна голова моделі задається як відображення

$$f: \tilde{h}_c \mapsto z_c = (z_{c,1}, \dots, z_{c,K}, z_{c,0}) \in \mathbb{R}^{K+1}, \tag{2.39}$$

де $z_{c,k}$ – логіт для відповідного класу.

Для мультиміткової постановки використовується сигмоїдна активація

$$p_{c,k} = \sigma(z_{c,k}) = \frac{1}{1 + \exp(-z_{c,k})}, k = 0, \dots, K, \tag{2.40}$$

що інтерпретується як оцінка ймовірності наявності кожного антипатерну (або «чистоти») незалежно від інших [107].

2.5.2 Вихідний шар: ймовірнісні оцінки антипатернів та «без антипатерну»

Вихідний шар моделі повертає вектор $\mathbf{p}_c = (p_{c,1}, \dots, p_{c,K}, p_{c,0})$. Для прийняття рішення використовується набір порогів $\tau = (\tau_1, \dots, \tau_K, \tau_0)$. Компоненту cприсвоюються мітки

$$\hat{y}_{c,k} = \mathbb{I}(p_{c,k} \geq \tau_k), k = 1, \dots, K, \tag{2.41}$$

а клас «без антипатерну» активується, якщо

$$p_{c,0} \geq \tau_0 \text{та} \forall k \in \mathcal{K}: p_{c,k} < \tau_k. \tag{2.42}$$

У таблиці 2.3 наведені приклади поєднань $\max_k p_{c,k}$, $p_{c,0}$, показників невизначеності та відповідної дії системи: автоматичне виявлення антипатерну,

маркування як «чистий» компонент, або постановка в чергу на ручну перевірку чи open-set класифікацію.

Таблиця 2.3 – Правила інтерпретації вихідних ймовірностей/режиму open-set

$\max_k p_{c,k}$ (антипатерни, $k > 0$)	$p_{c,0}$ («без антипатерну»)	Показники невизначеності	Рекомендована дія системи
висока (наприклад, ≥ 0,80)	низька (≤ 0,20)	низька ентропія; низький energy-score; мала дисперсія	Автоматично зафіксувати наявність антипатерну класу $\arg\max_k p_{c,k}$; сформувати попередження без залучення експерта.
низька (≤ 0,20)	висока (наприклад, ≥ 0,90)	низька ентропія; низький energy-score; мала дисперсія	Маркувати компонент як «чистий»; не створювати попередження, опціонально понижувати пріоритет подальших перевірок.
Середня (0,40–0,70)	середня (0,30–0,60)	помірна ентропія; середній energy-score; помірна дисперсія	Вважати передбачення ненадійним; поставити компонент у чергу на ручний аудит чи рев'ю коду.
висока (≥ 0,80)	низька (≤ 0,20)	висока ентропія; високий energy-score; підвищена дисперсія	Інтерпретувати як потенційний open-set випадок (нетиповий або новий антипатерн); вимагати ручної перевірки, не виконувати автоматичне виправлення.
низька (≤ 0,30)	низька (≤ 0,50)	дуже висока ентропія; високий energy-score; висока дисперсія	Вважати зразок «поза розподілом» (out-of-distribution); маркувати як кандидат open-set і передавати лише в ручний режим аналізу без автоматичних висновків.

У практичному застосуванні пороги τ_k можуть відрізнятися для різних антипатернів (наприклад, більш консервативні для потенційно критичних

дефектів) і налаштовуються на валідаційних вибірках. Для частини компонентів, які генерують низьку впевненість, модель свідомо утримується від автоматичного присвоєння будь-якої з міток.

2.5.3 Енергетичні та ентропійні показники невизначеності

Одних лише порогів на $p_{c,k}$ недостатньо для відсікання нетипових або поза-розподільних (out-of-distribution, OOD) компонентів [105]. Тому на виході класифікаційної голови додатково обчислюються енергетичні та ентропійні показники невизначеності.

Енергетичний показник (energy-based score) для компонента c визначається як [106]:

$$E_c = -T \cdot \log \sum_{k=0}^{K} \exp\left(\frac{z_{c,k}}{T}\right), \tag{2.43}$$

де $T > 0$ – температурний параметр.

Низька енергія (великий модуль від'ємного значення) відповідає впевненим передбаченням, тоді як висока енергія вказує на потенційний OOD-зразок.

Прогностична ентропія розподілу $\mathbf{p}_c$:

$$H_c = -\sum_{k=0}^{K} p_{c,k} \log p_{c,k}. \tag{2.44}$$

Високе значення H_c свідчить про «розмиту» впевненість між класами, що характерно як для складних, так і для поза-розподільних компонентів [105].

Епістемічна дисперсія оцінюється за допомогою ансамблів моделей або стохастичних проходів (наприклад, dropout-інференс) [108, 109]. Нехай $\left\{p_{c,k}^{(m)}\right\}_{m=1}^{M}$ – прогнозні ймовірності M незалежних моделей або стохастичних реалізацій. Тоді

$$\bar{p}_{c,k} = \frac{1}{M}\sum_{m=1}^{M} p_{c,k}^{(m)}, \qquad \mathrm{Var}_{c,k} = \frac{1}{M}\sum_{m=1}^{M} (p_{c,k}^{(m)} - \bar{p}_{c,k})^2, \tag{2.45}$$

та агрегований показник епістемічної невизначеності

$$U_c^{\mathrm{epi}} = \sum_{k=0}^{K} \mathrm{Var}_{c,k}. \tag{2.46}$$

Поєднання E_c, H_c та U_c^{epi} дозволяє розрізняти «важкі, але знайомі» компоненти та справжні OOD-випадки, у яких модель не має достатнього досвіду.

На рисунку 2.5 наведена схематична залежність між енергією, ентропією та рішеннями моделі. Область низької енергії та низької ентропії відповідає впевненим передбаченням, область високої енергії або високої ентропії – open-set або відмова.

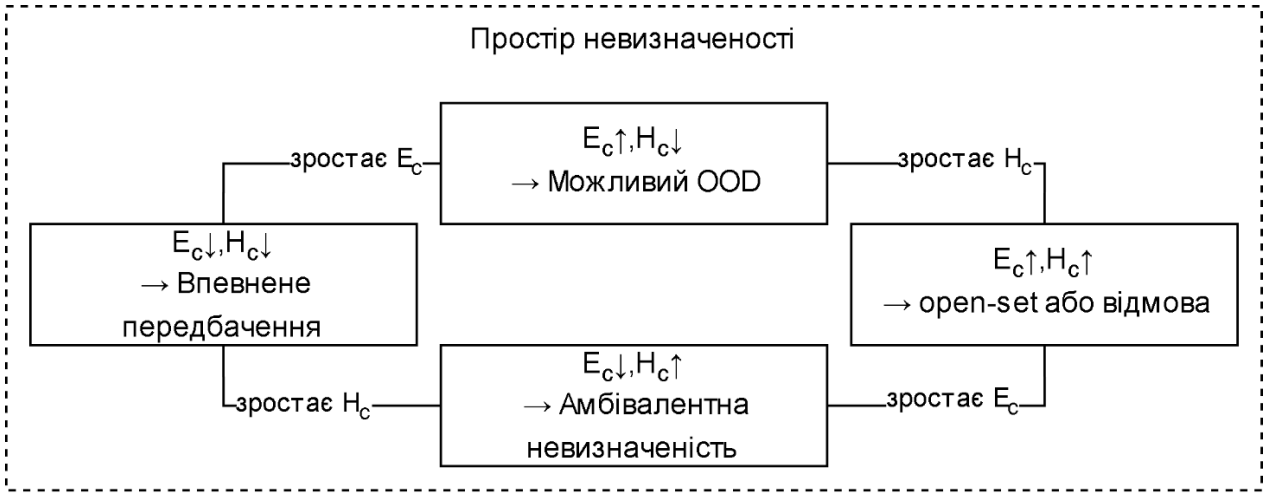

Рисунок 2.5 – Робочі області класифікатора в просторі енергетичного та ентропійного показників невизначеності.

2.5.4 Open-set або low-confidence режим: селективне передбачення та відмова

Для безпечного використання у CI/CD-конвеєрі класифікаційна підсистема працює в режимі селективного передбачення: модель видає рішення лише тоді,

коли інтегральний показник довіри перевищує заздалегідь встановлений поріг [107]. Нехай

$$S_c = f_{\text{conf}}(E_c, H_c, U_c^{\text{epi}}) \tag{2.47}$$

– скалярний score довіри, який зменшується зі зростанням невизначеності (наприклад, лінійна або нелінійна комбінація нормалізованих показників). Для заданого рівня покриття $\alpha \in (0,1]$ обирається поріг θ_α так, щоб приблизно частка прикладів з $S_c \geq \theta_\alpha$ дорівнювала α на валідаційній вибірці [107].

Правило прийняття рішення має вигляд:

– якщо $S_c \geq \theta_\alpha$ і $\exists k: p_{c,k} \geq \tau_k$ – компонент маркується виявленими антипатернами;

– якщо $S_c \geq \theta_\alpha$ та $\forall k: p_{c,k} < \tau_k$, але $p_{c,0} \geq \tau_0$ – компонент вважається «чистим»;

– якщо $S_c < \theta_\alpha$ – модель утримується від рішення (abstention), а компонент відноситься до open-set потоку для ручного аналізу або додаткового модулю кластеризації/збагачення навчальної вибірки.

Такий підхід дозволяє явно контролювати компроміс між ризиком (частка помилкових автоматичних рішень) та покриттям (частка компонентів, для яких приймається рішення), що особливо важливо в контексті технічного боргу та критичних змін у промислових системах [107, 110].

2.5.5 Функція втрат та регуляризація

Функція втрат комбінує терміни, спрямовані на якісну класифікацію відомих антипатернів, розмежування відомих та OOD-зразків і калібрування ймовірностей.

Крос-ентропія для відомих класів. Для навчальної множини $\mathcal{D}_{\text{in}}$ використовується бінарна крос-ентропія:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{|\mathcal{D}_{\text{in}}|} \sum_{c \in \mathcal{D}_{in}} \sum_{k=0} y_{c,k} \log p_{c,k} + (1 - y_{c,k}) \log (1 - p_{c,k})). \tag{2.48}$$

Енергетичний margin для розмежування in-distribution та OOD. Для навчального набору генеруються або збираються приклади $\mathcal{D}_{\text{out}}$, що моделюють поза-розподільні компоненти (наприклад, фрагменти з інших доменів або синтетично згруповані модулі) [106]. Втрати на енергії задаються як

$$\mathcal{L}_{\text{E}} = \frac{1}{\mid \mathcal{D}_{\text{in}} \mid} \sum_{c \in \mathcal{D}_{\text{in}}} \max\left(0, E_c - m_{\text{in}}\right) + \frac{1}{\mid \mathcal{D}_{\text{out}} \mid} \sum_{c \in \mathcal{D}_{\text{out}}} \max\left(0, m_{\text{out}} - E_c\right), \quad (2.49)$$

де $m_{\text{in}} < m_{\text{out}}$ – пороги-маржі для відомих і OOD-зразків.

Перший доданок притискає енергію відомих компонентів до низьких значень, другий – підштовхує енергію OOD-компонентів до високих.

Контрастивні або калібрувальні доданки. Для покращення як роздільної здатності представлень, так і каліброваності ймовірностей застосовуються додаткові регуляризатори [108, 110]:

– контрастивна складова $\mathcal{L}_{\text{con}}$, що зближує векторні подання $\tilde{h}_c$ для компонентів з однаковими антипатернами та відштовхує – для компонентів з різними антипатернами;

– калібрувальний термін $\mathcal{L}_{\text{cal}}$, наприклад, у вигляді Brier-втрат або температурного масштабування, який зменшує розбіжність між оцінками $p_{c,k}$ та емпіричними частотами [110].

Підсумкова функція втрат має вигляд

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda_{\text{E}}\mathcal{L}_{\text{E}} + \lambda_{\text{con}}\mathcal{L}_{\text{con}} + \lambda_{\text{cal}}\mathcal{L}_{\text{cal}}, \quad (2.50)$$

де $\lambda_{\text{E}}, \lambda_{\text{con}}, \lambda_{\text{cal}} \geq 0$ – вагові коефіцієнти

Така композиція дає змогу одночасно досягати високої якості мультиміткової класифікації відомих антипатернів, надійного відсікання поза-розподільних компонентів в open-set режимі та коректної інтерпретованості ймовірнісних оцінок.

2.6 Алгоритм побудови та функціонування багаторівневої моделі в індустріальному середовищі

Запропонована багаторівнева модель повинна бути інтегрована у типові процеси розробки програмного забезпечення (CI/CD, код-рев'ю, аналіз pull request-ів), забезпечуючи автоматичне виявлення антипатернів без істотного погіршення швидкодії конвеєра та з наданням інтерпретованих результатів для розробників [111, 112].

В даному дисертаційному дослідженні троеба формалізувати алгоритм побудови та функціонування моделі в індустріальному оточенні та оцінити обчислювальну складність та описано механізми інкрементальної обробки й пояснюваності.

2.6.1 Загальний конвеєр обробки

Робота моделі організовується як конвеєрна процедура над знімком репозиторію у фіксованому коміті r. На вхід алгоритму надходить SHA-ідентифікатор коміту та конфігурація аналізу (список гілок, фільтри директорій, пороги open-set режиму).

Побудова або оновлення CPG. Для кожного файлу, що відповідає підтримуваним мовам, генеруються проміжні подання (AST, CFG, PDG), після чого вони уніфікуються у Code Property Graph $G_r^{\mathrm{CPG}} = (V_r, E_r)$ із гетерогенними типами вузлів та ребер [65, 66]. При повторному запуску над еволюцією репозиторію використовується інкрементальне оновлення графа: для файлів, що не змінювалися між комітами r_{prev} та r, відповідні фрагменти CPG та попередні подання вузлів кешуються.

Обчислення ознак та нормалізація. Для кожного вузла $v \in V_r$ формуються чотири групи ознак:

$$x_v = [x_v^{\mathrm{str}} \parallel x_v^{\mathrm{sem}} \parallel x_v^{\mathrm{met}} \parallel x_v^{\mathrm{evo}}], \tag{2.51}$$

де x_v^{str} – структурні характеристики;

x_v^{sem} – семантичні ембеддинги токенів;

x_v^{met} – локальні метрики (наприклад, довжина методу, цикломатична складність);

x_v^{evo} – еволюційні ознаки, отримані з VCS (churn, вік, ко-зміни) [38, 45].

Для кожної групи застосовується нормалізація (z-score, робастна нормалізація за медіаною) з параметрами, оціненими на тренувальній вибірці [92].

На рисунку 2.6 наведена схема індустріального конвеєра функціонування багаторівневої моделі в індустріальному середовищі.

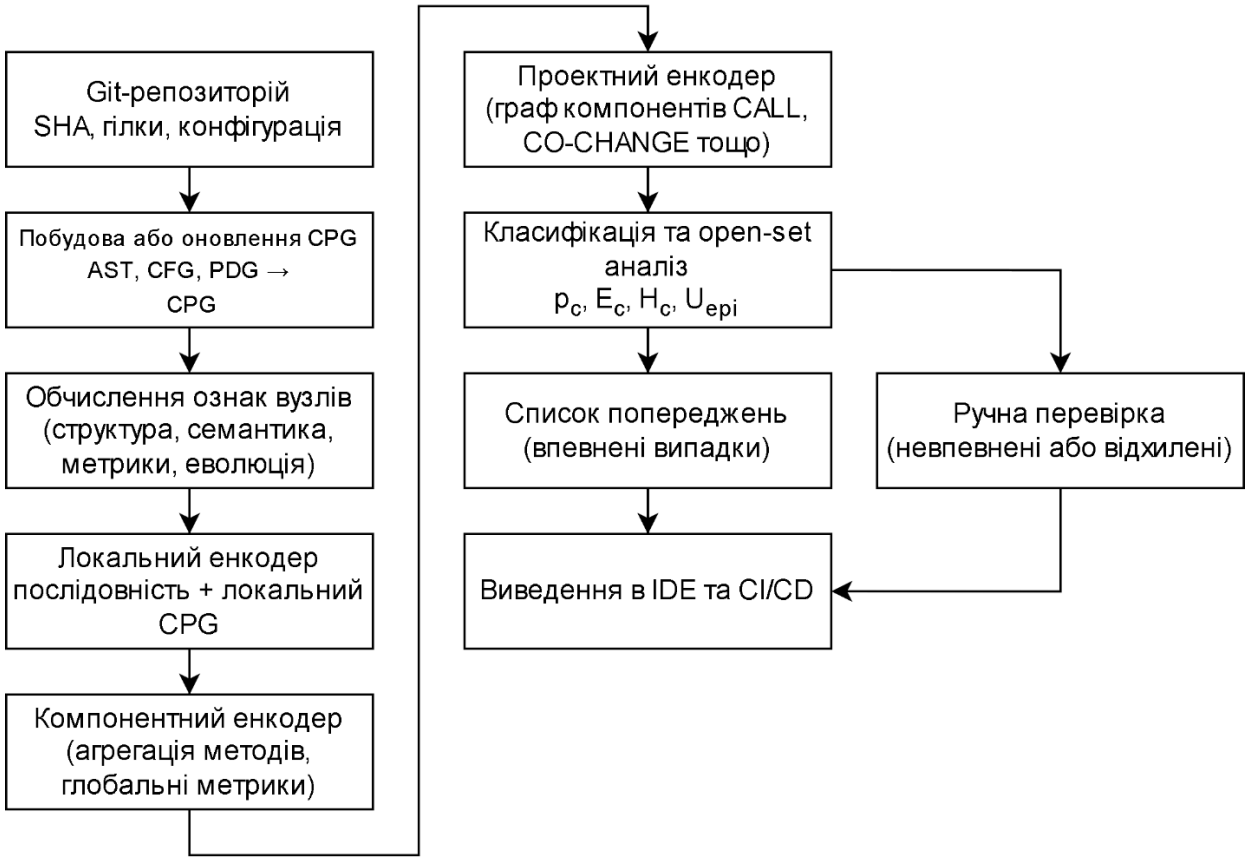

Рисунок 2.6 – Загальний конвеєр функціонування багаторівневої моделі в індустріальному середовищі

Проходи локального, компонентного та проєктного енкодерів. На локальному рівні послідовнісний енкодер і GNN формують представлення методів та базових блоків h_m, h_b.

Далі компонентний енкодер агрегує ці представлення у вектор компонента h_c з урахуванням внутрішнього CPG-підграфа та глобальних метричних або еволюційних ознак. Нарешті, проєктний енкодер виконує передачу повідомлення у графі взаємодії компонентів, отримуючи узгоджені контекстуалізовані подання $\tilde{h}_c$ (що було показано в підпунктах 2.4.2–2.4.4) [101–104].

Класифікація з open-set-головкою та формування списку попереджень. Для кожного компонента cобчислюються логіти z_c, ймовірності $p_{c,k}$ для класів антипатернів та показники невизначеності (енергія, ентропія, епістемічна дисперсія). Open-set-головка застосовує правило селективного передбачення: компоненти з надійними прогнозами потрапляють у список автоматичних попереджень, тоді як зразки з низькою впевненістю переводяться у режим «потребує ручної перевірки» (що було показано в підпункті 2.5) [105–107].

2.6.2 Оцінка обчислювальної складності

Для аналізу масштабованості моделі розглянемо такі параметри:

– $N_v = |V_r|$ – кількість вузлів CPG у проєкті;

– $N_e = |E_r|$ – кількість ребер;

– L – кількість шарів GNN на локальному та проєктному рівнях;

– $T_{\max}$ – максимальна довжина послідовності токенів для методу;

– M_c – кількість компонентів (класів / модулів);

– d – розмірність латентного простору подань.

Один прямий прохід GNN з відношеннєво-обізнаним поширенням повідомлень має асимптотичну складність

$$\mathcal{O}(L \cdot (N_v + N_e) \cdot d), \tag{2.52}$$

оскільки на кожному шарі виконується оновлення станів вузлів та агрегація повідомлень з усіх суміжних ребер [103]. Для послідовнісних енкодерів (бі-GRU, Transformer з фіксованою шириною) оцінка для одного методу має вигляд

$$\mathcal{O}(T_{\max} \cdot d^2), \tag{2.53}$$

а для всіх методів проєкту – $\mathcal{O}(N_m \cdot T_{\max} \cdot d^2)$, де N_m – кількість методів [94, 113].

Агрегація локальних представлень у вектори компонентів та подальша передача повідомлень у графі взаємодії компонентів має складність

$$\mathcal{O}(L_c \cdot (M_c + E_c) \cdot d), \tag{2.54}$$

де L_c – кількість шарів GNN на рівні компонентів;

E_c – кількість ребер графа взаємодії.

Як видно з таблиці 2.4, основною «вузькою» ланкою є GNN-обробка великих CPG для великих монолітних систем.

Для індустріальних репозиторіїв із сотнями тисяч вузлів використовуються батчинг, семплінг сусідства або кластеризація графа (наприклад, Cluster-GCN) [114], що знижує реальну часову складність до майже лінійної від обсягу аналізованого фрагмента.

Таблиця 2.4 –Асимптотична складність основних етапів конвеєра

Етап конвеєра	Пояснення параметрів	Часова складність	Пам'яттєва складність
Побудова / оновлення гібридного CPG за знімком репозиторію	N_v – кількість вузлів CPG; N_e – кількість ребер (AST/CFG/PDG/CALL).	$O(N_v + N_e)$	$O(N_v + N_e)$
Обчислення та нормалізація ознак для вузлів і ребер	Ті самі N_v, N_e, кількість каналів ознак вважається сталою.	$O(N_v + N_e)$	$O(N_v)$
Локальний енкодер методів	N_m – кількість методів; L – середня довжина послідовності на метод (токени, шляхи, оператори).	$O(N_m \cdot L)$	$O(N_m \cdot L)$
Компонентний енкодер (агрегація методів у вектор компонента)	M_c – кількість компонентів; L_c – середня кількість методів у компоненті.	$O(M_c \cdot L_c)$	$O(M_c \cdot L_c)$

Продовження таблиці 2.4

Етап конвеєра	Пояснення параметрів	Часова складність	Пам'яттєва складність
GNN-енкодер над гібридним CPG (локальний / проєктний рівень)	N_v – активні вузли; N_e – відповідні ребра; L – кількість шарів GNN.	$O\big(L \cdot (N_v + N_e)\big)$	$O(N_v + N_e)$
Message passing на графі взаємодії компонентів	M_c – вузли графа компонентів; L – кількість шарів message passing.	$O(L \cdot M_c)$	$O(M_c)$
Класифікаційна підсистема з open-set-головкою (для всіх компонентів)	M_c – кількість компонентів; кількість класів антипатернів вважається сталою	$O(M_c)$	$O(M_c)$
Формування списку попереджень та фільтрація за порогами невизначеності	M_c – кількість компонентів; показники невизначеності обчислюються з логітів/ймовірностей класифікатора	$O(M_c)$	$O(M_c)$

Класифікаційна голова та обчислення енергетичних або ентропійних показників невизначеності є лінійними за K (кількістю класів) і не домінують загальну складність.

2.6.3 Інкрементальний режим для pull request-ів

Для інтеграції у процеси розробки за pull request-ами (PR) модель працює в інкрементальному режимі, коли повна побудова CPG та повний прохід енкодерів виконуються лише періодично (наприклад, ночами або на релізних гілках), а для окремих PR обробляються лише змінені фрагменти [17, 22].

Нехай Δr – PR, що переводить репозиторій з коміту r_{base} до стану r_{PR}. На першому кроці обчислюється множина змінених файлів $\mathcal{F}_\Delta$ та індукований підграф

$$G_\Delta^{\text{CPG}} = G_{r_{\text{PR}}}^{\text{CPG}}[V_\Delta], \tag{2.55}$$

де V_Δ – вузли, що відповідають зміненим методам, базовим блокам та безпосереднім околам у CPG.

Для незмінених вузлів використовується кеш попередніх подань h_v^{cached}, збережених під час аналізу r_{base}. Інкрементальний алгоритм виконує:

– локальне оновлення послідовнісних і графових подань лише для вузлів із V_Δ;

– переобчислення векторів компонентів h_c тільки для тих компонентів, які містять змінені методи;

– обмежена передача повідомлень у графі взаємодії компонентів на радіус 1–2 від змінених компонентів, що дозволяє відобразити локальний вплив змін, не перераховуючи всю модель.

Таким чином, фактична складність у режимі PR масштабується з $|V_\Delta|$ та кількістю заторкнутих компонентів, які у практиці значно менші за N_v та M_c [111, 113]. Це дає змогу інтегрувати модель у перевірки PR без значного збільшення часу збірки.

2.6.4 Інтерпретованість результатів

Для прийняття рішень щодо рефакторингу та обробки антипатернів розробникам потрібні не лише «чорні» попередження, а й зрозумілі пояснення щодо причин спрацювання моделі [73]. У запропонованій моделі інтерпретованість забезпечується на трьох рівнях.

Ієрархічна увага. На етапі узгодження локальних, компонентних та проєктних подань використовується механізм ієрархічної уваги, який дозволяє отримати ваги $\alpha_{m|c}$ для методів усередині компонента, ваги $\alpha_{b|m}$ для базових блоків у методі та ваги $\alpha_{c|P}$ для компонентів у проєктному контексті. Для кожного попередження ці ваги використовуються для виділення «hot spots» – конкретних методів та блоків, що зробили найбільший внесок у прогнозовані антипатерни.

Ваги каналів ознак. У ф'южн-шарах модель оперує вагами каналів $\beta^{\text{str}}, \beta^{\text{sem}}, \beta^{\text{met}}, \beta^{\text{evo}}$, що визначають відносну важливість структурних, семантичних, метричних та еволюційних ознак. Для кожного компонента формується розклад:

$$\beta_c = (\beta_c^{\text{str}}, \beta_c^{\text{sem}}, \beta_c^{\text{met}}, \beta_c^{\text{evo}}), \tag{2.56}$$

який показує, чи пов’язане спрацювання моделі переважно з нетиповою структурою коду, відхиленням метрик (наприклад, значне перевищення порогових значень), чи з «підозрілою» еволюційною історією (часті виправлення, нестабільні зміни) [77, 83].

«Картка пояснення» для IDE або CI. На основі вищезгаданих механізмів формується компактна «картка пояснення» для кожного попередження, яка включає:

– тип(и) виявлених антипатернів та рівень впевненості;

– топ-3 методи та базові блоки з найбільшими вагами уваги;

– суттєві метрики (довжина методу, coupling, churn тощо), що вийшли за типові діапазони;

– короткий огляд еволюційних подій (кількість і характер останніх комітів, розподіл авторів) [72].

У середовищі IDE картка відображається як інтерактивна панель, пов’язана з відповідними ділянками коду, а у CI/CD – як розширений запис у логах або веб-інтерфейсі перевірок, що дозволяє трасувати кожне попередження до конкретних фрагментів коду та історії змін.

Це знижує «когнітивний бар’єр» прийняття рекомендацій моделі та сприяє її прийняттю в командах розробки [73, 111].

2.7 Експериментальне дослідження багаторівневої моделі виявлення антипатернів

Експериментальне дослідження має на меті кількісно оцінити якість багаторівневої моделі виявлення антипатернів, порівняти її з наявними підходами, проаналізувати стійкість до невідомих зразків, міжмовний та міжпроєктний перенос, а також підтвердити практичну придатність у DevOps-сценаріях.

Додатково виконано якісний аналіз помилок та пояснюваності рішень моделі у порівнянні з експертними оцінками.

2.7.1 Набори даних та протоколи експериментів

Опис поліглотного JVM-корпусу. Для оцінювання запропонованої моделі сформовано поліглотний JVM-корпус, що включає відкриті репозиторії на мовах Java, Kotlin та Scala з доменів корпоративних застосунків, бібліотек та інструментів розробки.

До корпусу відібрано проєкти з тривалою історією змін (понад 3 роки) та мінімальною кількістю зірок/форків, що забезпечує достатній обсяг еволюційних даних [38, 46].

Статистику корпусу – кількість репозиторіїв, компонентів (класів/модулів), методів, середню та медіанну довжину методів, розподіл мов – узагальнено в таблиці 2.5.

Таблиця 2.5 – Огляд поліглотного JVM-корпусу

Мова	Кількість проєктів	Кількість компонентів (класів/модулів)	Кількість методів	Кількість рядків коду (KLOC)	Частка компонентів, %
Java	45	38 200	310 500	5 950	72,4
Kotlin	18	8 750	63 800	1 420	16,6
Scala	12	5 770	34 600	810	11,0
Разом	75	52 720	408 900	8 180	100,0

Як видно з таблиці, частка Java-компонентів переважає, проте суттєва присутність Kotlin та Scala дозволяє оцінити міжмовний перенос у подальших експериментах.

На рисунку 2.7 наведена гістограма розподілу довжини методів (у рядках коду) для Java, Kotlin і Scala з накладеними квантильними кривими.

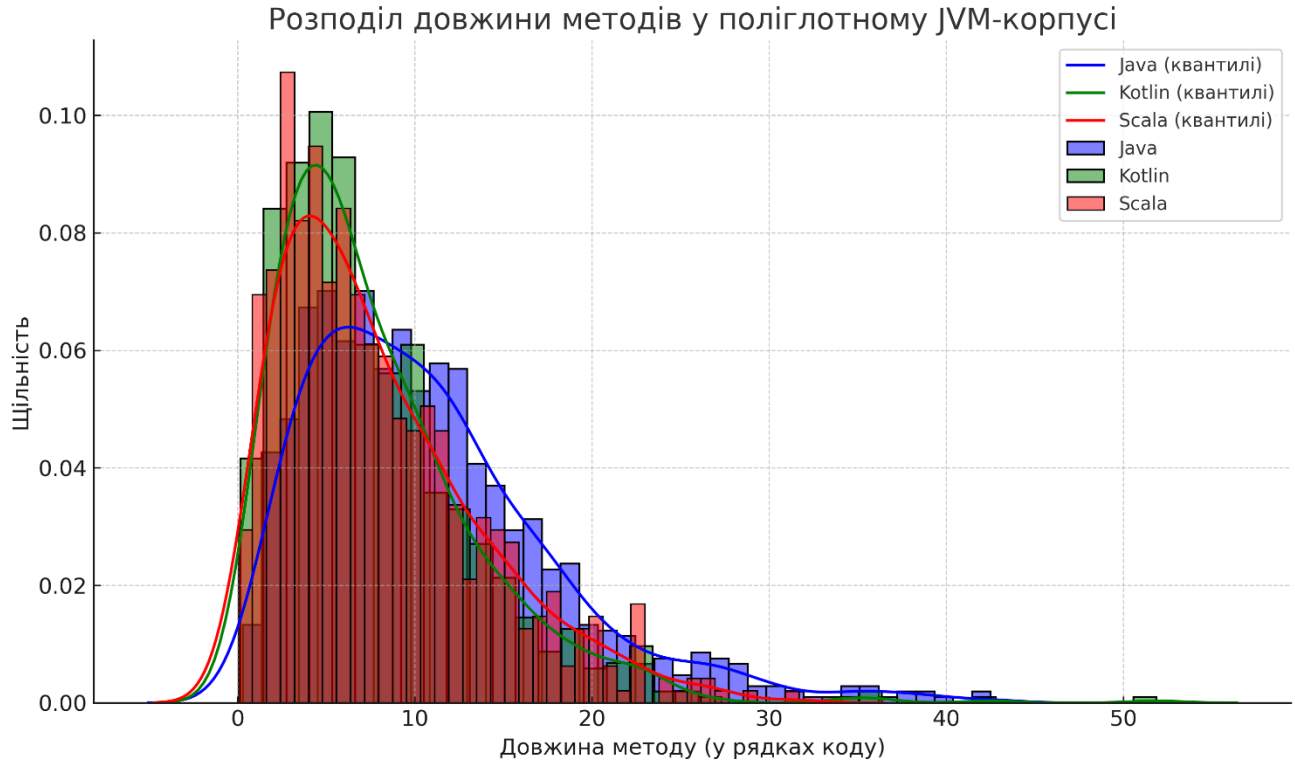

Рисунок 2.7 – Розподіл довжини методів у поліглотному JVM-корпусі

Формування розмітки антипатернів. Розмітка антипатернів здійснювалася в кілька етапів. Спочатку до кожного репозиторію були застосовані детектори на базі правил, що реалізують класичні набори правил для «Long Method», «God Class», «Feature Envy», «Data Class» та змінних антипатернів типу «Shotgun Surgery» [36, 37]. Далі, за історією змін та дефектів виділялися еволюційні шаблони (компоненти з високим churn, повторними дефектними комітами та частими рефакторингами), які слугували додатковими «м'якими» сигналами наявності антипатернів [45].

На третьому етапі виконано вибірковий ручний аудит підмножини компонентів із суперечливими або низько-конфідентними мітками, що дозволило скоригувати помилки інструментів на базі правил та уточнити складні випадки (перекриття кількох антипатернів у одному компоненті).

Схеми розбиття даних. Щоб уникнути витоків інформації та переоцінки якості, застосовано дві комплементарні схеми розбиття. У міжпроєктному сценарії усі компоненти проєкту повністю належать одному зі сплітів (train/val/test), що

моделює застосування моделі до нових репозиторіїв. У часово-усвідомленому сценарії розбиття виконується за часовою віссю:

$$t_{\text{train}} < t_{\text{val}} < t_{\text{test}}, \tag{2.57}$$

де всі коміти у валідаційному та тестовому наборах хронологічно пізніші за тренувальні, що наближує умови до реального використання у довготривалих проєктах [66].

На рисунку 2.8 зображено схематичне зображення часово-усвідомленого розбиття.

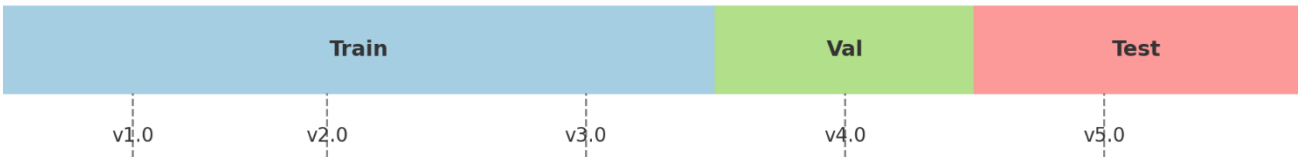

Рисунок 2.8 – Часово-усвідомлене розбиття корпусу для імітації еволюції проєктів

Сценарії оцінювання. Розглядаються чотири основні сценарії оцінювання:

– closed-set – всі типи антипатернів присутні в тренувальному наборі, а тест містить лише відомі класи;

– open-set – один або кілька типів антипатернів приховуються під час навчання, а на етапі тестування з’являються як «невідомі»;

– міжмовний перенос – модель навчається переважно на Java-компонентах і тестується на Kotlin/Scala;

– міжпроєктний перенос – усі тестові проєкти відсутні в тренувальному наборі.

У подальших підрозділах ці сценарії позначаються як Closed, Open, Cross-lang та Cross-project відповідно.

2.7.2 Базові підходи та варіанти запропонованої моделі

Правила та метрики. Першою групою базових підходів є класичні статичні аналізатори запахів коду, що реалізують набір метрик та евристик, подібних до описаних у роботах Lanza–Marinescu та Marinescu [36, 37].

Для кожного антипатерну використовуються фіксовані пороги (наприклад, порогові значення для WMC, CBO, LOC), а вихідна оцінка перетворюється на бінарну мітку. Такі інструменти виступають природною «мінімальною базою» для порівняння автоматизованих ML-підходів [75].

Спрощені ML-моделі. Друга група базових моделей включає:

– AST-only GNN – графову нейронну мережу, що працює лише з AST-поданнями без CFG/PDG та еволюційних ознак;

– текстову модель коду – трансформер, навчений на сирих токенізованих файлах (без структурної інформації) за схемою багатоміткової класифікації;

– метрик-орієнтований класифікатор – градієнтний бустинг над традиційними метриками об’єктно-орієнтованого дизайну [39];

– історико-орієнтований ранжувальник – модель градуйованого ранжування (learning-to-rank), яка використовує лише еволюційні та дефектні ознаки (частота виправлень, час до першої помилки тощо) [45, 65].

Ці варіанти дозволяють відокремлено оцінити внесок структурних, текстових, метричних та еволюційних ознак.

Абляційні варіанти гібридної моделі. Для аналізу архітектурних рішень розглядаються абляційні варіанти гібридної моделі:

– без еволюційних ознак (x^{evo} видалені з вектора ознак);

– без проєктного рівня (усувається передача повідомлень у графі взаємодії компонентів);

– без open-set-головки (модель працює лише в режимі звичайної softmax-класифікації);

– без ієрархічної уваги (просте усереднення локальних подань).

Порівняння повної та абляційних моделей дозволяє кількісно оцінити важливість кожного рівня представлення та каналу ознак.

2.7.3 Результати виявлення антипатернів у closed-set сценарії

Узагальнені метрики якості. Для оцінювання якості у closed-set-сценарії використовуються macro-AUPRC, macro-F1 та FPR@95TPR, обчислені у багатомітковій постановці як середні значення по класах. Нехай K – кількість типів антипатернів, тоді

$$\text{macro-F1} = \frac{1}{K}\sum_{k=1}^{K} \text{F1}_k \text{ ,macro-AUPRC} = \frac{1}{K}\sum_{k=1}^{K} \text{AUPRC}_k. \qquad (2.58)$$

Показник FPR@95TPR визначається як частота хибних спрацювань при фіксованому рівні повноти $\text{TPR} = 0{,}95$ і широко застосовується для оцінки компромісу між повнотою та надійністю виявлення [115]. У таблиці 2.6 наведено порівняння запропонованої моделі, абляційних варіантів та базових підходів у міжпроєктному розбитті. Як видно з таблиці, повна багаторівнева модель демонструє найвищі macro-AUPRC та macro-F1 при суттєво нижчому FPR@95TPR порівняно з інструментами на базі правил та простими ML-базовими моделями.

Покласовий аналіз базових антипатернів. Покласовий аналіз показує, що для «Long Method» та «God Class» запропонована модель особливо виграє від поєднання структурних та еволюційних ознак: macro-F1 для цих класів зростає на 5–10 процентних пунктів у порівнянні з AST-only GNN і метрик-орієнтованим класифікатором.

Для «Feature Envy» та «Data Class» найбільший внесок дають текстові та семантичні представлення коду, тоді як для «Shotgun Surgery»-like суттєву роль відіграє проєктний рівень, який враховує розподіл змін (рис. 2.9).

Таблиця 2.6 – Результати у closed-set-сценарії

Модель	Macro-AUPRC	Macro-F1	FPR@95TPR
Метрики/правила (Sonar-like)	0,48	0,41	0,22
AST-only GNN	0,53	0,47	0,21
Текстова модель (BERT)	0,56	0,50	0,20
Метрик-орієнтований XGBoost	0,59	0,55	0,20
Історико-орієнтований ранжувальник	0,57	0,51	0,21
Гібридна модель (без еволюційних ознак)	0,65	0,60	0,19
Гібридна модель (без проєктного рівня)	0,66	0,61	0,19
Гібридна модель (без open-set голови)	0,68	0,62	0,18
Повна гібридна модель (CPG + канали + open-set)	0,71	0,64	0,18

Вплив окремих каналів ознак та рівнів моделі. Абляційний аналіз (табл. 2.7) демонструє, що вилучення еволюційних ознак найбільше погіршує якість для антипатернів, пов’язаних зі стабільністю («Shotgun Surgery»-like), тоді як вилучення проєктного рівня негативно впливає на антипатерни, де важливі міжкомпонентні залежності (високий coupling, «God Class»).

Таблиця 2.7 – Вплив абляцій на якість виявлення антипатернів у closed-set-сценарії

Вилучений компонент	Macro-F1	Macro-AUPRC	Коментар щодо впливу
– еволюційні ознаки (–evo)	0.60	0.65	Зниження для Shotgun Surgery-like
– проєктний рівень (–project)	0.61	0.66	Падіння якості для God Class
– open-set головка (–open-set)	0.62	0.68	Незначне зниження; критично для OSR
– ієрархічна увага (–attention)	0.61	0.67	Зменшення інтерпретованості
Без аболяції (повна модель)	0.64	0.71	Найкраща якість серед усіх конфігурацій

Вимкнення open-set-головки незначно змінює closed-set-метрики, але в подальшому істотно погіршує open-set-стійкість.

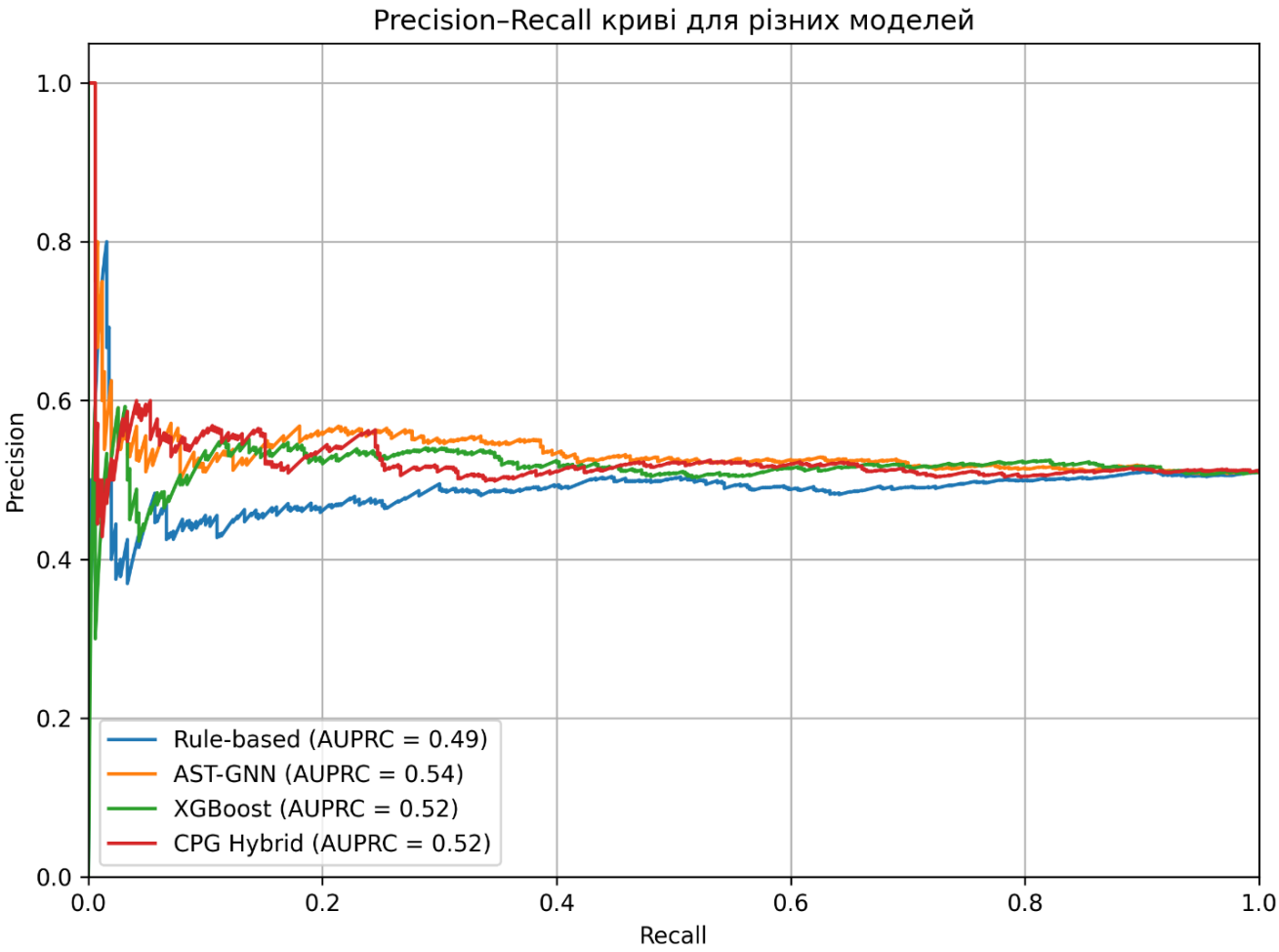

Рисунок 2.9 – Криві precision–recall для сумарного класу «будь-який антипатерн» для різних моделей

2.7.4 Дослідження open-set-стійкості та каліброваності невизначеності

Протоколи з прихованими класами та «незнайомими» зразками. Для оцінювання open-set-стійкості застосовано два типи протоколів.

У першому з них один із базових антипатернів (наприклад, «Shotgun Surgery»-like) повністю вилучається із тренувального набору та розглядається як «невідомий» клас на тесті.

У другому до тестового набору додаються компоненти з проєктів, що належать іншим мовам або доменам (наприклад, утиліти для наукових обчислень), які у тренуванні не використовувалися.

У обох випадках завдання open-set-детекції зводиться до відокремлення відомих класів (антипатерни + «без антипатерну») від невідомих зразків [51, 91].

Показники AUROC-OSR, TNR@TPR та ECE. Якість open-set-детекції оцінюється показниками AUROC-OSR та TNR@TPR (true negative rate при заданому рівні повноти для відомих класів).

Крім того, аналізується каліброваність ймовірнісних оцінок за Expected Calibration Error (ECE):

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)|, \quad (2.59)$$

де B_m – m-ий бін за довірою;

$\mathrm{acc}(B_m)$ – емпірична точність у біні;

$\mathrm{conf}(B_m)$ – середня передбачена ймовірність;

n – загальна кількість зразків [116, 117].

Результати, наведені в таблиці 2.8, показують, що енергетична open-set-головка з використанням energy-based score та глибоких ансамблів зменшує ECE та підвищує AUROC-OSR порівняно зі звичайною softmax-класифікацією.

Таблиця 2.8 – Показники AUROC-OSR, TNR@TPR та ECE для різних стратегій оцінки невизначеності

Метод оцінки невизначеності	AUROC-OSR ↑	TNR@TPR=95% ↑	ECE ↓	Коментар
Softmax confidence	0.81	0.57	0.102	Високий ризик помилкових спрацьовувань
Entropy thresholding	0.84	0.61	0.089	Краще виявлення «невідомих», але слабке калібрування
Energy-based (DEMTR)	0.88	0.68	0.065	Поліпшена чутливість до out-of-distribution зразків
Deep ensembles + energy	0.91	0.73	0.043	Найкраща AUROC-OSR та ECE; рекомендовано для CI/CD

Поведінка моделі в режимі селективного передбачення. У режимі селективного передбачення для кожного компонента обчислюється показник довіри $s(c)$, а модель повертає прогноз лише якщо $s(c) \geq \tau$; інакше відбувається «відмова» (abstention). Аналіз залежності recall від частки відмов демонструє, що, відмовляючись від 10–20 % найбільш невпевнених прогнозів, можна істотно знизити частоту хибних спрацювань, зберігши при цьому високий рівень покриття критичних антипатернів [107, 118] (рис. 2.10).

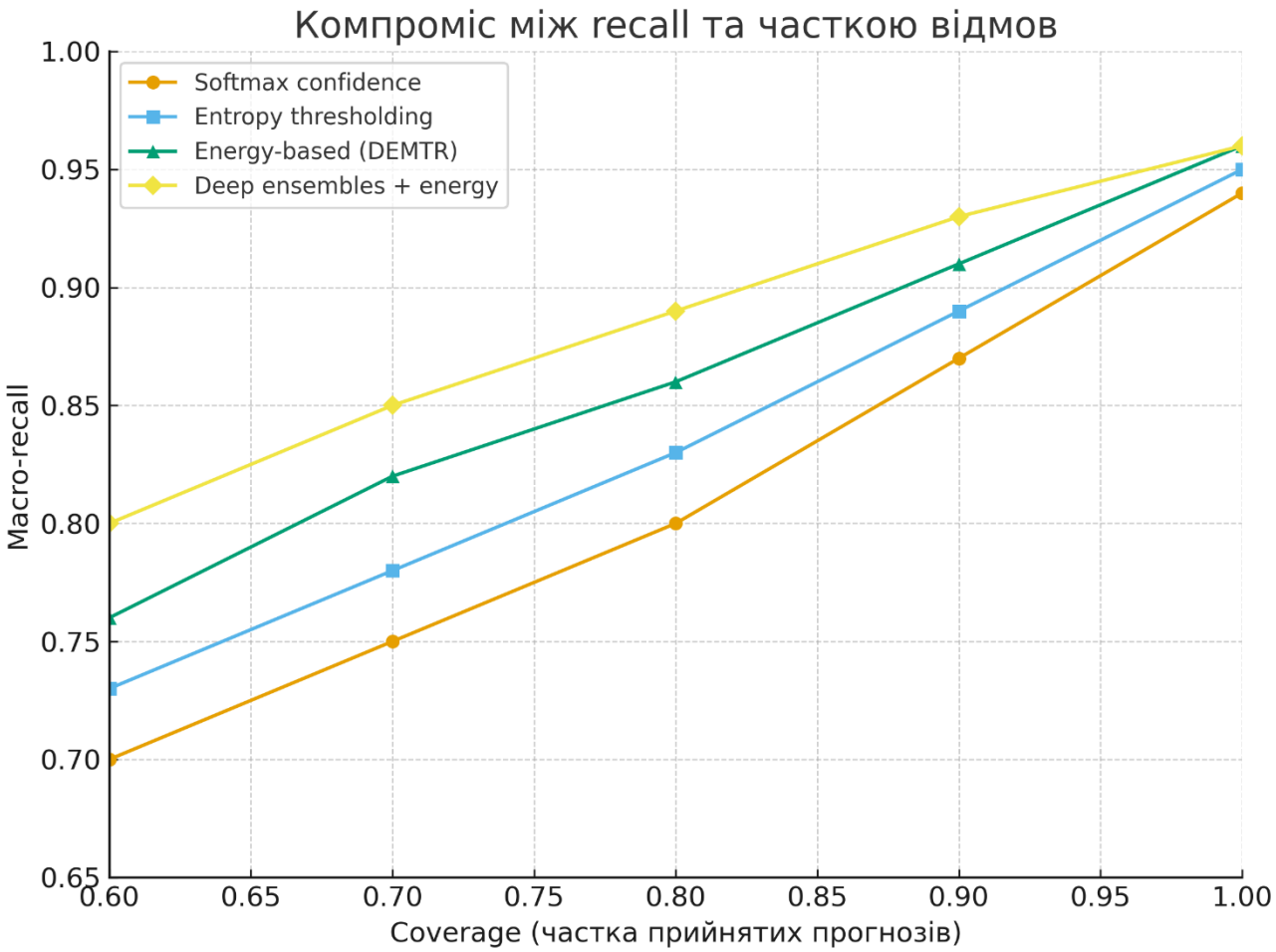

Рисунок 2.10 – Компроміс між recall та часткою відмов у режимі селективного передбачення

2.7.5 Аналіз міжмовного та міжпроєктного переносу

Сценарій навчання на Java та тестування на Kotlin/Scala. Для оцінювання міжмовного переносу модель навчалася лише на Java-проєктах, а тестування

проводилося на Kotlin- та Scala-компонентах. Незважаючи на синтаксичні відмінності, гібридне графове подання коду та спільний простір ознак дозволяють зберегти прийнятну якість: macro-F1 знижується помірно порівняно з налаштуванням для однієї мови, але значно перевищує результати текстових та метрик-орієнтованих базових моделей, які гірше переносяться між мовами.

Порівняння з одно-видовими моделями. У таблиці 2.9 подано порівняння результатів багаторівневої моделі та одно-видових моделей (AST-only, текстових, метрик-орієнтованих) у Cross-lang та Cross-project сценаріях.

Таблиця 2.9 – macro-F1 та macro-AUPRC у Cross-lang та Cross-project сценаріях для різних типів моделей

Тип моделі	Cross-lang macro-F1	Cross-lang macro-AUPRC	Cross-project macro-F1	Cross-project macro-AUPRC
AST-only GNN	0.42	0.45	0.46	0.49
Текстова модель (BERT)	0.38	0.41	0.44	0.46
Метрик-орієнтований XGBoost	0.39	0.43	0.47	0.51
Повна гібридна модель	0.52	0.56	0.60	0.63

Запропонована модель показує найменший розрив між within-project та cross-project якістю, що свідчить про кращу узагальнюваність на нові репозиторії [92, 119].

Вплив нормалізації ознак. Додаткові експерименти показують, що нормалізація ознак із використанням статистик у розрізі проєктів (project-wise normalization) покращує стабільність результатів у міжпроєктних експериментах, зменшуючи чутливість моделі до відмінностей у масштабах метрик між проєктами. Така стратегія зменшує варіацію macro-F1 між різними тестовими репозиторіями, що є важливою властивістю для практичного розгортання в неоднорідних портфелях систем [92, 120].

2.7.6 Оцінювання інкрементальної продуктивності та придатності до CI/CD

Час інференсу та пікова пам'ять для pull request-ів. Для оцінки придатності до інтеграції у CI/CD-конвеєри вимірювалися час інференсу та пікове споживання пам'яті для pull request-ів трьох категорій: *small* (до 50 змінених рядків), *medium* (50–300 рядків) та *large* (понад 300 рядків).

Завдяки інкрементальному аналізу, час обробки виявився близьким до лінійного відносно кількості змінених вузлів CPG, а для small/medium PR у більшості проєктів не перевищував кількох секунд, що є прийнятним для pipeline-ів з суворими обмеженнями на тривалість [68, 111] (табл. 2.10).

Таблиця 2.10 – Час інференсу та споживання пам'яті

Категорія PR	Час інференсу, с	Пікове споживання пам'яті, МБ
Small (≤50 рядків)	1,3	280
Medium (50–300 рядків)	3,8	410
Large (>300 рядків)	8,5	610

Вплив глибини GNN та локального енкодера. Окремі вимірювання показали, що збільшення глибини GNN понад 3–4 шари дає лише незначне покращення якості при помітному зростанні затримок.

Аналогічно, розширення локального текстового енкодера (збільшення розмірності прихованого шару, використання більш важких трансформерів) швидко призводить до зростання часу обробки без пропорційного виграшу у метриках (рис. 2.11). Це дозволило сформувати рекомендації щодо «легких» конфігурацій моделі для CI/CD-застосувань.

Рекомендації для DevOps-сценаріїв. На основі отриманих результатів сформульовано практичні рекомендації:

– використовувати інкрементальний режим аналізу для PR-перевірок, залишаючи повний аналіз усієї кодової бази для періодичних нічних або релізних запусків;

– обмежувати глибину GNN до 3–4 шарів та застосовувати компактні трансформерні енкодери;

– налаштовувати селективний режим так, щоб модель повертала попередження для 70–80 % компонентів, відсікаючи найбільш невпевнені випадки.

Такі налаштування дозволяють узгодити якість виявлення антипатернів з вимогами до часу виконання та стабільності CI/CD-процесів [72, 74].

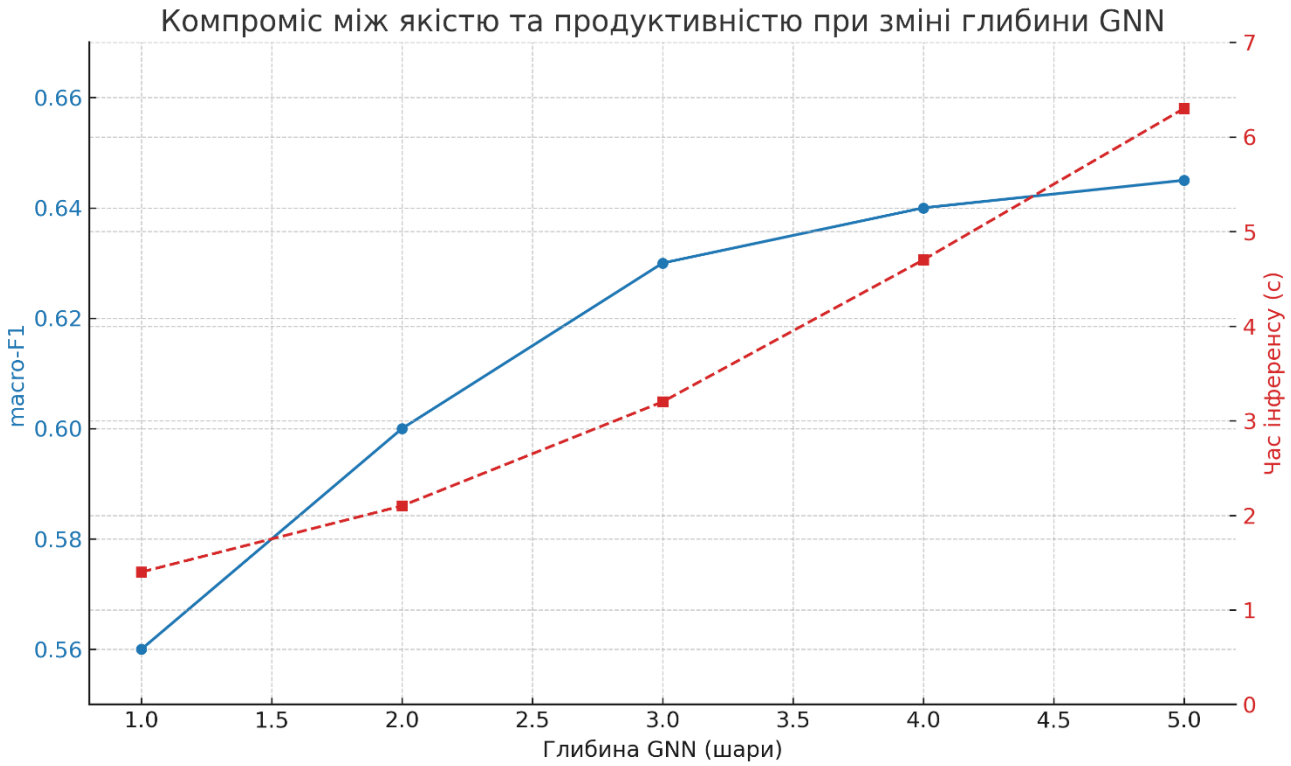

Рисунок 2.11 – Компроміс між якістю та продуктивністю при зміні глибини GNN

2.7.7 Якісний аналіз роботи моделі

Приклади правильно виявлених антипатернів. Якісний аналіз зосереджувався на репрезентативних прикладах правильно виявлених антипатернів. Для «Long Method» модель часто виділяла методи з надмірною логікою обробки та великою кількістю вкладених умов, де детектори на базі правил іноді не спрацьовували через жорсткі пороги. Візуалізація відповідних CPG-підграфів показує, що увага

моделі зосереджується на піддеревах із високою цикломатичною складністю та частими викликами до зовнішніх сервісів (рис. 2.12).

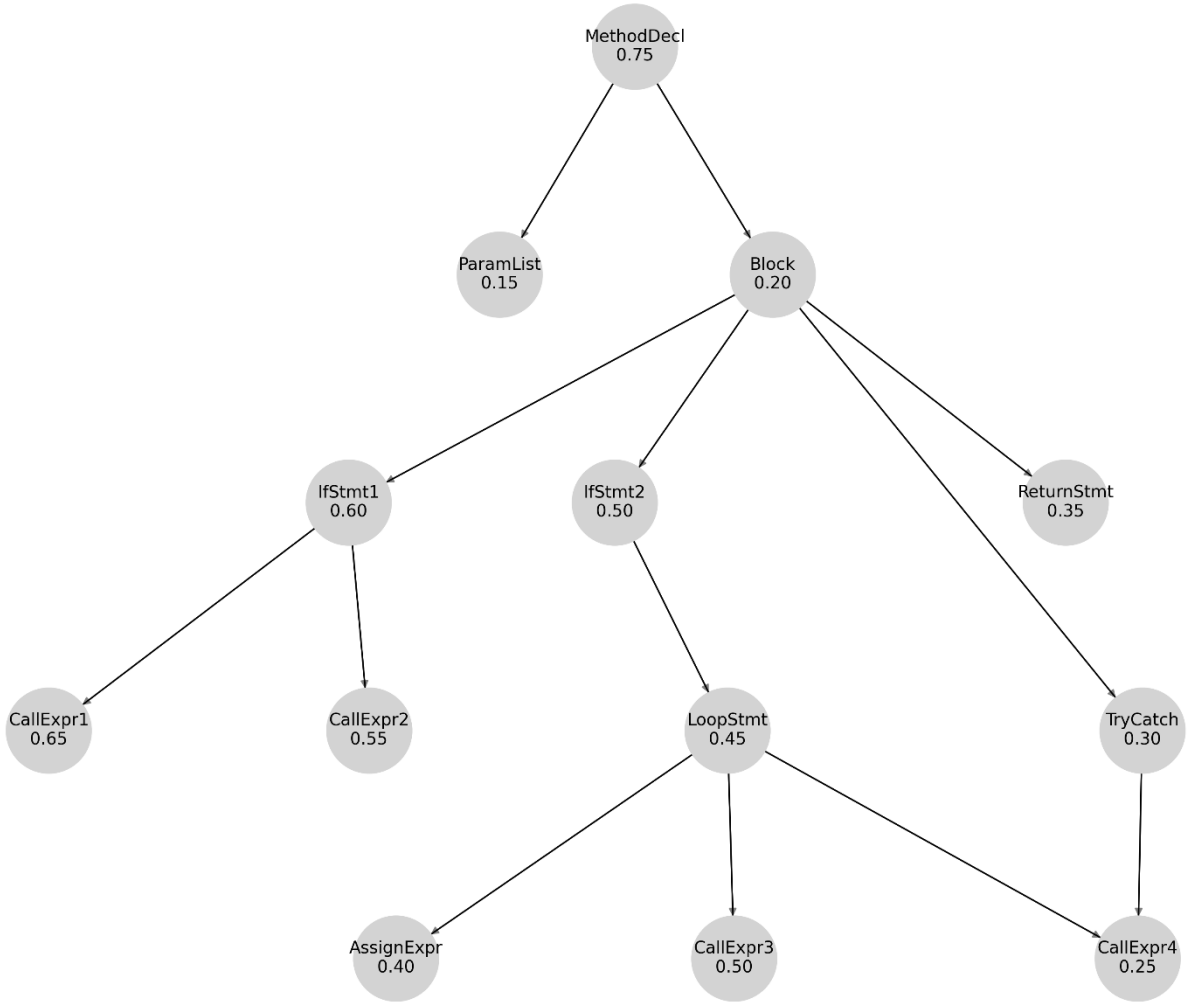

Рисунок 2.12 – Візуалізація CPG-фрагмента для правильно виявленого антипатерну «Long Method»

Аналіз типових помилок. Серед хибнопозитивних (false positives) випадків домінують компоненти з високими значеннями метрик (довгі класи, методи з великою кількістю параметрів), які, однак, були спеціально спроєктовані як «фасади» або «агрегатори» і не розглядаються експертами як антипатерни. Хибнонегативи (false negatives) часто пов’язані з тонкими семантичними залежностями, які важко виявити навіть експертам, наприклад, прихована логіка

доступу до глобального стану через кілька рівнів обгорток. Це підтверджує, що проблема виявлення антипатернів лишається частково суб’єктивною, а модель, по суті, наближається до певного «середнього» погляду на якість коду [23, 59].

Пояснення рішень моделі та порівняння з експертами. Для вибірки спірних випадків будувалися heatmap-пояснення на основі ієрархічної уваги та аналізу ваг каналів ознак. У багатьох прикладах експерти погоджувалися з тим, що компоненти з високою часткою еволюційних ознак (часті дефектні коміти, нестабільність) є «підозрілими», навіть якщо формально вони не потрапляють під класичні визначення антипатернів. Це свідчить, що модель здатна виявляти потенційно проблемні елементи, які виходять за рамки традиційних rule-based шаблонів.

Порівняння з підходами explainable AI у програмній інженерії показує, що поєднання графових уваг, каналових ваг та лаконічних «карток пояснень» наближається до сучасних рекомендацій щодо інтерпретованості моделей у SE-аналітиці [73, 115].

2.8 Висновки до другого розділу

У другому розділі дисертаційної роботи обґрунтовано архітектуру багаторівневої моделі виявлення антипатернів у програмних компонентах на основі гібридного графового подання коду. Запропонований підхід поєднує чотири типи сигналів – структурні, семантичні, метричні та еволюційні – у формі Code Property Graph (CPG), що дозволяє моделювати складну природу антипатернів у промислових програмних системах.

Сформульовано вимоги до моделі, зокрема: підтримка мультимодального подання ознак, багаторівневої структури («метод – компонент – проєкт»), режиму open-set або low-confidence для роботи з невизначеністю, міжмовної переносимості та інкрементальної обробки для сумісності з CI/CD.

Запропоновано механізми побудови локального, компонентного та проєктного рівнів моделі, що включають:

– гетерогенну GNN з урахуванням типів зв’язків у CPG;

– агрегацію локальних представлень методів і базових блоків за допомогою уваги;

– модель графу взаємодії компонентів із еволюційними атрибутами ребер;

– ієрархічну увагу для узгодження локальних, компонентних та проєктних контекстів.

Запропоновано класифікаційну підсистему з open-set головою, яка поєднує ймовірнісну оцінку антипатернів з енергетичними та ентропійними показниками невизначеності, що дозволяє здійснювати селективне передбачення і зменшити ризик помилкових рішень.

Розроблено інкрементальний конвеєр функціонування моделі для pull request-аналізу, який масштабований до великих промислових репозиторіїв, а також механізми пояснюваності рішень на основі ваг уваги та каналів ознак, що підвищують прийнятність моделі в DevOps-практиках.

Отриманий наступний пункт наукової новизни:

– *вперше* запропоновано багаторівневу модель виявлення антипатернів на основі гібридного графа, що уніфікує синтаксичні дерева, графи потоку керування і графи залежностей у граф властивостей коду, з додатковими семантичними та еволюційними ознаками. Запропонована модель дозволяє виявляти типові і нетипові антипатерни, працювати з багатомовними програмними проєктами і забезпечувати сумісність із процесами безперервної інтеграції та доставлення.

3 МОДЕЛІ ТА МЕТОДИ РЕКОМЕНДАЦІЇ, ПЛАНУВАННЯ ТА ПРОЦЕСНОГО ОЦІНЮВАННЯ РЕФАКТОРИНГІВ ДЛЯ УСУНЕННЯ АНТИПАТЕРНІВ

3.1 Концепція усунення антипатернів на основі результатів багаторівневої моделі виявлення

3.1.1 Роль детектора антипатернів у циклі «виявлення – усунення – моніторинг»

У попередньому розділі було розроблено багаторівневу модель виявлення антипатернів, яка на основі гібридного графового подання коду, семантичних вбудовувань і еволюційних ознак формує для кожного програмного компонента узгоджений діагностичний профіль. На відміну від класичних інструментів виявлення «запахів» коду, що повертають переважно бінарні або жорстко порогові попередження [121], багаторівнева модель надає для кожного компонента вектор ймовірностей належності до різних класів антипатернів, а також чисельні оцінки невизначеності.

На концептуальному рівні роботу детектора можна подати як відображення стану репозиторію $R(t)$ у момент часу t у множину діагностичних записів $W(t)$:

$$W(t) = D(R(t)), \tag{3.1}$$

де D – багаторівнева модель виявлення антипатернів.

Для компонента i вона повертає вектор

$$\mathbf{z}_i(t) = (p_i^{(1)}(t), \dots, p_i^{(K)}(t), u_i(t)), \tag{3.2}$$

де $p_i^{(k)}(t)$ – оцінка ймовірності наявності k-го антипатерну;

$u_i(t)$ – агрегований показник невизначеності (наприклад, ентропія чи енергетичний score).

Саме ці вектори $\mathbf{z}_i(t)$ надалі використовуються як вхідні дані для модулів рекомендації та планування рефакторингів.

У життєвому циклі управління якістю великої програмної системи детектор реалізує фазу «виявлення» в замкненому контурі «виявлення – усунення – моніторинг». В попередньому розділі була запропонована максимально надійна та багатовимірна «карта» антипатернів у кодовій базі, з урахуванням багаторівневої структури та еволюції проєкту. Натомість, поточний розділ зосереджений на етапах «усунення» та подальшого «моніторингу»: на основі множини $W(t)$ потрібно сформувати множину «кандидатів» на рефакторинг, скомпонувати їх у мінімально інвазивні послідовності та оцінити їхній вплив на внутрішню якість та процесні показники (CI/CD, дефектність тощо) [122, 123]. Таким чином, детектор відповідає на запитання «де і що саме є проблемою?», тоді як моделі, що розробляються в даному розділі, мають відповісти на запитання «що, де і коли доцільно змінювати, щоб безпечно зменшити технічний борг?».

3.1.2 Обмеження існуючої практики рефакторингу в індустріальних репозиторіях

Незважаючи на широке впровадження інструментів статичного аналізу та детекторів «запахів» коду, їх практичне використання в індустріальних репозиторіях зазвичай зводиться до роботи з плоскими списками попереджень, інтегрованими в IDE або CI/CD-конвеєри [124].

Такі списки, як правило, не враховують еволюційний контекст, міжкомпонентні залежності та неоднорідність впливу антипатернів на підтримуваність і дефектність, що призводить до перевантаження розробників сигналами у великих кодових базах [121, 124].

Першим суттєвим обмеженням є відсутність явного моделювання невизначеності.

Більшість інструментів трактує кожне попередження як достовірне, що за умов шумних даних і контекстної залежності рішень спричиняє накопичення хибних спрацювань і зниження довіри до засобів аналізу [126]. На практиці це призводить або до масового ігнорування попереджень, або до вимкнення частини правил.

Другим недоліком є слабкий зв’язок між рішеннями про рефакторинг і процесними показниками розробки. У типовій практиці антипатерни аналізуються ізольовано від стабільності збірок, частоти деплоїв, часу відновлення після відмов та інших DevOps-метрик, хоча сучасні підходи розглядають внутрішню якість і якість процесу як взаємопов’язані складові безперервної якості ПЗ [123]. Це ускладнює обґрунтування моменту та доцільності рефакторингу.

Третім обмеженням є відсутність аналізу причинно-наслідкових зв’язків між рефакторингами та змінами показників якості. У реальних проєктах покращення метрик після втручань часто неможливо однозначно атрибутувати саме рефакторингам через вплив супутніх факторів, таких як зміна навантаження, оновлення бібліотек або організаційні зміни. Це ускладнює накопичення знань про ефективність різних типів рефакторингу в конкретному проєкті.

Отже, наявна практика потребує переходу від плоских списків попереджень до інтегрованої моделі підтримки рефакторингу, яка враховує невизначеність рішень, поєднує сигнали внутрішньої якості з процесними метриками та забезпечує аналіз причинно-наслідкового впливу послідовностей рефакторингів.

3.1.3 Концептуальна трирівнева модель усунення антипатернів

Для подолання виявлених обмежень у дисертації запропоновано концептуальну трирівневу модель усунення антипатернів, інтегровану в цикл «виявлення – усунення – моніторинг» та побудовану на виходах багаторівневої моделі виявлення.

Рівень 1. Локальні рекомендації рефакторингів. На цьому рівні на основі діагностичних векторів $z_i(t)$ формується множина кандидатів на локальні

рефакторинги для конкретних антипатернів (зокрема дублікатів коду і класичних «запахів»). Для кожного кандидата оцінюються очікуваний виграш у внутрішній якості, зменшення дублювання, орієнтовні витрати та ризики.

Рівень 2. Композиція мінімально інвазивних послідовностей. Оскільки локальні рекомендації можуть бути взаємозалежними або конфліктними, на планувальному рівні виконується відбір і впорядкування підмножини рефакторингів, що задовольняє ресурсні та інтерфейсні обмеження й мінімізує сумарний вплив на архітектуру та процес розробки.

Рівень 3. Процесне SPC-оцінювання ефективності. Після виконання планів рефакторингу здійснюється моніторинг часових рядів метрик якості та процесу з використанням методів статистичного контролю процесів (SPC), що дозволяє виявляти стійкі зрушення та відокремлювати ефекти рефакторингу від випадкових флуктуацій.

Взаємодію багаторівневого детектора, рекомендаційного модуля, планувальника та SPC-модуля схематично подано на рисунку 3.1.

Виходячи зі схеми, представленої на рисунку, можна зробити висновок, що усунення антипатернів розглядається не як одноразова дія, а як складова безперервного управління якістю програмного забезпечення, узгоджена з підходами багаторівневого оцінювання якості у великих програмних системах [125].

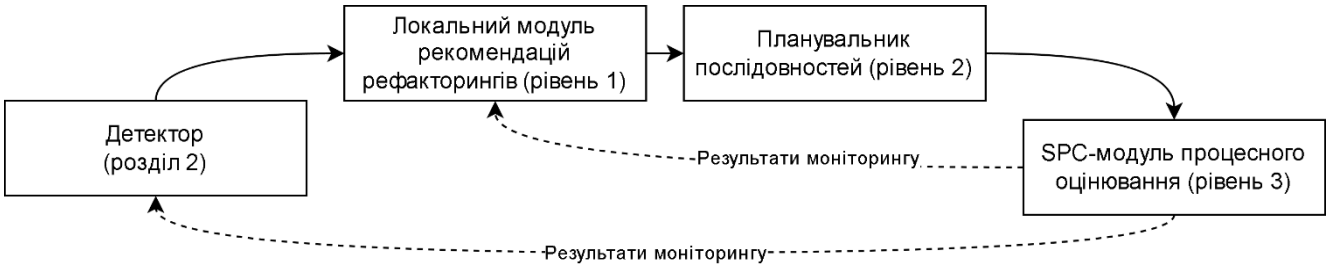

Рисунок 3.1 – Концептуальна трирівнева модель усунення антипатернів у циклі «виявлення – усунення – моніторинг»

3.1.4 Формулювання вимог до моделей рекомендації, планування та оцінювання

Із запропонованої концептуальної моделі безпосередньо випливають вимоги до моделей рекомендації рефакторингів, планування їх послідовностей та процесного оцінювання ефективності.

По-перше, усі моделі мають бути узгоджені з гібридним графовим поданням і багаторівневою структурою, сформованими в розділі 2. Це передбачає роботу з ознаками Code Property Graph, семантичними вбудовуваннями та еволюційними метриками, а також підтримку режиму open-set для коректної обробки невпевнених і атипових випадків.

По-друге, задачі рекомендації рефакторингів є багатоцільовими: одночасно враховуються зміни внутрішньої якості коду, щільності дублікатів, ризиків для стабільності CI/CD і трудомісткості виконання. Відповідно, необхідна явна постановка багатоцільової оптимізації з урахуванням невизначеності в критеріях прийняття рішень [121, 124].

По-третє, планування послідовностей рефакторингів має спиратися на причинно-обґрунтований аналіз, що дозволяє відрізняти статистичні кореляції від реального впливу втручань на метрики якості та уникати повторення неефективних або ризикових дій у межах конкретного проєкту.

По-четверте, модуль процесного оцінювання повинен забезпечувати статистично верифіковану інтерпретацію результатів рефакторингів, використовуючи контрольні карти, довірчі інтервали та формалізовані правила виявлення стійких зрушень у показниках якості та процесу [123, 126].

У сукупності ці вимоги окреслюють простір рішень, у межах якого в наступних підрозділах розробляються багатоцільова модель рекомендації рефакторингів, модель композиції мінімально інвазивних послідовностей та SPC-модель процесного оцінювання ефективності.

3.2 Багатоцільова модель рекомендації рефакторингів дублікатів коду з оцінкою невизначеності

3.2.1 Постановка задачі рекомендації рефакторингів для дублікатів коду

Дублювання коду є однією з ключових причин накопичення технічного боргу та зниження підтримуваності великих програмних систем [127, 128]. У межах роботи під групою дублікатів розуміється множина з щонайменше двох фрагментів коду, виявлених детектором дублікатів та таких, що мають істотну синтаксичну або семантичну подібність:

$$g = \{s_1, s_2, \dots, s_m\}, \qquad m \geq 2. \tag{3.3}$$

В даній дисертаційній роботі, у якості базової класифікації дублікатів виступає класифікація Type I–III [129, 130]: від текстуально ідентичних фрагментів до дублікатів з обмеженими структурними модифікаціями при збереженні поведінки. Для кожної групи дублікатів необхідно обрати дію з дискретного простору рефакторингів

$$\mathcal{A} = \{\text{Extract Method, Move Method, Pull Up, Inline, No refactoring}\}. \tag{3.4}$$

де перші чотири дії відповідають класичним рефакторингам, а варіант *No refactoring* формалізує усвідомлене рішення не виконувати зміни. Це важливо, оскільки не всі дублікати є шкідливими і в окремих випадках виконують роль стабільних шаблонів повторного використання [127, 129].

Модель рекомендації працює з вектором ознак x_g, що описує групу дублікатів (структурні, семантичні, еволюційні та контекстні характеристики), і формалізується як ймовірнісна залежність

$$p(a|\mathbf{x}_g), a \in \mathcal{A}, \tag{3.5}$$

На відміну від стандартної багатокласової класифікації, задача має дві принципові особливості. По-перше, модель повинна підтримувати open-set поведінку, тобто виявляти групи дублікатів, для яких жодна зі стандартних дій не має достатньої підтримки в даних. Для цього вводиться розширений простір дій

$$\mathcal{A}^{+} = \mathcal{A} \cup \{\text{Unknown}\}, \tag{3.6}$$

та правило відмови від рішення

$$\max_{a \in \mathcal{A}} p(a \mid \mathbf{x}_g) < \tau \Rightarrow a^{*}(g) = \text{Unknown}, \tag{3.7}$$

де τ – поріг впевненості, що задається з урахуванням допустимого рівня ризику [131, 133]

По-друге, рекомендації мають бути *confidence-aware*: окрім самих значень $p(a \mid x_g)$ оцінюється предиктивна невизначеність, зокрема через ентропію розподілу

$$H(p(\cdot \mid \mathbf{x}_g)) = -\sum_{a \in \mathcal{A}} p(a \mid \mathbf{x}_g) \log\, p(a \mid \mathbf{x}_g), \tag{3.8}$$

а також застосовуються методи калібрування та селективної класифікації [132, 133].

В результаті система не лише пропонує тип рефакторингу для групи дублікатів, але і надає оцінку впевненості, що є критично важливим для безпечної інтеграції рекомендацій у процес безперервної розробки (рис. 3.2).

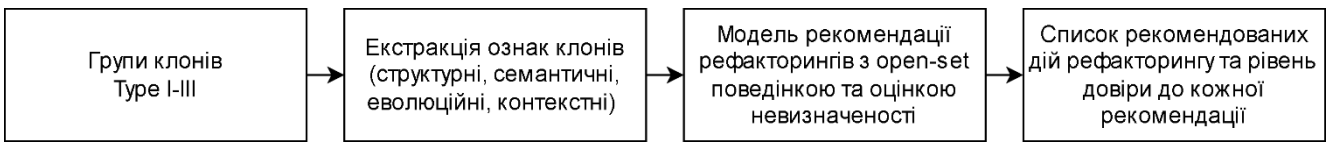

Рисунок 3.2 – Постановка задачі рекомендації рефакторингів для груп дублікатів

3.2.2 Ознаки для моделі рекомендації на основі гібридного графового подання

Для задачі рекомендації рефакторингів груп дублікатів коду використовується гібридне графове поданняб коду у вигляді Code Property Graph (CPG), яке поєднує AST, CFG та PDG і доповнюється метриками якості та еволюційними атрибутами [134, 135]. Формально програмний компонент описується орієнтованим графом

$$G = (V, E, \tau, \varphi), \tag{3.9}$$

де V – множина вузлів (оператори, вирази, змінні, методи);

E – множина типізованих ребер (syntactic, control, data, call);

$\tau(v)$ – тип вузла;

$\varphi(v)$ – вектор його локальних атрибутів (наприклад, лексема, роль, числові метрики).

Для кожного фрагмента дублікату $s_i \in g$ формується чотири групи ознак:

– структурні $x_{s_i}^{(str)}$ – характеристики локальної підструктури CPG (ступені вузлів, частоти типів, наявність типових підграфів, зв’язність у CFG/PDG) [134, 136];

– семантичні $x_{s_i}^{(sem)}$ – векторні подання коду з моделей типу code2vec, CodeBERT або GraphCodeBERT, що враховують синтаксичні та датафлоу-залежності [136, 137];

– метричні $x_{s_i}^{(met)}$ – класичні показники складності та розміру (LOC, цикломатична складність, глибина вкладеності, fan-in/fan-out тощо);

– еволюційні $x_{s_i}^{(evol)}$ – агреговані характеристики історії змін (churn, вік, кількість авторів, ко-зміни з дефектними файлами) [127].

Сумарний вектор ознак фрагмента визначається як

$$\mathbf{z}_{s_i} = [\mathbf{x}_{s_i}^{(str)} \| \mathbf{x}_{s_i}^{(sem)} \| \mathbf{x}_{s_i}^{(met)} \| \mathbf{x}_{s_i}^{(evol)}] \in \mathbb{R}^d. \quad (3.10)$$

Для переходу від окремих фрагментів до групи дублікатів $g = \{s_1, \dots, s_m\}$ застосовується операція агрегації (усереднення, медіана, перцентилі або pooling):

$$\mathbf{x}_g = \mathrm{Agg}(\{\mathbf{z}_{s_1}, \dots, \mathbf{z}_{s_m}\}). \quad (3.11)$$

У випадку використання GNN агрегація реалізується як диференційовний пулінг над підграфом CPG, індукованим вузлами дублікатів та їхнім локальним контекстом [138].

Додатково враховується контекст компонентів, у яких розміщені дублікати коду. Нехай $C(g)$ – множина відповідних компонентів. Для кожного $c \in C(g)$ обчислюється вектор ознак y_c, після чого формується контекстний вектор

$$\mathbf{x}_g^{(ctx)} = \mathrm{Agg}(\{\mathbf{y}_c | c \in C(g)\}). \quad (3.12)$$

Фінальний вектор ознак для моделі рекомендації має вигляд

$$\mathbf{x}_g^{(tot)} = [\mathbf{x}_g \| \mathbf{x}_g^{(ctx)}]. \quad (3.13)$$

Таким чином, гібридне графове подання забезпечує одночасний облік локальної структури та семантики дублікатів коду, їхнього метричного й еволюційного профілю, а також глобального контексту компонентів, що створює багатовимірну основу для навчання моделі рекомендації рефакторингів з урахуванням ризику, зусиль і очікуваної користі.

3.2.3 Моделі оцінювання очікуваної вигоди та трудомісткості рефакторингу

У рекомендаційній підсистемі, що працює з групами дублікатів, кожна кандидатна операція рефакторингу розглядається не лише з точки зору

«семантичної доцільності», а й крізь призму очікуваних витрат і вигоди. Попередні емпіричні дослідження показують, що не всі рефакторинги приводять до покращення якості: частина з них погіршує окремі показники, збільшує складність підтримки або не окупає витрати на внесення змін [139]. Це обґрунтовує потребу в явних моделях прогнозу зусиль та ефекту рефакторингу, інтегрованих у процес формування рекомендацій.

Модель прогнозу зусиль рефакторингу. Для кожної групи дублікатів g та кандидатної дії рефакторингу a (наприклад, «Extract Method», «Pull Up Method», «Replace Conditional with Polymorphism» тощо) вводиться вектор опису зусиль

$$\mathbf{e}(g,a) = (e_{\text{loc}}(g,a), e_{\text{compl}}(g,a), e_{\text{comp}}(g,a))^{\top}, \tag{3.14}$$

де $e_{\text{loc}}(g,a)$ – очікувана зміна кількості рядків коду (LOC);

$e_{\text{compl}}(g,a)$ – показник складності змін (оцінений за історичними метриками churn, кількістю елементарних операцій редагування тощо);

$e_{\text{comp}}(g,a)$ – очікувана кількість компонентів/файлів, що будуть змінені.

Прогноз кожної компоненти вектора зусиль ґрунтується на ознаках, сформованих у підрозділі 3.2.2, тобто на структурних, семантичних, метричних та еволюційних характеристиках гібридного графового подання коду (CPG) для групи дублікатів коду та пов’язаних компонентів. Для LOC-змін використовується регресійна модель

$$e_{\text{loc}}(g,a) = \mathbb{E}\big[\big|\Delta\text{LOC}_{g,a}\big|\big|\mathbf{x}_g, \mathbf{z}_a\big|\,\big], \tag{3.15}$$

де $\mathbf{x}_g$ – вектор ознак CPG для групи дублікатів g;

$\mathbf{z}_a$ – кодування типу рефакторингу (one-hot або ембеддинг).

Аналогічно визначаються

$$e_{\text{compl}}(g,a) = \mathbb{E}\big[\Delta C_{\text{edit}} \mid \mathbf{x}_g, \mathbf{z}_a\big], \qquad e_{\text{comp}}(g,a) = \mathbb{E}\big[\Delta N_{\text{files}} \mid \mathbf{x}_g, \mathbf{z}_a\big], \tag{3.16}$$

де ΔC_{edit} – характеризує складність змін (кількість елементарних операцій у дереві змін);

ΔN_{files} – приріст числа модулів, що були зачеплені операцією.

У навчанні моделі зусиль як цільові значення використовуються фактично спостережені характеристики попередніх рефакторингів, відновлені з історії змін у системі керування версіями (VCS): величини $|\Delta LOC|$, кількість модифікованих файлів, складність патча тощо. Такий підхід дозволяє адаптувати оцінку трудомісткості до конкретного проєкту, стилю розробки та практик команди.

Модель прогнозу вигоди рефакторингу. Очікувана вигода від рефакторингу описується вектором

$$\mathbf{b}(g,a) = (B_{\text{maint}}(g,a), B_{\text{clone}}(g,a), B_{\text{def}}(g,a))^{\top}, \tag{3.17}$$

де $B_{\text{maint}}(g,a)$ – очікувана зміна інтегральної метрики підтримуваності;

$B_{\text{clone}}(g,a)$ – зменшення щільності дублікатів;

$B_{\text{def}}(g,a)$ – очікуване зменшення дефектності у відповідних компонентах.

Нехай $\mathbf{m}(g) \in \mathbb{R}^K$ – вектор вихідних метрик підтримуваності для групи дублікатів (наприклад, поєднання індексу підтримуваності, цикломатичної складності, метрик зв'язаності та зв'язності), а $\mathbf{m}'(g,a)$ – прогнозовані метрики після застосування дії a. Тоді модель вигоди за підтримуваністю визначається як

$$\begin{gathered}\Delta\mathbf{m}(g,a) = \mathbb{E}\left[\mathbf{m}'(g,a) - \mathbf{m}(g) \mid \mathbf{x}_g, \mathbf{z}_a\right], \\ B_{\text{maint}}(g,a) = \mathbf{w}^{\top}\Delta\mathbf{m}(g,a),\end{gathered} \tag{3.18}$$

де $\mathbf{w}$– вектор ваг, що задає важливість окремих метрик для цільової системи.

У такий спосіб $B_{\text{maint}}(g,a) > 0$ інтерпретується як очікуване покращення підтримуваності.

На рисунку 3.3 наведена концептуальна схема моделей прогнозу та трудомісткості та вигоди рефакторингу.

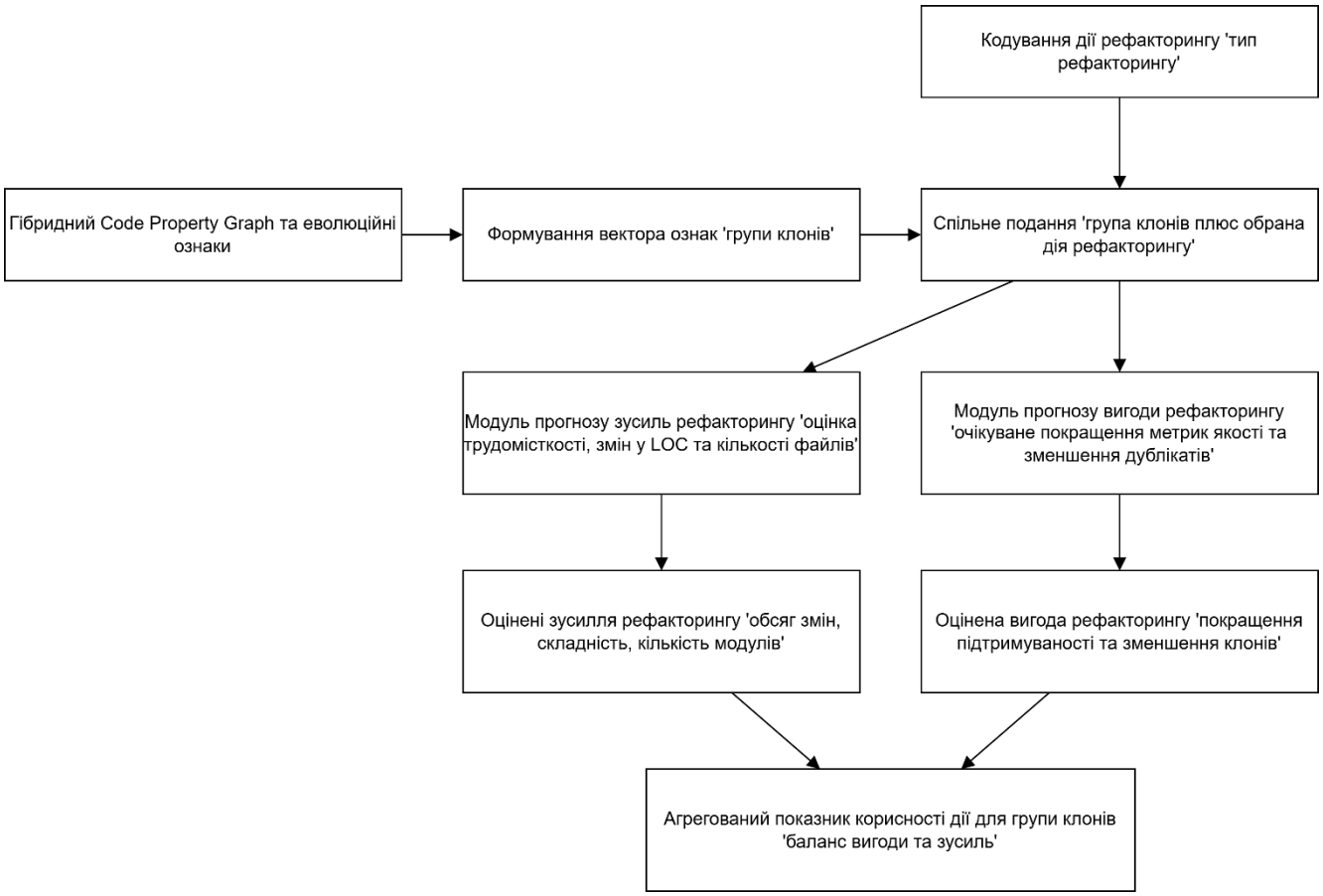

Рисунок 3.3 – Концептуальна схема моделей прогнозу зусиль та вигоди рефакторингу на основі ознак CPG

Для компонент, що містять дубльовані фрагменти, вводиться локальна щільність дублікатів $d_{\text{clone}}(g)$ (частка дубльованого коду у відповідному модулі). Модель прогнозує очікуване зменшення цього показника:

$$B_{\text{clone}}(g,a) = d_{\text{clone}}(g) - \mathbb{E}\left[d'_{\text{clone}}(g,a) \mid \mathbf{x}_g, \mathbf{z}_a\right], \tag{3.19}$$

де $d'_{\text{clone}}(g,a)$ – прогнозована щільність клонів після рефакторингу.

Вплив на дефектність моделюється через зміну ймовірності наявності дефектів у компоненті, пов’язаній з групою дублікатів. Нехай $p_{\text{def}}(g)$ – поточна оцінка ймовірності дефекту (наприклад, за моделлю JIT-defect-prediction), а $p'_{\text{def}}(g,a)$ – прогнозована ймовірність після рефакторингу. Тоді

$$B_{\text{def}}(g,a) = p_{\text{def}}(g) - \mathbb{E}\left[p'_{\text{def}}(g,a) \mid \mathbf{x}_g, \mathbf{z}_a\right]. \tag{3.20}$$

Позитивні значення $B_{\text{def}}(g,a)$ означають очікуване зменшення ризику дефектів у результаті усунення дублікатів або інших антипатернів.

Усі компоненти векторів $\mathbf{e}(g,a)$ та $\mathbf{b}(g,a)$ приводяться до безрозмірного масштабу, наприклад, шляхом min-max нормалізації в межах проєкту, що дозволяє поєднувати їх в єдиній функції корисності. Узагальнений показник очікуваної доцільності застосування дії a до групи дублікатів g задається як

$$U(g,a) = \alpha\, B_{\text{maint}}(g,a) + \beta\, B_{\text{clone}}(g,a) + \gamma\, B_{\text{def}}(g,a) - \lambda\|\mathbf{e}(g,a)\|_1, \quad (3.21)$$

де $\alpha,\ \beta,\ \gamma,\ \lambda \geq 0$ – параметри, що відображають пріоритети проєкту між покращенням підтримуваності, зменшенням клонів, зниженням дефектності та припустимою трудомісткістю змін.

Узагальнена структура вимірюваних величин, що використовуються в моделях, наведена в таблиці 3.1.

Таблиця 3.1 – Основні прогнозовані показники зусиль і вигоди рефакторингу

Параметр	Позначення	Тип	Нормалізоване значення	Інтерпретація
Зміна LOC	e_{loc}^{norm}	зусилля	0,38	Помірний обсяг змін коду
Складність патча	e_{compl}^{norm}	зусилля	0,44	Середня складність редагування
Кількість змінених файлів	e_{comp}^{norm}	зусилля	0,46	Рефакторинг зачіпає кілька компонентів
L1-норма зусиль	$\|\mathbf{e}(g,a)\|_1^{norm}$	агрегат	0,42	Узагальнена трудомісткість операції
Приріст підтримуваності	B_{maint}^{norm}	вигода	0,71	Відчутне покращення внутрішньої якості
Зменшення щільності клонів	B_{clone}^{norm}	вигода	0,76	Значне усунення дублювання
Зниження дефектності	B_{def}^{norm}	вигода	0,64	Суттєве зменшення ризику дефектів
Сумарна вигода	B_{total}^{norm}	агрегат	0,70	Загальний позитивний ефект рефакторингу
Корисність дії	$U(g,a)$	результат	0,19	Рефакторинг доцільний за заданими вагами
Рішення планувальника	–	висновок	рекоменд.	Дія включається до плану рефакторингів

Таким чином, запропоновані моделі дозволяють для кожної групи дублікатів узгоджено оцінити очікувані витрати (через $\mathbf{e}(g,a)$) та вигоду (через $\mathbf{b}(g,a)$) для набору альтернативних дій рефакторингу. Це створює основу для подальшого ранжування рекомендацій за корисністю та інтеграції критеріїв технічного боргу й обмежень ресурсів у процес напівавтоматизованого рефакторингу. В таблиці 3.2 проілюстрований приклад оцінки для рефакторингу «Extract Method».

Таблиця 3.2 – Приклад оцінювання зусиль та вигоди для групи клонів при застосуванні рефакторингу «Extract Method»

Параметр	Позначення	Тип	Значення (кейс)	Інтерпретація
Тип дублікатів	–	характеристика	Type II	Дублікати з різними ідентифікаторами та літералами
Кількість фрагментів	m	характеристика	4	Кількість дубльованих фрагментів у групі
Рефакторинг	a	дія	Extract Method	Виділення спільної логіки в окремий метод
Зміна LOC	e_{loc}	зусилля	92	Очікуване сумарне скорочення/перерозподіл рядків коду
Складність патча	e_{compl}	зусилля	41	Кількість елементарних edit-операцій у VCS
Змінені файли	e_{comp}	зусилля	3	Кількість класів, яких торкається рефакторинг
Зміна підтримуваності	B_{maint}	вигода	+6.4	Приріст інтегрального індексу підтримуваності
Зменшення щільності клонів	B_{clone}	вигода	4.8%	Частка дубльованого коду, усунута в модулі
Зниження дефектності	B_{def}	вигода	0.037	Зменшення ймовірності дефектів у компоненті
Нормалізовані зусилля	$\|\mathbf{e}(g,a)\|_1$	агрегат	0.42	L1-норма нормалізованого вектора зусиль
Агрегована вигода	B_{total}	агрегат	0.61	Зважена сума компонентів вигоди
Корисність дії	$U(g,a)$	результат	0.19	Підсумкова корисність для ранжування рекомендацій
Рішення системи	–	висновок	рекомендовано	Рефакторинг доцільний за заданими порогами

3.2.4 Пропонована багатоцільова модель рекомендації рефакторингів з оцінкою невизначеності

У попередніх підрозділах було введено моделі прогнозу очікуваної вигоди $\mathbf{b}(g,a)$ та трудомісткості $\mathbf{e}(g,a)$ для груп дублікатів g (дії a), які є кандидатами на рефакторинг. У промисловому контексті цього недостатньо: необхідно також враховувати ризик порушення безперервної інтеграції (CI-risk), тобто ймовірність того, що застосування рефакторингу призведе до падіння збірки, невдалого проходження автоматичних тестів або до значного збільшення часу пайплайна. Таким чином, задача рекомендації рефакторингів природно формулюється як багатоцільова задача з принаймні трьома цілями:

1) максимізація очікуваної вигоди;

2) мінімізація зусиль;

3) мінімізація CI-ризику.

Багатовимірне представлення вигода/трудомісткість/CI-риск. Для кожної пари (g,a) вводиться вектор цілей

$$\mathbf{y}(g,a) = (b(g,a), E(g,a), R_{\mathrm{CI}}(g,a))^{\mathsf{T}}, \tag{3.22}$$

де $b(g,a)$ – агрегований показник очікуваної вигоди (побудований на основі компонент $\mathbf{b}(g,a)$, визначених у підрозділі 3.2.3);

$E(g,a)$ – агрегована оцінка зусиль (на основі $\mathbf{e}(g,a)$);

$R_{\mathrm{CI}}(g,a)$ – ймовірність або очікувана «вартість» ризику порушення CI-процесу.

Модель CI-ризику навчається за історією змін у системі керування версіями разом з логами CI/CD: кожне спостереження містить ознаки зміни (код, метрики, еволюційні характеристики) та бінарний результат build_failed $\in \{0,1\}$ чи більш детальний індикатор проблем (хибно-позитивні спрацювання тестів, збільшення часу збірки тощо). У найпростішому випадку

$$R_{\mathrm{CI}}(g,a) = \mathbb{P}\big(\text{build\_failed} = 1 \mid \mathbf{x}_g, \mathbf{z}_a\big), \tag{3.23}$$

де $\mathbf{x}_g$ – вектор ознак з гібридного CPG для групи g;

$\mathbf{z}_a$ – кодування типу рефакторингу.

Далі будується багатоцільова модель, яка для кожної пари (g,a) повертає прогностичний розподіл

$$p\big(\mathbf{y}|\mathbf{x}_g, \mathbf{z}_a\big), \tag{3.24}$$

а не лише точкову оцінку. Це дає змогу враховувати як очікувані значення, так і невизначеність для вигоди, трудомісткості та CI-рису. Для практичної інтеграції у пайплайн CI/CD використовується як векторне представлення $\mathbf{y}(g,a)$, так і скаляризована функція корисності

$$S(g,a) = \lambda_b\, \mathbb{E}[b(g,a)] - \lambda_E\, \mathbb{E}[E(g,a)] - \lambda_R\, \mathbb{E}[R_{\mathrm{CI}}(g,a)], \tag{3.25}$$

де $\lambda_b,\ \lambda_E,\ \lambda_R \geq 0$ – вагові коефіцієнти, що відображають пріоритети команди між вигодою, зусиллями і ризиком порушення CI.

На рисунку 3.4 наведено геометричне представлення багатовимірного простору «Вигода – Трудомісткість – CI-риск».

Ймовірнісне моделювання та оцінка невизначеності. Для оцінки $p\big(\mathbf{y}|\mathbf{x}_g, \mathbf{z}_a\big)$ використовується ансамбль моделей $f^{(1)}, \dots, f^{(M)}$, натренованих на однакових даних, але з різними початковими вагами та/або підвибірками [140, 141]. Для кожної пари (g,a) отримуємо ансамблеві прогнози $\mathbf{y}^{(m)}(g,a)$ та апроксимуємо

$$\begin{aligned} \widehat{\boldsymbol{\mu}}(g,a) &= \frac{1}{M}\sum_{m=1}^{M} \mathbf{y}^{(m)}\,(g,a), \widehat{\Sigma}(g,a) \\ &= \frac{1}{M}\sum_{m=1}^{M} (\mathbf{y}^{(m)}(g,a) - \widehat{\boldsymbol{\mu}}(g,a))(\mathbf{y}^{(m)}(g,a) - \widehat{\boldsymbol{\mu}}(g,a))^{\top}, \end{aligned} \tag{3.26}$$

де $\hat{\boldsymbol{\mu}}(g, a)$ – оцінка маточікування вектора цілей,

$\hat{\Sigma}(g, a)$ – його коваріаційна матриця.

Діагональні елементи $\hat{\Sigma}(g, a)$ інтерпретуються як оцінки дисперії (невизначеності) по кожній з цілей; великі значення сигналізують про ненадійність відповідних прогнозів.

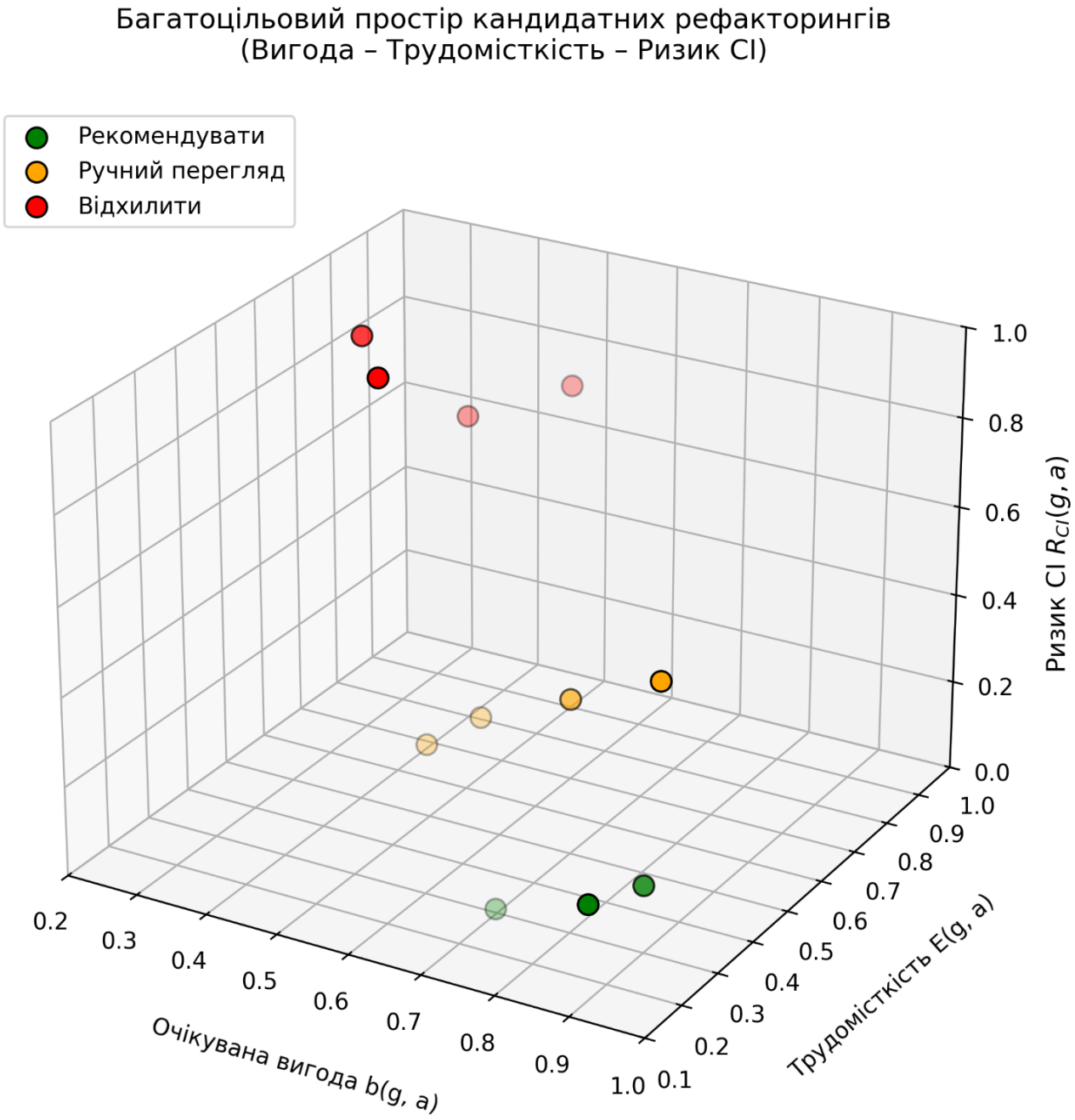

Рисунок 3.4 – Багатоцільове представлення кандидатів у рефакторинг у просторі «Вигода – Трудомісткість – CI-риск»

Для прийняття рішень система додатково формулює бінарну задачу: чи варто рекомендувати дію a для групи g з огляду на багатовимірну корисність. Позначимо $y \in \{0,1\}$, де $y = 1$ означає «рефакторинг рекомендується». Ансамбль повертає емпіричний розподіл

$$\hat{p} = \left(\mathbf{y} = 1 | \mathbf{x}_g, \mathbf{z}_a\right) = \frac{1}{M} \sum_{m=1}^{M} \mathbb{I}\left(f^{(m)}\left(\mathbf{x}_g, \mathbf{z}_a\right) \text{ рекомендує } a\right). \quad (3.27)$$

На його основі обчислюються стандартні показники невизначеності [142]:

– показник впевненості:

$$\text{conf}(g, a) = \max_{k \in \{0,1\}} \hat{p}(y = k | \mathbf{x}_g, \mathbf{z}_a), \quad (3.28)$$

– предиктивна ентропія

$$H(g, a) = -\sum_{k \in \{0,1\}} \hat{p}\left(y = k | \mathbf{x}_g, \mathbf{z}_a\right) \log \hat{p}\left(y = k | \mathbf{x}_g, \mathbf{z}_a\right). \quad (3.29)$$

Низька ентропія $H(g, a)$ і високий показник впевненості $\text{conf}(g, a)$ свідчать про «впевнене» рішення, тоді як висока ентропія і/або низький показник впевненості – про високу невизначеність, наприклад при недостатньо представлених у тренувальних даних типах коду чи рефакторингів [143].

Оскільки нейромережеві моделі часто є погано каліброваними (тобто їхні ймовірності не відповідають реальним частотам подій), у запропонованій моделі використовується післятренувальне калібрування, зокрема температурне масштабування або ізотонна регресія [144]. Калібровані значення $\hat{p}(y = k | \cdot)$ є передумовою коректного використання ентропії та порогів конфіденції.

Open-set/low-confidence режим та вибір відмови від рекомендацій. У реальних умовах система неминуче стикається з «нетиповими» випадками: новими мовними конструкціями, рідкісними архітектурними патернами, нестандартними

практиками розробки. Відповідно до парадигми open-set recognition [141, 145], для таких зразків доцільно не намагатися формувати сильну рекомендацію, а явним чином «утриматися від відповіді».

У запропонованій моделі open-set/low-confidence режим реалізується через пороги для значень впевненості та ентропії:

$$\text{рекомендувати } (g,a) = \begin{cases} 1, \text{якщо } S(g,a) \geq \tau_S \wedge conf(g,a) \geq \tau_C \wedge H(g,a) \leq \tau_H \\ 0, \text{ інакше} \end{cases}, \quad (3.30)$$

де τ_S – мінімальне значення скорингової функції;

τ_c – мінімальна конфіденція;

τ_H – максимальне припустиме значення ентропії.

Кандидати, для яких

$$\text{conf}(g,a) < \tau_c \text{ або } H(g,a) > \tau_H, \quad (3.31)$$

позначаються як «low-confidence» і не потрапляють у автоматичний список рекомендованих рефакторингів. Для них система або пропонує ручний перегляд, або подає інформацію до окремого модуля активного навчання для подальшого розширення тренувальної вибірки.

Таким чином, багатоцільова модель рекомендації рефакторингів не лише узгоджує вигоду, трудомісткість та CI-ризик у єдиному просторі цілей, але й явно моделює невизначеність прогнозів. Це дозволяє уникнути агресивних рекомендацій у незнайомих регіонах простору коду та підтримувати безпечний режим роботи в умовах безперервної інтеграції.

3.2.5 Метод багатоцільового ранжування та відбору «безпечних» рекомендацій

У попередніх підрозділах для кожного кандидата на рефакторинг r були отримані нормовані оцінки очікуваної вигоди B_r, трудомісткості E_r та ризику

порушення CI/CD-потоку або стабільності релізів R_r. Доповнюючи їх оцінкою невизначеності/впевненості моделі U_r (або, навпаки, $C_r = 1 - U_r$), одержуємо багатовимірний простір цілей, у якому вибір конкретних дій доцільно формулювати як багатоцільову задачу оптимізації з відмовою від «небезпечних» рекомендацій [146].

Побудова Парето-фронту. Нехай для кожного кандидата r визначено вектор цілей у канонічній постановці «мінімізації»:

$$\mathbf{f}_r = (f_{r,1}, f_{r,2}, f_{r,3}, f_{r,4}) = (-B_r,\ E_r,\ R_r,\ U_r), \tag{3.32}$$

де $-B_r$– «вартість» у вимірі користі (максимізація B_r еквівалентна мінімізації $-B_r$);

E_r – очікувані зусилля;

R_r – оцінка ризику негативного впливу на потік змін;

U_r – міра невизначеності (наприклад, калібрована ентропія або дисперсія ансамблю).

Кандидат i *домінує* над кандидатом j, якщо він не гірший за всіма цілями та кращий принаймні за однією:

$$\mathbf{f}_i \prec \mathbf{f}_j \Leftrightarrow \forall k\colon f_{i,k} \le f_{j,k} \wedge \exists k\colon f_{i,k} < f_{j,k}. \tag{3.33}$$

Тоді Парето-фронт $\mathcal{P}$ визначається як множина всіх недомінованих кандидатів [146, 147]:

$$\mathcal{P} = \left\{ r \middle| \nexists_s \colon \mathbf{f}_S \prec \mathbf{f}_{j,k} \right\}. \tag{3.34}$$

З обчислювальної точки зору використовується ітеративне виділення «шарів» недомінованих рішень (Pareto layers), або ж відразу застосовується еволюційний алгоритм багатоцільової оптимізації (наприклад, NSGA-II) для пошуку апроксимації Парето-фронту [147].

Як показано на рисунку 3.5, Парето-фронт визначає підмножину рефакторингів, які є оптимальними з точки зору компромісу між користю та зусиллями (з урахуванням ризику та невизначеності), тоді як усі інші кандидати завідомо гірші за певною комбінацією цілей.

Штрафи за низьку впевненість та правило абстиненції. Навіть серед Парето-оптимальних рішень частина кандидатів може бути отримана в зонах високої невизначеності моделі (наприклад, на межі області навчальних даних або у випадку потенційного open-set-сценарію). Щоб явно врахувати цей аспект, вводиться штраф за невизначеність. Для простоти покладемо, що $U_r \in [0;1]$, де 0 відповідає повній впевненості, а 1 – максимальної невизначеності. Тоді можна визначити скориговану оцінку користі

$$B'_r = B_r - \alpha U_r, \tag{3.35}$$

де $\alpha \geq 0$ – коефіцієнт штрафу.

Таким чином, навіть для «вигідного» за користю рефакторингу низька впевненість (велике U_r) зменшує його пріоритет.

Ймовірнісне моделювання та оцінка невизначеності. Для оцінки $p(\mathbf{y}|\mathbf{x}_g, \mathbf{z}_a)$ використовується ансамбль моделей $f^{(1)}, \dots, f^{(M)}$, натренованих на однакових даних, але з різними початковими вагами та/або підвибірками [140, 141]. Для кожної пари (g, a) отримуємо ансамблеві прогнози $\mathbf{y}^{(m)}(g,a)$ та апроксимуємо

$$\begin{aligned} \widehat{\boldsymbol{\mu}}(g,a) &= \frac{1}{M}\sum_{m=1}^{M} \mathbf{y}^{(m)}(g,a), \hat{\Sigma}(g,a) \\ &= \frac{1}{M}\sum_{m=1}^{M} (\mathbf{y}^{(m)}(g,a) - \widehat{\boldsymbol{\mu}}(g,a))(\mathbf{y}^{(m)}(g,a) - \widehat{\boldsymbol{\mu}}(g,a))^{\top}, \end{aligned} \tag{3.36}$$

де $\widehat{\boldsymbol{\mu}}(g,a)$ – оцінка маточікування вектора цілей,

$\hat{\Sigma}(g,a)$ – його коваріаційна матриця.

Діагональні елементи $\hat{\Sigma}(g, a)$ інтерпретуються як оцінки дисперії (невизначеності) по кожній з цілей; великі значення сигналізують про ненадійність відповідних прогнозів.

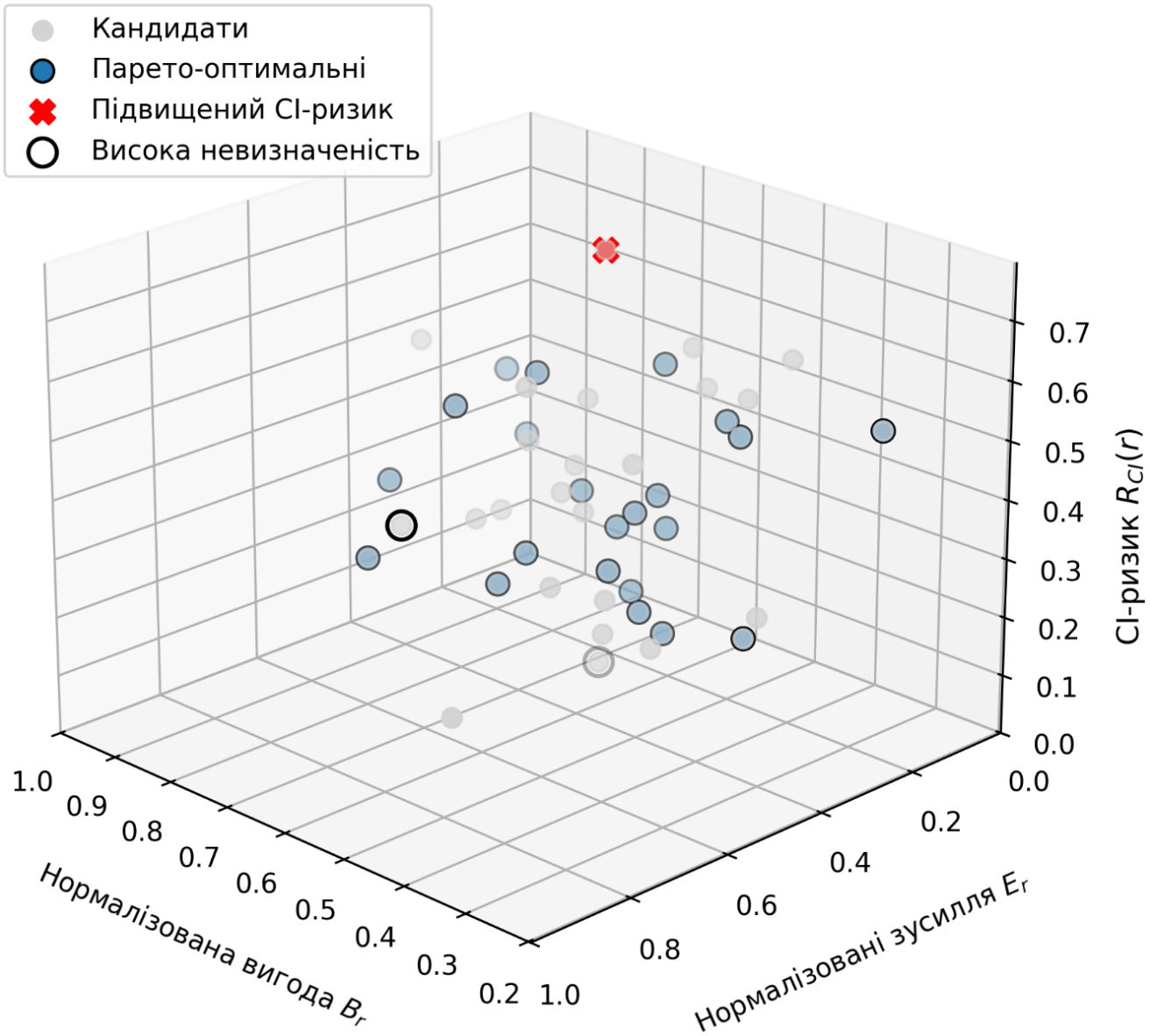

Рисунок 3.5 – Парето-представлення кандидатів на рефакторинг у просторі показників вигоди, трудомісткості та CI-ризику

У рамках *селективної класифікації* (selective prediction) [148] вводиться правило абстиненції: система має право не видавати рекомендацію, якщо жоден кандидат не задовольняє заданим обмеженням за невизначеністю та ризиком. Формально множина «безпечних» кандидатів визначається як

$$S = \{r \in P | U_r \leq U_{max}, R_r \leq R_{max}\}, \quad (3.37)$$

де U_{max} та R_{max} – порогові значення, узгоджені з політиками якості та DevOps-практикою конкретного проєкту.

Якщо $S = \emptyset$, система утримується від відповіді: вона сигналізує про виявлені антипатерни, але не формує конкретних рекомендацій щодо рефакторингу, перекладаючи рішення на експертів.

Формування пріоритизованого списку рефакторингів. На останньому кроці необхідно перетворити множину «безпечних» рішень S на впорядкований список, придатний для практичного планування. Для цього застосовується агрегована функція ранжування

$$score_r = \omega_B B_r - \omega_E E_r - \omega_R R_r - \omega_U U_r, r \in S, \quad (3.38)$$

де $w_B, w_E, w_R, w_U \geq 0$ – вагові коефіцієнти, які задають відносну важливість користі, зусиль, ризику та невизначеності.

Сортування кандидатів за спаданням score_r визначає глобальний порядок виконання рефакторингів; при цьому відношення Парето-домінування не порушується, а невизначеність явно враховується у вигляді штрафу. Як проілюстровано в таблиці 3.3, розробник або планувальник отримує відсортований список, у якому кожен елемент супроводжується прозорими числовими оцінками та статусом «рекомендовано» чи «відхилено» згідно з правилом утримання.

Таблиця 3.3 – Фрагмент сформованого списку кандидатів на рефакторинг

ID	Оцінка користі B_r	Оцінка зусиль E_r	Ризик R_r	Невизначеність U_r	Інтегральний бал $score_r$	Статус
R-014	0,82	0,30	0,18	0,12	0,41	рекомендовано
R-027	0,75	0,45	0,22	0,15	0,28	рекомендовано
R-005	0,91	0,62	0,35	0,48	0,07	відхилено (абстиненція за U_r)
R-031	0,68	0,25	0,41	0,10	0,22	відхилено (абстиненція за R_r)
R-019	0,59	0,20	0,14	0,08	0,27	рекомендовано
R-042	0,87	0,55	0,29	0,52	0,03	відхилено (абстиненція за U_r)

Це спрощує інтеграцію моделі у процеси пріоритизації технічного боргу, дає змогу порівнювати альтернативні плани рефакторингу та, за потреби, коригувати ваги w_B, w_E, w_R, w_U відповідно до поточних цілей проєкту (фокус на стабільності релізів, агресивне зниження технічного боргу тощо).

3.2.6 Експериментальне дослідження моделі рекомендації рефакторингів

Метою експериментального дослідження є оцінювання запропонованої моделі рекомендації рефакторингів для клонів з погляду:

– якості класифікації типів рефакторингу;

– коректності open-set-поведінки та механізму абстиненції;

– ефективності пріоритизації кандидатів;

– внеску окремих груп ознак (структурних, семантичних та еволюційних) у кінцеві метрики.

Набори даних і протокол розмітки рефакторингових рішень для клонів. Для оцінювання моделі було сформовано корпус клонів на основі декількох великих JVM-проєктів (Java/Kotlin), відібраних за критеріями відкритості репозиторію, активної історії змін та наявності достатньої кількості pull request-ів.

На вихідному коді застосовувався комбінований підхід до виявлення клонів (text/token-based і структурний), після чого клон-пари агрегувалися у *клон-класи* $G = \{g_1, \dots, g_M\}$, де кожний клас g_j містить щонайменше два фрагменти коду, які вважаються взаємними клонами.

Для кожного класу дублікатів g_j експерти-розробники фіксували реальне або бажане рефакторингове рішення y_j з множини

$$\mathcal{Y} = \{\text{Extract, PullUp, Parameterize, Inline, NoRefactor}\}, \tag{3.39}$$

де NoRefactor – відповідає випадкам, коли збереження дублікатів вважалося виправданим (контекстно залежна логіка, тимчасові реалізації і т.д.).

Для підвищення надійності розмітки кожен клон-клас анотувався двома незалежними експертами; у випадку розбіжностей рішення приймалося на основі узгоджувальної сесії або голосування третього експерта. Ступінь узгодженості оцінювався коефіцієнтом Каппа Коена κ, обчислюваним за стандартною формулою

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \tag{3.40}$$

де p_o – спостережувана частка збігів;

p_e – очікувана частка збігів за випадкового вибору.

Схема побудови корпусу та протоколу розмітки узагальнено показана на рисунку 3.6, що демонструє послідовність кроків від вихідного репозиторію до отримання структурованого набору даних, придатного для навчання та тестування моделі рекомендацій.

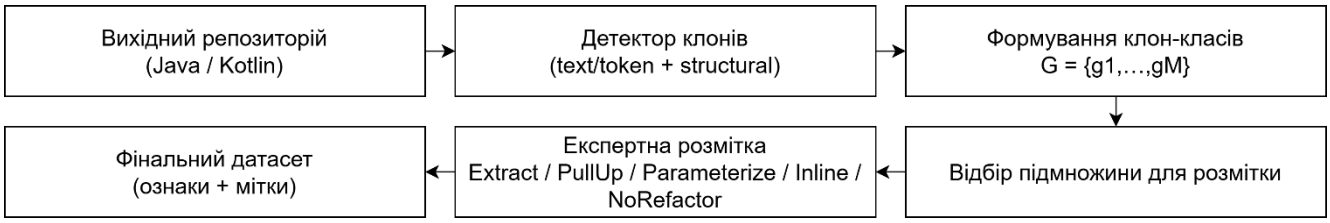

Рисунок 3.6 – Схема побудови корпусу дублікатів та протоколу експертної розмітки для навчання і тестування моделі рекомендації рефакторингів

Базові підходи для порівняння. Для коректної оцінки виграшу від використання еволюційних ознак та моделювання невизначеності було розгорнуто декілька базових підходів:

1) *модель без еволюційних ознак* – використовуються лише структурні та семантичні дескриптори (метрики розміру, когезії, структурні шаблони, embedding-подання коду), тоді як churn, історія co-change, вік компонентів та інформація про авторів вилучені із вектора ознак. Це дозволяє оцінити внесок еволюційного контексту в класифікацію та ранжування;

2) *модель без моделювання невизначеності* – архітектура та ознаки збігаються з повною моделлю, але видається лише точкова оцінка $p(y|x)$ без квантифікації невизначеності та без open-set-режиму. Всі зразки примусово віднесені до одного з відомих класів $\mathcal{Y}$, механізм утримання відсутній;

3) *rule-based підхід* – набір евристичних правил, що використовує порогові обмеження для окремих метрик (наприклад, розмір методу, відсоток дублікатів у файлі, різниця довжин клонів, кількість викликів у спільному контексті) для вибору типу рефакторингу або рішення «не рефакторити». Такі правила є узагальненням практик, описаних у класичній літературі з рефакторингу та технічного боргу, але не включають жодних даних про еволюцію проєкту.

Зведений опис базових підходів наведено в таблиці 3.4, що дозволяє інтерпретувати подальші порівняння результатів у контексті наявних/відсутніх компонентів моделі.

Таблиця 3.4 – Підсумковий опис базових підходів для порівняння

Позначення моделі	Ознаки (struct / sem / evol)	Наявність open-set-режиму	Тип прийняття рішення
Запропонована повна модель	є / є / є	є	ML
Модель без еволюційних ознак	є / є / немає	є	ML
Модель без моделювання невизначеності	є / є / є	немає	ML
Евристичний rule-based підхід	є / немає / немає	немає	rule-based

Якість класифікації типів рефакторингу та open-set-поведінки. Якість класифікації типів рефакторингу оцінювалася на рівні клон-класів g_j за стандартними метриками точності та макро-усередненого F-міра. Для множини тестових прикладів $\{(x_i, y_i)\}_{i=1}^{N}$ точність визначається як

$$\text{Acc} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(\hat{y}_i = y_i), \tag{3.41}$$

де $\hat{y}_i$ – прогнозований тип рефакторингу;

$\mathbb{I}(\cdot)$ – індикаторна функція.

Макро-усереднений F-міра обчислюється як

$$\mathrm{F1}_{\mathrm{macro}} = \frac{1}{|\mathcal{Y}|}\sum_{c\in\mathcal{Y}} \mathrm{F1}_c, \tag{3.42}$$

де $\mathrm{F1}_c$ – значення F-міри для окремого класу c.

Такий підхід забезпечує збалансовану оцінку для рідкісних типів рефакторингу.

Open-set-поведінка оцінювалася через аналіз пари «ризик–покриття» (risk–coverage). Нехай $C_i \in [0; 1]$ – міра впевненості моделі в рішенні для прикладу i, а τ – поріг абстиненції. Тоді покриття для заданого порогу

$$\mathrm{cov}(\tau) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(\, C_i \geq \tau), \tag{3.43}$$

а селективний ризик

$$\mathrm{risk}(\tau) = \frac{\sum_{i:C_i\geq\tau} \mathbb{I}(\,\hat{y}_i \neq y_i)}{\sum_{i:C_i\geq\tau} 1}. \tag{3.44}$$

Для open-set-сценаріїв, де частина дублікатів відповідає рефакторинговим діям, відсутнім у $\mathcal{Y}$, аналізувалася здатність моделі відсікти такі випадки за рахунок високої невизначеності та переходу в режим утримання.

На рисунку 3.7 показано, що модель з open-set-механізмом демонструє менший селективний ризик за того ж рівня покриття порівняно з варіантом без моделювання невизначеності, що підтверджує коректність її «обережної» поведінки щодо нетипових дублікатів.

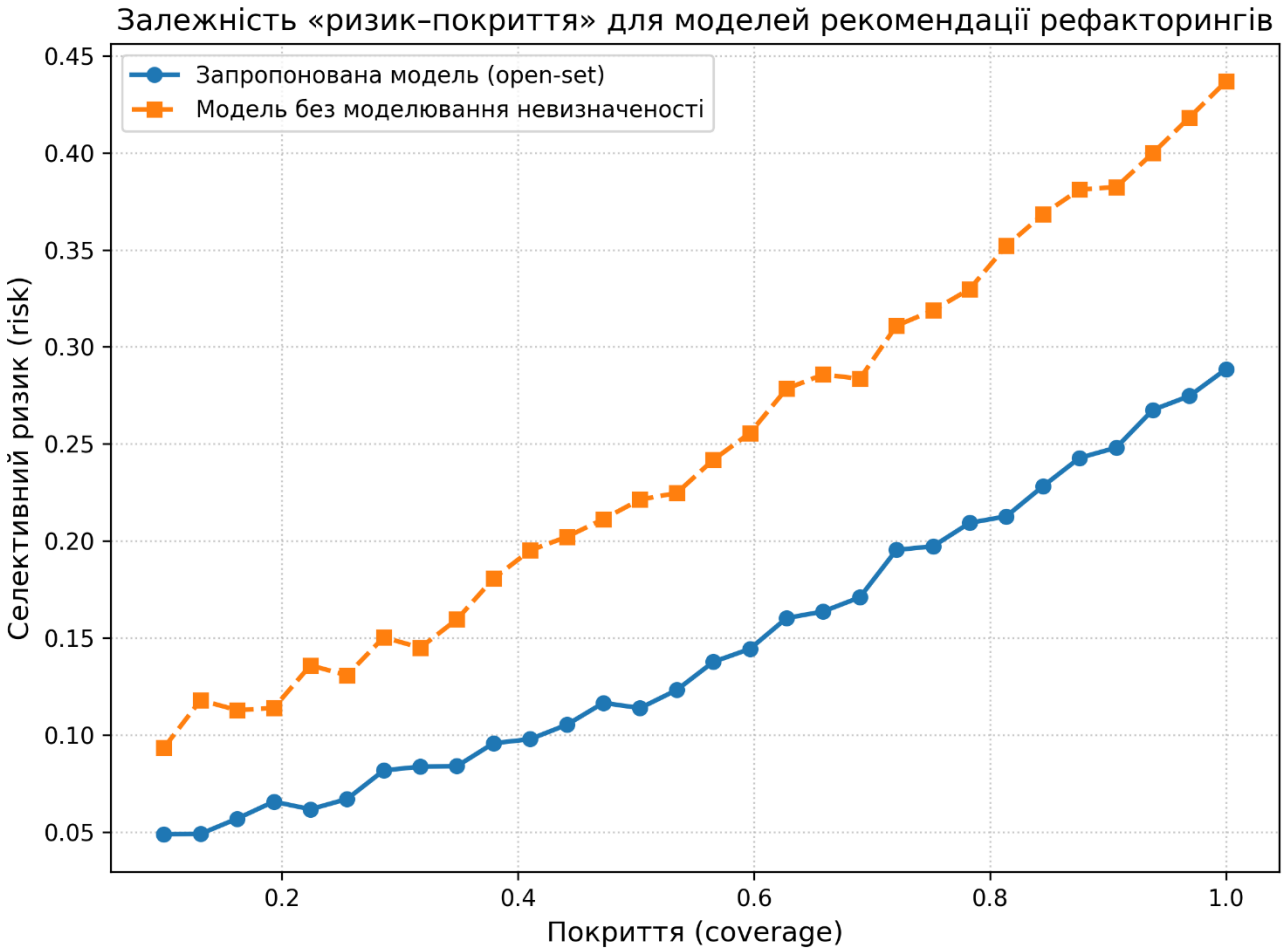

Рисунок 3.7 – Порівняння open-set-поведінки моделей за кривими «ризик–покриття»

Оцінка якості пріоритезації. Оскільки результат роботи системи подається розробникам у вигляді відсортованого списку рекомендованих рефакторингів, центральною метрикою є якість ранжування. Для її оцінювання використовувалася нормалізована кумулятивна вигода (NDCG@k), широко застосовувана в інформаційному пошуку та системах рекомендацій [149, 150].

Нехай для певного модуля проєкту модель генерує впорядкований список рефакторингових рекомендацій $(r_1, \dots, r_k)$, а кожному елементу r_i відповідає оцінка «корисності» $g_i \in \{0,1,2, \dots\}$, отримана за підсумками ручного аудиту або із залученням проксі-метрик (зменшення дефектності, зниження щільності клонів тощо). Тоді

$$\mathrm{DCG@k} = \sum_{i=1}^{k} \frac{2^{g_i} - 1}{\log_2(i+1)}, \qquad \mathrm{NDCG@k} = \frac{\mathrm{DCG@k}}{\mathrm{IDCG@k}}, \tag{3.45}$$

де IDCG@k – значення DCG@k для ідеального (за спаданням g_i) ранжування.

Додатково аналізувалася сукупна корисність top-k рекомендацій

$$U@k = \frac{1}{k}\sum_{i=1}^{k} u(r_i), \tag{3.46}$$

де $u(r_i) \in [0;1]$ – нормалізована оцінка ефекту від застосування конкретного рефакторингу (наприклад, за зміною обраних внутрішніх метрик якості або кількості дефектів після релізу).

На рисунку 3.8 наведено характерні криві NDCG@k для різних підходів. Запропонована модель демонструє систематичну перевагу у верхній частині списку (малі k), що є критично важливим у практичному контексті, де розробники зазвичай опрацьовують лише перші кілька рекомендацій у кожному релізному циклі.

Абляційний аналіз внеску структурних, семантичних та еволюційних ознак. Для кількісної оцінки внеску окремих груп ознак було проведено абляційний аналіз, у межах якого навчалися та тестувалися такі варіанти моделі:

– *Struct* – лише структурні метрики (розмір, складність, глибина вкладеності, показники когезії тощо);

– *Struct+Sem* – додатково включено семантичні представлення коду (embedding-и на основі шляхів у AST/CPG);

– *Struct+Evol* – структурні метрики та еволюційні ознаки (churn, co-change, вік файлів, кількість авторів);

– *Full* – повний набір ознак (структурні, семантичні та еволюційні) із моделюванням невизначеності.

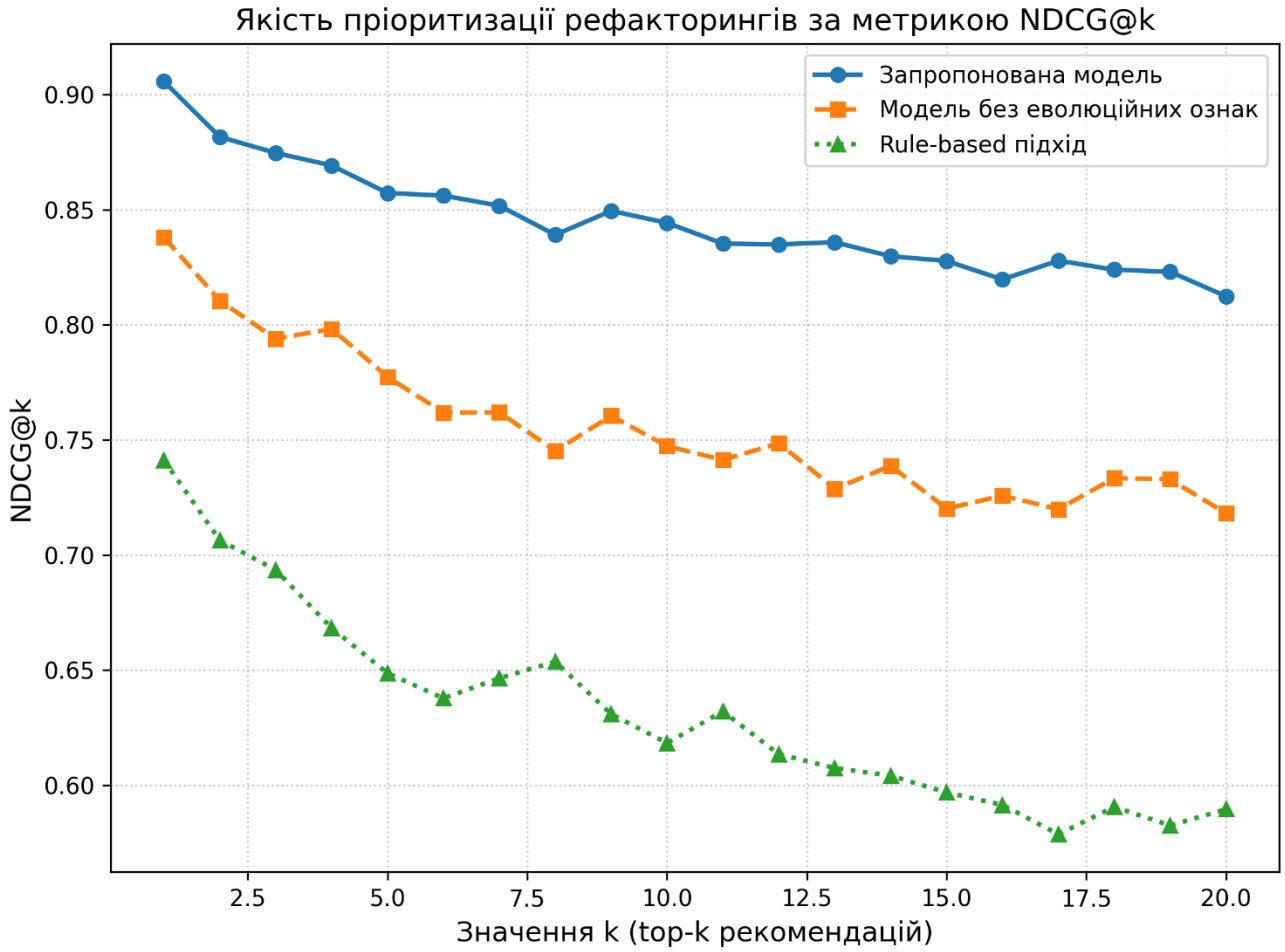

Рисунок 3.8 – Порівняння якості пріоритизації рефакторингів за метрикою NDCG@k для запропонованої моделі, варіанту без еволюційних ознак та rule-based підходу

Для кожного варіанта оцінювалися класифікаційні метрики Acc, $\mathrm{F1}_{\mathrm{macro}}$ та ранжувальні метрики NDCG@k, $U@k$. Для узагальнення впливу групи ознак $\mathcal{F}$ на метрику M використовувалася різниця

$$\Delta M_{\mathcal{F}} = M(\text{Full}) - M(\text{без } \mathcal{F}), \tag{3.47}$$

де $M(\text{Full})$ – значення метрики для повної моделі;

$M(\text{без } \mathcal{F})$ – для варіанта, у якому відповідні ознаки вилучено.

Зведені результати, подані у таблиці 3.5, свідчать, що структурні ознаки формують базовий рівень якості класифікації, тоді як семантичні представлення

коду суттєво підвищують $\mathrm{F1}_{\mathrm{macro}}$ завдяки кращому розрізненню типів рефакторингу з подібним структурним патерном. Еволюційні ознаки, у свою чергу, забезпечують найбільший приріст NDCG@k та $U@k$, оскільки дають змогу віддавати пріоритет рефакторингам, пов'язаним із «гарячими» компонентами, що часто змінюються або мають багату дефектну історію. Сукупний ефект трьох груп ознак, підсилений моделюванням невизначеності, формує основу для «безпечної» та практично корисної моделі рекомендації рефакторингів.

Таблиця 3.5 – Результати абляційного аналізу внеску груп ознак

Конфігурація моделі	Acc	F1macro	NDCG@5	U@5	ΔAcc	ΔF1macro	ΔNDCG@5	ΔU@5
Модель зі структурними ознаками	0,71	0,64	0,58	0,42	−0,09	−0,12	−0,18	−0,20
Модель зі структурними та семантичними ознаками	0,76	0,72	0,61	0,45	−0,04	−0,04	−0,15	−0,17
Модель зі структурними та еволюційними ознаками	0,74	0,67	0,70	0,59	−0,06	−0,09	−0,06	−0,03
Повна модель	0,80	0,76	0,76	0,62	0,00	0,00	0,00	0,00

3.3 Модель композиції «мінімально інвазивних» послідовностей рефакторингів на основі причинно-наслідкового аналізу

3.3.1 Постановка задачі планування послідовностей рефакторингів

Побудована в підрозділі 3.2 модель рекомендації рефакторингів генерує множину кандидатних перетворень коду, узгоджених з виявленими антипатернами з розділу 2. Однак у реальному проєкті рефакторинги рідко застосовуються ізольовано: їх необхідно впорядкувати у послідовності, сумісні з обмеженим бюджетом змін, вимогами до стабільності API та безперервністю CI/CD-процесів. Така постановка природно узгоджується з відомими підходами до планування рефакторингів і релізів, де зміни розглядаються як задачі багатокритеріальної оптимізації та комбінаторного планування [151, 152].

Нехай

$$R = \{r_1, \dots, r_n\} \tag{3.48}$$

– множина кандидатних рефакторингів, згенерованих на етапі 3.2, а

$$P = \{p_1, \dots, p_m\} \tag{3.49}$$

– множина екземплярів антипатернів, виявлених у проєкті. Кожному рефакторингу r_i відповідає:

– вартість змін (churn) $c(r_i) \in \mathbb{R}_+$, що оцінює кількість модифікованих рядків/файлів;

– множина усуваних антипатернів $\mathrm{cov}(r_i) \subseteq P$;

– індикатор (або ймовірність) порушення зовнішнього API $b(r_i) \in [0,1]$;

– індикатор впливу на стабільність CI/CD $\sigma(r_i) \in [0,1]$ (оцінка ризику падіння збірки, зростання часу прогону тестів тощо).

План рефакторингу задається впорядкованою послідовністю

$$\pi = (r_{i_1}, r_{i_2}, \dots, r_{i_L}), r_{i_k} \in R, \tag{3.50}$$

яка інтерпретується як послідовність застосування рефакторингів у межах одного або кількох релізних інтервалів. Для кожного інтервалу (спринта, релізу) накладається бюджет змін

$$\sum_{k=1}^{L} c(r_{i_k}) \leq B_{\text{churn}}, \tag{3.51}$$

де B_{churn} – максимальний обсяг безпечних модифікацій коду (визначається командою), що не порушує продуктивність команди та прийнятний рівень технічного ризику [153].

Обмеження на недопущення небажаних API-розривів формулюється як заборона рефакторингів, що змінюють публічні контракти, або як додаткове обмеження на сумарний «ступінь ламкості»

$$\sum_{k=1}^{L} b(r_{i_k}) \le B_{\text{api}}, \tag{3.52}$$

де B_{api} – зазвичай наближено до нуля для мінорних релізів.

Аналогічно, вимога стабільності CI/CD описується верхньою межею на сумарний ризик порушення конвеєра:

$$\sum_{k=1}^{L} \sigma(r_{i_k}) \le B_{\text{ci}}, \tag{3.53}$$

де B_{ci} – узгоджується з політикою команди щодо допустимої частоти падіння збірок.

Як показано на рисунку 3.9, простір можливих послідовностей можна інтерпретувати як орієнтований граф станів, де задача планування зводиться до пошуку шляху від поточного стану коду до стану з меншою питомою вагою антипатернів за умови дотримання ресурсних і ризикових обмежень. У цьому контексті «мінімально інвазивною» називатимемо таку послідовність $\pi^{\setminus *}$, яка досягає заданого рівня покращення внутрішньої якості (наприклад, усуває щонайменше частку τ сукупної ваги антипатернів) за мінімального сумарного втручання в кодову базу. Формально:

$$\pi^{\setminus *} = \arg\min_{\pi} \sum_{k=1}^{L} c(r_{i_k}) \tag{3.54}$$

за умов

$$Q(\pi) \ge \tau, \sum_{k=1}^{L} b(r_{i_k}) \le B_{\text{api}}, \sum_{k=1}^{L} \sigma(r_{i_k}) \le B_{\text{ci}}, \tag{3.55}$$

де $Q(\pi)$ – функція ефекту плану (наприклад, зменшення сумарної важкості антипатернів у модулі).

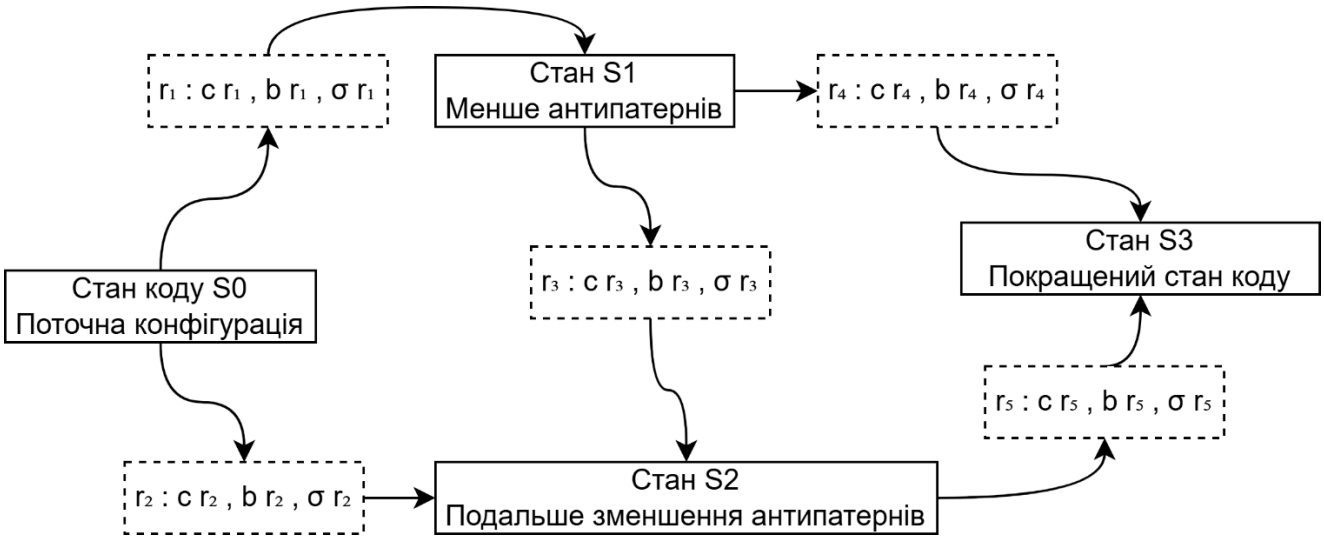

Рисунок 3.9 – Графова модель планування послідовностей рефакторингів

Таким чином, задача планування послідовностей рефакторингів у дисертації формулюється як задача обмеженої багатокритеріальної оптимізації, тісно пов'язана з дослідженнями у сфері планування рефакторингів та релізного планування [152–154].

3.3.2 Структура даних еволюції репозиторію та станів компонентів

Для задачі планування послідовностей рефакторингів потрібне уніфіковане подання еволюції програмного репозиторію, яке одночасно враховує структуру системи та часову динаміку змін. У дисертації еволюція репозиторію розглядається як послідовність інтервенцій (комітів, білдів, релізів) над графом компонентів, доповнена вектором ознак стану кожного модуля, що включає інформацію про антипатерни, метрики внутрішньої якості, еволюційні характеристики та CI-метрики [155, 156].

Нехай у момент часу t стан системи описується орієнтованим графом компонентів

$$G_t = (V, E_t), \tag{3.56}$$

де V – множина компонентів (модулів, пакетів, мікросервісів);

E_t – множина залежностей між ними (виклики методів, імпорти, мережеві виклики тощо).

Як показано на рисунку 3.10, кожна подія в історії $\mathcal{H}$ інтерпретується як перехід між станами графа, що дає змогу явно пов'язати локальні зміни в коді з еволюцією якісних характеристик компонентів та поведінкою CI/CD-конвеєра.

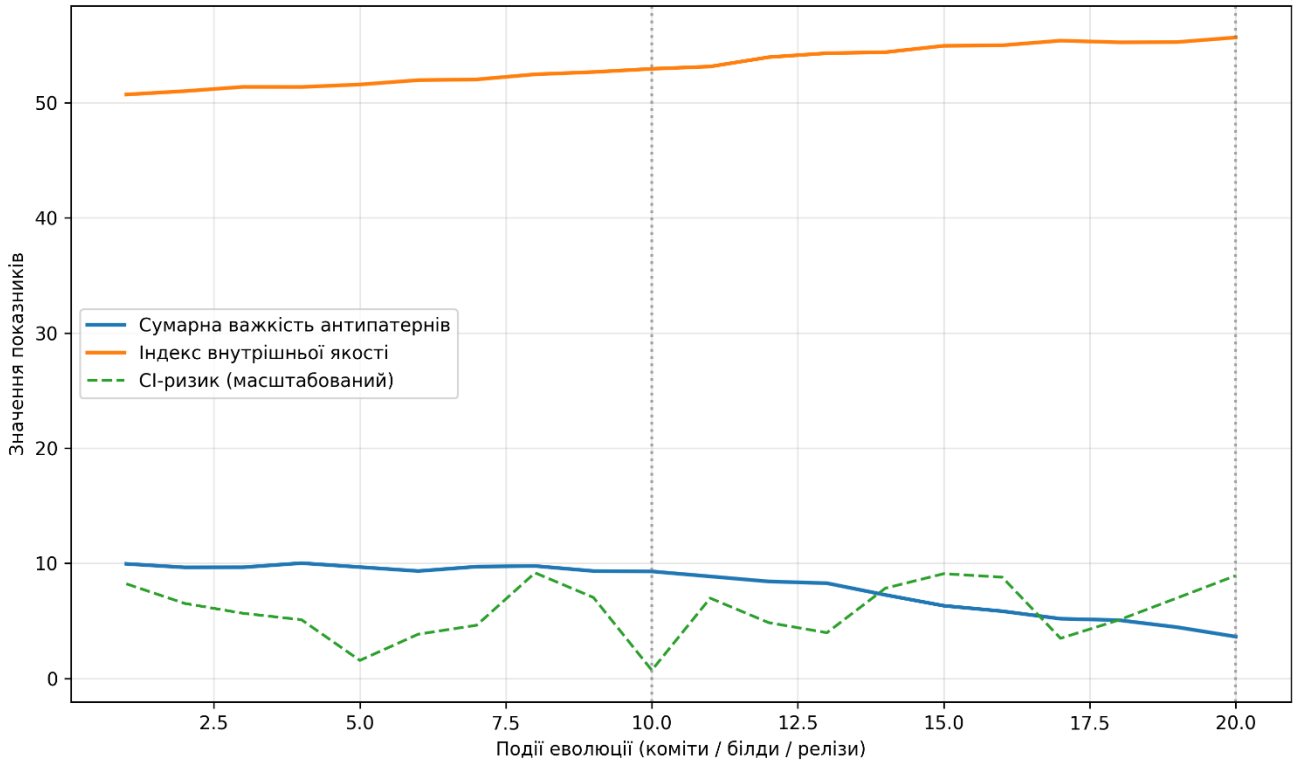

Рисунок 3.10 – Часово-орієнтована схема еволюції репозиторію

Історія репозиторію моделюється як впорядкована послідовність подій

$$\mathcal{H} = (e_1, e_2, \dots, e_T), \tag{3.57}$$

де кожна подія e_k є інтервенцією над графом $G_{t_{k-1}}$ і переводить систему у новий стан G_{t_k}. До множини подій входять:

– коміти (e_k^{commit}) – локальні модифікації підмножини вершин та ребер;

– білди (e_k^{build}) – запуск конвеєра збірки і тестування над поточним станом;

– релізи (e_k^{release}) – фіксація контрольних точок еволюції, пов'язаних із постачанням функціональності або технічного боргу [155].

Для кожного компонента $v \in V$ у момент часу t визначається вектор стану

$$\mathbf{x}_v(t) = [\mathbf{s}_v(t), \mathbf{q}_v(t), \mathbf{e}_v(t), \mathbf{c}_v(t)], \tag{3.58}$$

де $\mathbf{s}_v(t)$ – ознаки, пов'язані з антипатернами: кількість і типи кодових та архітектурних антипатернів, їх сумарна та зважена важкість у модулі (наприклад, число екземплярів «Long Method», «God Class», наявність клонів тощо). Для кожного типу антипатерну фіксуються як бінарні індикатори, так і числові оцінки важкості;

$\mathbf{q}_v(t)$ – вектор метрик внутрішньої якості: розмір (LOC), цикломатична складність, метрики зв'язності та зв'язаності (наприклад, CBO, LCOM), стабільність інтерфейсів, а також агреговані індекси підтримуваності, узгоджені з моделлю якості ISO/IEC 25010 [157];

$\mathbf{e}_v(t)$ – еволюційні характеристики, отримані з VCS-історії: churn за останні wкомітів, вік компонента (час від першої появи в репозиторії), частота змін, інтенсивність ко-змін із іншими модулями (кількість комітів, у яких vзмінювався разом із компонентом u), а також кількість дефектів, пов'язаних із модулем у задач-трекері [155, 156];

$\mathbf{c}_v(t)$ – CI-метрики, що відображають вплив компонента на стабільність конвеєра зборки: кількість падінь білда, у яких компонент входив до множини змінених файлів; середній час збірки після його змін; частота тестових збоїв, асоційованих із цим модулем; участь у «гарячих» релізах та аварійних виправленнях [157].

Для подальшого планування рефакторингів важливо, що вектор стану $\mathbf{x}_v(t)$ є узгодженим у часі: усі чотири групи ознак оновлюються після кожної події e_k, що

змінює відповідний компонент. Це дає можливість аналізувати траєкторії модулів у просторі ознак, виявляти «дегенеративні» траєкторії (зростання технічного боргу, збільшення нестабільності CI/CD) та оцінювати потенційний ефект кандидатів у рефакторинг, згенерованих моделлю підрозділу 3.2.

Для наочності основні групи ознак стану компонента зведено в таблицю 3.6, яка використовується як логічна схема для побудови конкретних реалізацій структури даних у прототипному інструментальному засобі.

Таблиця 3.6 – Групи ознак стану програмного компонента та приклади показників

Група ознак	Приклади метрик	Джерело даних
Ознаки антипатернів	Кількість екземплярів антипатернів («Long Method», «God Class», клоновані фрагменти); сумарна та зважена важкість антипатернів; бінарні індикатори наявності окремих типів антипатернів	Статичний аналіз коду
Метрики внутрішньої якості	Розмір компонента (LOC); цикломатична складність; показники зв’язаності та когезії (CBO, LCOM); стабільність інтерфейсів; агреговані індекси підтримуваності відповідно до ISO/IEC 25010	Статичний аналіз коду
Еволюційні характеристики	Churn за останні w комітів; вік компонента; частота змін; інтенсивність ко-змін з іншими модулями; кількість пов’язаних дефектів	VCS (Git), задач-трекер
CI-метрики	Кількість падінь білда за участі компонента; середній час збірки після змін; частота тестових збоїв; участь у аварійних виправленнях та «гарячих» релізах	CI/CD-конвеєр

3.3.3 Формальна модель причинно-наслідкових зв’язків між рефакторингами та якістю

Як було показано в підрозділі 3.3.2, у момент часу t кожний компонент v описується вектором стану (3.58), що включає інформацію про антипатерни,

метрики внутрішньої якості, еволюційні характеристики та CI-метрики. У цьому підрозділі вводиться формальна модель, яка пов’язує застосування рефакторингів до компонентів із подальшими змінами їхньої якості у причинно-наслідкових термінах, спираючись на апарат потенційних результатів (potential outcomes) [158, 159].

Потенційні результати для модулів та компонентів. Нехай для заданого інтервалу $[t, t+\Delta]$ розглядається бінарна змінна «лікування» (treatment)

$$A_v(t) \in \{0,1\}, \tag{3.59}$$

де $A_v(t) = 1$ – над компонентом v у цьому інтервалі було здійснено певний клас рефакторингу (або їх комбінацію);

$A_v(t) = 0$ – що рефакторинг для цього компонента не застосовувався.

Нехай $Y_v(t+\Delta)$ – скалярний показник якості компонента v в момент $t+\Delta$, який може відповідати, наприклад, агрегованому індексу підтримуваності (на основі метрик $\mathbf{q}_v$), ймовірності дефекту чи показнику негативного впливу на CI/CD. У моделі потенційних результатів для кожного компонента вводяться дві неспостережні гіпотетичні величини [158–160]:

– $Y_v^{(1)}(t+\Delta)$ – якість, якщо над v виконано рефакторинг у $[t, t+\Delta]$;

– $Y_v^{(0)}(t+\Delta)$ – якість, якщо над v рефакторинг не виконано у $[t, t+\Delta]$.

Спостережуваний результат має вигляд

$$Y_v(t+\Delta) = A_v(t)\, Y_v^{(1)}(t+\Delta) + (1 - A_v(t))\, Y_v^{(0)}(t+\Delta). \tag{3.60}$$

Таким чином, для кожного компонента реалізується лише один з двох потенційних результатів, що зумовлює фундаментальну проблему контрфактичного порівняння [159].

Індивідуальний (локальний) причинний ефект рефакторингу для компонента vу даному інтервалі визначається як

$$\tau_v(t) = Y_v^{(1)}(t+\Delta) - Y_v^{(0)}(t+\Delta), \tag{3.61}$$

тобто як різниця якості за сценарію з рефакторингом та без нього. Узагальнюючи, для вектора якості $\mathbf{Y}_v(t+\Delta)$ (наприклад, поєднання підтримуваності, надійності та продуктивності, узгоджених із ISO/IEC 25010 [111]) можна розглядати векторний ефект

$$\boldsymbol{\tau}_v(t) = \mathbf{Y}_v^{(1)}(t+\Delta) - \mathbf{Y}_v^{(0)}(t+\Delta), \tag{3.62}$$

але надалі для спрощення увага зосереджується на скалярному показнику Y_v.

Локальні та умовні середні ефекти рефакторингів. У реальних даних $\tau_v(t)$ безпосередньо недоступний, тому в моделі планувальника послідовностей рефакторингів використовуються статистичні оцінки локальних та умовних середніх ефектів [159–161].

Індивідуальний передбачений ефект (ITE, Individual Treatment Effect) для компонента vвизначається як умовне математичне сподівання:

$$\hat{\tau}_v(t) = \mathbb{E}[Y_v^{(1)}(t+\Delta) - Y_v^{(0)}(t+\Delta) | \mathbf{x}_v(t)]. \tag{3.63}$$

Ця величина інтерпретується як найкраща (в межах обраної моделі) оцінка очікуваного покращення або погіршення якості при застосуванні рефакторингу до конкретного компонента зі станом $\mathbf{x}_v(t)$. Для аналізу гетерогенності ефектів та побудови політик рефакторингу вводиться *умовний середній ефект (CATE, Conditional Average Treatment Effect)* для підпростору коваріат $\mathbf{X}$:

$$\tau(\mathbf{x}) = \mathbb{E}[Y^{(1)} - Y^{(0)} | \mathbf{X} = \mathbf{x}], \tag{3.64}$$

де $\mathbf{X}$ – вектор коваріат, що описують важливі властивості компонента та його роль у системі

У контексті даного дисертаційного дослідження, до $\mathbf{X}$ входять:

– складність – LOC, цикломатична складність, глибина вкладеності, значення класичних ООП-метрик [36, 39];

– роль в архітектурі – центральність у графі залежностей, належність до критичних сервісів, положення в архітектурних шарах [31, 106];

– історія змін – churn, вік компонента, частота ко-змін із іншими модулями, історія дефектів [38, 45, 49].

Тоді *CATE* дає змогу, наприклад, порівнювати очікуваний ефект одного й того ж типу рефакторингу для «молодих» та «старих» компонентів, для модулів з високою/низькою складністю чи для різних архітектурних ролей.

На рисунку 3.11 схематично показано причинно-наслідкову структуру, що лежить в основі моделі.

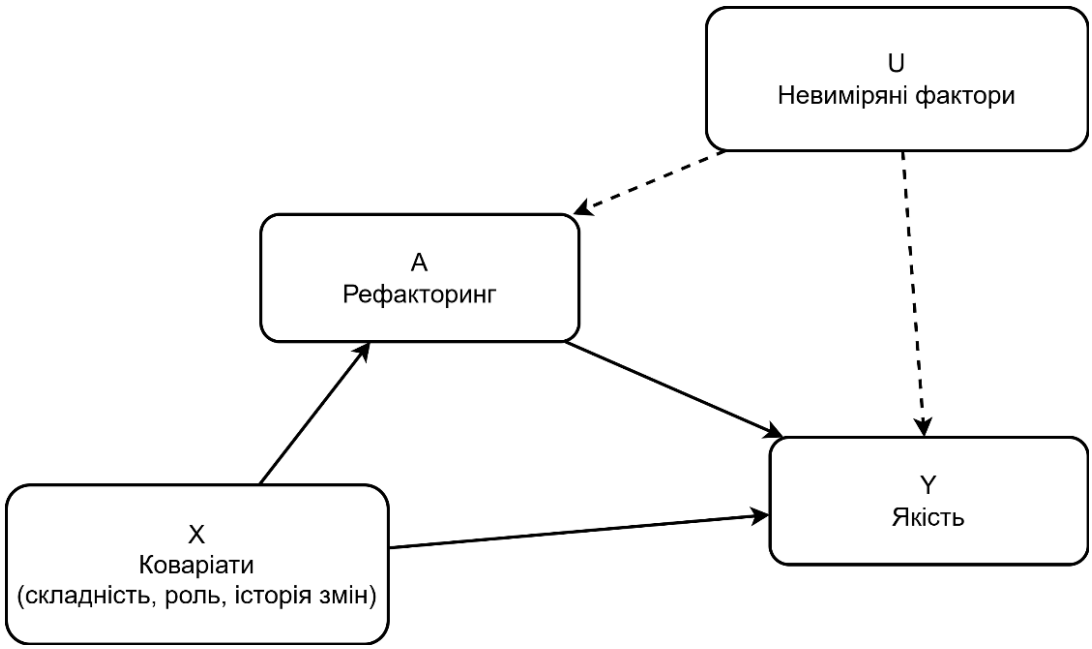

Рисунок 3.11 – Спрощена причинно-наслідкова діаграма, що відображає вплив рефакторингу на якість програмного компонента з урахуванням коваріат стану та можливих невиміряних факторів

Коваріати **X** впливають і на ймовірність застосування рефакторингу, і на результуючу якість, тому коректна оцінка $\tau(\mathbf{x})$ вимагає коригування за **X** (matching, вагове коригування чи моделювання результату) [158–161].

Врахування коварат у причинно-наслідковій моделі. Базове припущення моделі – *умовна незалежність (strong ignorability)* [158, 159]:

$$\left(Y^{(0)}, Y^{(1)}\right) \perp\!\!\!\perp A|\mathbf{X}, \tag{3.65}$$

тобто після фіксації коваріат $\mathbf{X}$ вибір модулів для рефакторингу може вважатися квазі-випадковим.

Це дозволяє оцінювати CATE за допомогою моделей результату. Один з можливих варіантів – лінійна або узагальнена лінійна модель з перетином між лікуванням і коваріатами:

$$Y = \beta_0 + \beta_A A + \boldsymbol{\beta}_X^{\mathrm{T}}\mathbf{X} + \boldsymbol{\beta}_{AX}^{\mathrm{T}}(A \cdot \mathbf{X}) + \varepsilon, \tag{3.66}$$

де $A \cdot \mathbf{X}$ – покомпонентний добуток;

ε– випадкова похибка.

У такому разі умовний середній ефект для компонентів з характеристиками $\mathbf{X} = \mathbf{x}$ дорівнює

$$\tau(\mathbf{x}) = \mathbb{E}[Y \mid A = 1, \mathbf{X} = \mathbf{x}] - \mathbb{E}[Y \mid A = 0, \mathbf{X} = \mathbf{x}] = \beta_A + \boldsymbol{\beta}_{AX}^{\mathrm{T}}\mathbf{x}, \tag{3.67}$$

а індивідуальна оцінка $\hat{\tau}_v(t)$ отримується підстановкою $\mathbf{x}_v(t)$ у цю формулу.

З практичної точки зору, у межах запропонованої моделі планувальник послідовностей рефакторингів використовує оцінки $\hat{\tau}_v(t)$ як «корисність» (expected utility) для кандидата рефакторингу над компонентом v з урахуванням його поточного стану та контексту.

Високі значення $\hat{\tau}_v(t)$ за одночасно прийнятного ризику (невизначеності) слугують підставою для включення рефакторингу у послідовність, тоді як для компонентів з низьким або негативним очікуваним ефектом рефакторинг може бути відкладений або відхилений.

3.3.4 Запропонована причинно-орієнтована модель планування «мінімально інвазивних» послідовностей

У попередньому підрозділі було введено індивідуальні та умовні середні причинні ефекти рефакторингів $\hat{\tau}_v(t)$ для окремих компонентів. У цьому підрозділі ці локальні оцінки використовуються як будівельні блоки причинно-орієнтованої моделі планування послідовностей рефакторингів, що максимізують очікуваний приріст якості при жорсткому контролі інвазивності змін. Під «мінімально інвазивними» розуміються плани, які досягають суттєвого покращення внутрішньої якості при обмеженому churn, мінімальній ймовірності зламу зовнішніх API та низькому ризику порушення CI/CD-процесів [162–165].

Корисність плану з урахуванням impact-обмежень. Нехай П – множина планів-кандидатів рефакторингів. Кожен план $\pi \in$ П визначається впорядкованою послідовністю кроків

$$\pi = (a_1, a_2, \dots, a_K), \tag{3.68}$$

де a_k – крок, який описується трійкою (v_k, r_k, t_k) – компонент v_k, тип рефакторингу r_k і момент (ітерація) застосування t_k.

Для кроку a_k причинно-орієнтований модуль оцінювання надає прогнозований індивідуальний ефект $\hat{\tau}(a_k)$, що відповідає очікуваному приросту обраної метрики якості для компонента v_k за умов застосування рефакторингу r_k[158–162]. Очікуваний приріст інтегральної якості для плану πзадається як

$$\Delta Q(\pi) = \sum_{k=1}^{K} \hat{\tau}(a_k). \tag{3.69}$$

Паралельно вводиться вектор impact-показників плану

$$\mathbf{I}(\pi) = \left(I^{churn}(\pi), I^{api}(\pi), I^{ci}(\pi)\right), \tag{3.70}$$

де $I^{\text{churn}}(\pi)$ – сумарний обсяг змінених рядків коду (чутливий до ризику дефектів та merge-конфліктів) [45, 155];

$I^{\text{api}}(\pi)$ – міра потенційної несумісності API (зміни публічних сигнатур, контрактів);

$I^{\text{ci}}(\pi)$ – оцінка ризику для CI/CD (ймовірність падіння збірки, тривалість прогону тестів, нестабільність пайплайнів) [104, 108].

Для кожного виду impact задається верхня межа допустимого впливу

$$\mathbf{B} = (B^{\text{churn}}, B^{\text{api}}, B^{\text{ci}}), \tag{3.71}$$

що відображає політику організації щодо обмеження технічного боргу й змін у продуктивному середовищі [8, 9].

Тоді базова корисність плану визначається як

$$U(\pi) = \Delta Q(\pi) \text{ за умов } \mathbf{I}(\pi) \leq \mathbf{B}, \tag{3.72}$$

тобто план вважається прийнятним лише у разі виконання компонентних обмежень

$$I^{\text{churn}}(\pi) \leq B^{\text{churn}}, \qquad I^{\text{api}}(\pi) \leq B^{\text{api}}, \qquad I^{\text{ci}}(\pi) \leq B^{\text{ci}}. \tag{3.73}$$

Задача вибору «мінімально інвазивної» послідовності може бути сформульована як задача багатокритеріальної оптимізації: максимізувати $\Delta Q(\pi)$ при мінімізації $\mathbf{I}(\pi)$ та виборі планів, що лежать на Парето-фронті «якість – інвазивність» [164, 165].

На рисунку 3.12 показано, що запропонована модель відбирає плани, розташовані в зоні високого приросту якості та помірного impact, відкидаючи як «ризикові» послідовності з високим churn/API-впливом і нестабільним CI.

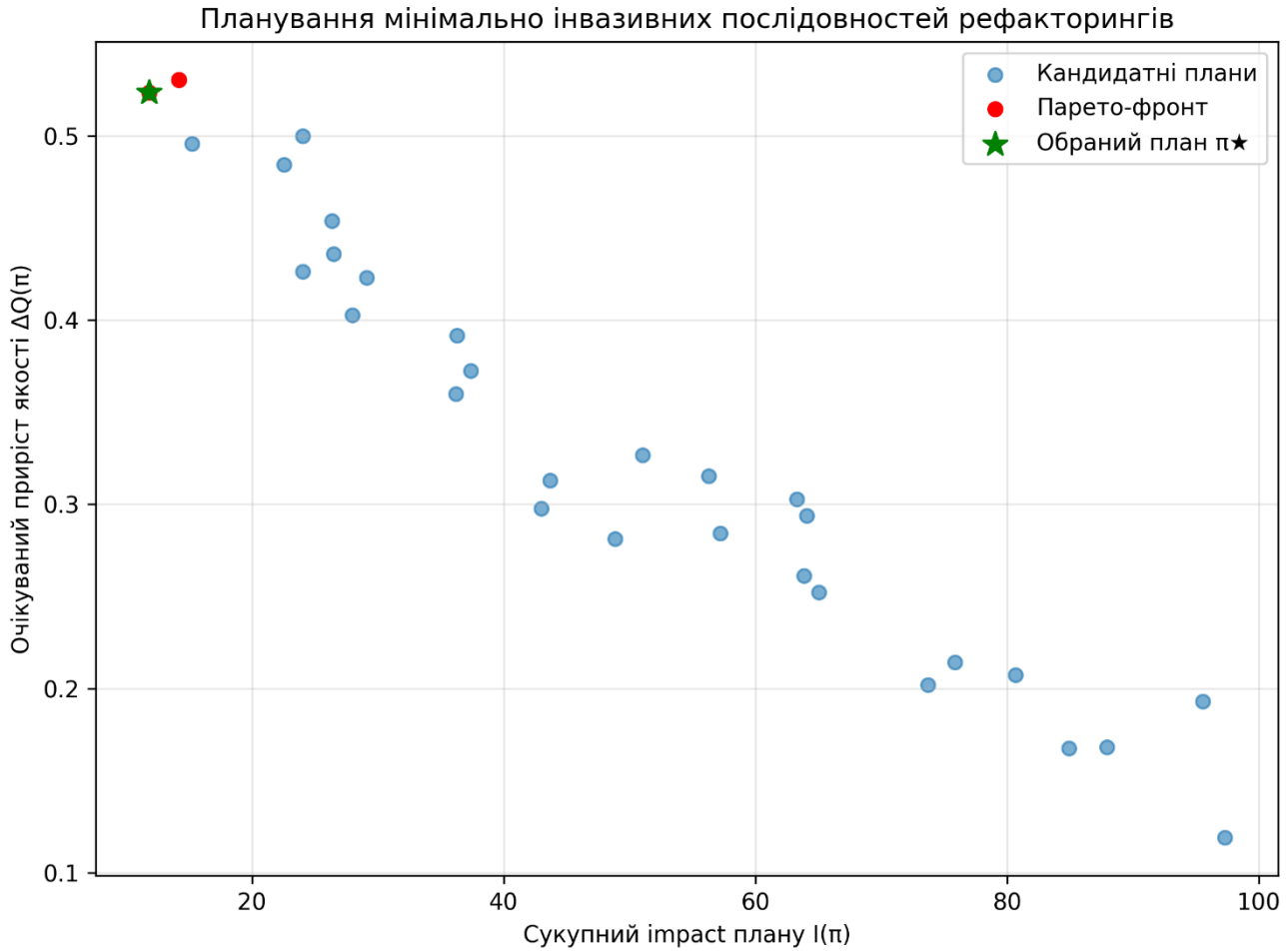

Рисунок 3.12 – Діаграма «очікуваний приріст якості – impact» для кандидатних планів рефакторингів

Консервативне планування на основі нижніх довірчих меж. Оскільки оцінки $\hat{\tau}(a_k)$ отримуються з моделей машинного навчання та містять статистичну невизначеність [105, 107], використання тільки точкових прогнозів може призводити до надто агресивних планів. Для уникнення ризикових рішень у моделі введено *консервативну корисність* на основі нижніх довірчих меж.

Нехай для кожного кроку a_k додатково оцінюються стандартне відхилення $\sigma(a_k)$ або інша міра невизначеності (наприклад, ширина предиктивного інтервалу) [142, 143]. Для заданого рівня довіри $1 - \alpha$ нижня довірча межа локального ефекту запишеться як

$$\underline{\tau}_{\alpha}(a_k) = \hat{\tau}(a_k) - z_{1-\alpha}\,\sigma(a_k), \tag{3.74}$$

де $z_{1-\alpha}$ – квантиль стандартного нормального розподілу.

Тоді *консервативна корисність* плану визначається як

$$U_{\alpha}^{-}(\pi) = \sum_{k=1}^{K} \underline{\tau}_{\alpha}(a_k), \tag{3.75}$$

і задача планування набуває вигляду

$$\max_{\pi \in \Pi} U_{\alpha}^{-}(\pi) \text{ за умов } \mathbf{I}(\pi) \leq \mathbf{B}. \tag{3.76}$$

Таким чином, навіть якщо точкові оцінки $\hat{\tau}(a_k)$ для деяких кроків є високими, але невизначеність $\sigma(a_k)$ значна, їх внесок у $U_{\alpha}^{-}(\pi)$ буде зменшений, що автоматично обмежує включення ризикових рефакторингів до фінального плану. Такий підхід є аналогом risk-averse оптимізації у задачах стохастичного програмування та робастної оптимізації [163, 164].

Обмеження на порядок і сумісність кроків. Окрім числових обмежень на impact, планувальник повинен гарантувати *коректний порядок і логічну сумісність* кроків послідовності. Для цього множина кроків-кандидатів $\{a_k\}$ розглядається як набір бінарних змінних

$$x_k \in \{0,1\}, \tag{3.77}$$

де $x_k = 1$ – включення кроку a_k у план.

Тоді консервативне планування можна записати у вигляді змішано-цілочисельної оптимізаційної задачі:

$$\begin{gathered} \max_{x} U_{\alpha}^{-}(x) = \sum_{k} x_k \underline{\tau}_{\alpha}(a_k), \\ \text{за умов } \textstyle\sum_k x_k\, c^{churn}(a_k) \leq B^{churn}, \\ \sum_k x_k c^{api}(a_k) \leq B^{api}, \sum_k x_k c^{ci}(a_k) \leq B^{ci}, \end{gathered} \tag{3.78}$$

де $c^{\text{churn}}(a_k)$, $c^{\text{api}}(a_k)$, $c^{\text{ci}}(a_k)$ – одиничні impact-витрати для кроку a_k.

Обмеження на порядок і сумісність кроків формалізуються додатковими лінійними співвідношеннями [153, 154, 166]:

– передування (precedence) – якщо застосування кроку a_j потребує попереднього виконання a_i (наприклад, виділення методу можливе лише після спрощення умовного оператора), вводиться обмеження $x_j \leq x_i$.

– взаємна несумісність (conflict) – якщо кроки a_i та a_j не можуть бути виконані в одному релізному циклі (наприклад, два різні варіанти рефакторингу одного й того ж інтерфейсу), використовується

$$x_i + x_j \leq 1. \tag{3.79}$$

– глобальні інваріанти – збереження коректності збірки та тестів моделюється як система обмежень

$$g_m(\mathrm{x}) \leq 0, \qquad m = 1, \dots, M, \tag{3.80}$$

де g_m – функції, кожна з яких відповідає певному інваріанту (наприклад, збереження контрактів зовнішніх API, відсутність зламаних залежностей між модулями, успішне проходження обов’язкових smoke-тестів).

В таблиці 3.7 узагальнено основні класи обмежень, що використовуються в причинно-орієнтованій моделі планування, разом з їх математичними формулюваннями та інтуїтивними прикладами з промислових проєктів.

У підсумку, запропонована причинно-орієнтована модель формулює задачу вибору послідовності рефакторингів як задачу консервативної багатокритеріальної оптимізації: план максимізує нижню довірчу межу очікуваного приросту якості, водночас залишаючись в межах заданих impact-бюджетів і дотримуючись структурних обмежень на порядок і сумісність кроків. Це забезпечує «мінімально інвазивні» зміни коду, узгоджені з практиками обережного управління технічним боргом і релізним плануванням у масштабних програмних системах [163–167].

Таблиця 3.7 – Класи обмежень у причинно-орієнтованій моделі планування послідовностей рефакторингів

Клас обмежень	Формальний запис	Інтуїтивна інтерпретація (приклад з промислових проєктів)
Impact-обмеження (churn)	$\sum_k x_k c^{churn}(a_k) \le B^{churn}$	Обмеження сумарного обсягу змін коду в одному релізному циклі для запобігання різкому зростанню дефектів, merge-конфліктів і перевантаженню команди
Impact-обмеження (API)	$\sum_k x_k c^{api}(a_k) \le B^{api}$	Заборона або жорстке обмеження змін публічних інтерфейсів у мінорних релізах з метою збереження зворотної сумісності
Impact-обмеження (CI/CD)	$\sum_k x_k c^{ci}(a_k) \le B^{ci}$	Контроль сумарного ризику порушення CI/CD-конвеєра (падіння збірок, збільшення часу тестування, нестабільні пайплайни)
Обмеження порядку (precedence)	$x_j \le x_i$	Деякі рефакторинги можуть виконуватися лише після попередніх кроків (наприклад, Extract Method після спрощення умовної логіки)
Обмеження сумісності (conflict)	$x_i + x_j \le 1$	Взаємовиключні рефакторинги одного компонента або інтерфейсу, які не можуть бути виконані в одному релізі
Глобальні інваріанти	$g_m(x) \le 0,$ $m = 1, \dots, M$	Збереження ключових властивостей системи: коректна збірка, проходження обов'язкових тестів, відсутність зламаних міжмодульних залежностей

3.3.5 Метод побудови планів рефакторингу з бюджетними та інтерфейсними обмеженнями

Запропонована в попередніх підрозділах причинно-орієнтована модель дає змогу отримувати оцінки очікуваної корисності окремих кроків та їх послідовностей. У цьому підрозділі формалізується метод побудови конкретних планів рефакторингу у вигляді обмеженого пошуку в просторі дій, що враховує бюджетні (churn, зміни API, ризики для CI/CD) та інтерфейсні обмеження. На відміну від локальних стратегій «крок-за-кроком», метод працює з послідовностями атомарних рефакторингів, гарантуючи дотримання глобальних обмежень та припиняючи пошук при досягненні цільового рівня корисності.

Простір дій і станів. Нехай $\mathcal{A}$ – каталог атомарних рефакторингів, що включає класичні операції («Extract Method», «Move Method», «Inline Class» тощо) та спеціалізовані рекомендації, сформовані на основі моделі розділу 3.2. Для кожної дії $a \in \mathcal{A}$ зберігається:

– оцінений вектор впливу

$$\mathbf{c}(a) = \left(c_{churn}(a), c_{api}(a), c_{ci}(a)\right) \tag{3.81}$$

– прогнозований приріст якості $\Delta Q(a)$ та пов'язана з ним невизначеність (дисперсія, довірчий інтервал), отримані з причинно-орієнтованої моделі.

План рефакторингу задається скінченною послідовністю атомарних кроків

$$\pi = (a_1, a_2, \dots, a_K), a_k \in \mathcal{A}. \tag{3.82}$$

Кумулятивний «impact» плану описується векторною сумою

$$\mathbf{C}(\pi) = \sum_{k=1}^{K} \mathbf{c}(a_k) \tag{3.83}$$

із поелементним порівнянням із вектором бюджетів $\mathbf{B} = (B_{\text{churn}}, B_{\text{api}}, B_{\text{ci}})$. План вважається допустимим, якщо

$$\mathbf{C}(\pi) \leq \mathbf{B} \tag{3.84}$$

у сенсі поелементної нерівності. Інтерфейсні обмеження (стабільність публічних API, контрактів між сервісами тощо) кодуються як предикати $g_j(\pi) \in \{0,1\}$, що мають виконуватися для всього плану:

$$g_j(\pi) = 1 \forall j \in J. \tag{3.85}$$

Евристичний пошук планів з обрізанням (pruning). Побудова планів формулюється як задача евристичного пошуку в дереві станів, де вузол n відповідає частковому плану $\pi_{1:k}$ і поточному стану репозиторію, а дуга – застосуванню чергового атомарного рефакторингу. Для навігації в цьому дереві використовуються алгоритми best-first або A* або beam-search, добре відомі в задачах планування [168–170].

Для кожного вузла nобчислюється оцінка «якості» плану з урахуванням бюджетів та невизначеності:

$$F(n) = U_{\alpha}^{-}(\pi_{1:k}) - \lambda \cdot \phi(\mathrm{C}(\pi_{1:k}), \mathbf{B}). \tag{3.86}$$

де $U_{\alpha}^{-}(\pi)$ – нижня довірча межа очікуваного приросту якості;

$\phi(\cdot)$ – функція штрафу за наближення до бюджетів;

$\lambda \geq 0 -$ коефіцієнт ваги штрафу

У випадку використання алгоритму A* оцінка розщеплюється на фактичну частину $G(n)$ (накопичений приріст якості для $\pi_{1:k}$) та евристичну верхню оцінку потенційного приросту від ще не виконаних кроків $H(n)$ [168, 169]:

$F(n) = G(n) + H(n), H(n)$ – адитивна й припустима евристика. (3.87)

Pruning (відсікання) виконується на рівні вузла n, якщо виконується хоча б одна з умов:

– бюджетне порушення: $\mathbf{C}(\pi_{1:k}) \not\leq \mathbf{B}$;

– порушення інтерфейсних обмежень: існує j, для якого $g_j(\pi_{1:k}) = 0$;

– «нереалістична» невизначеність: довірчий інтервал $[U_{\alpha}^{-}(\pi_{1:k}), U_{\alpha}^{+}(\pi_{1:k})]$ повністю лежить нижче поточного найкращого значення, тобто навіть оптимістичний сценарій не обіцяє покращення порівняно з уже знайденими планами.

Схематичну структуру дерева пошуку з позначенням вузлів, відсічених різними типами обмежень проілюстровано на рисунку 3.13. На ньому гілки, що

порушують бюджети, виділено одним типом штрихування, гілки з порушенням інтерфейсних контрактів – іншим, а гілки, відкинуті через високу невизначеність, – третім.

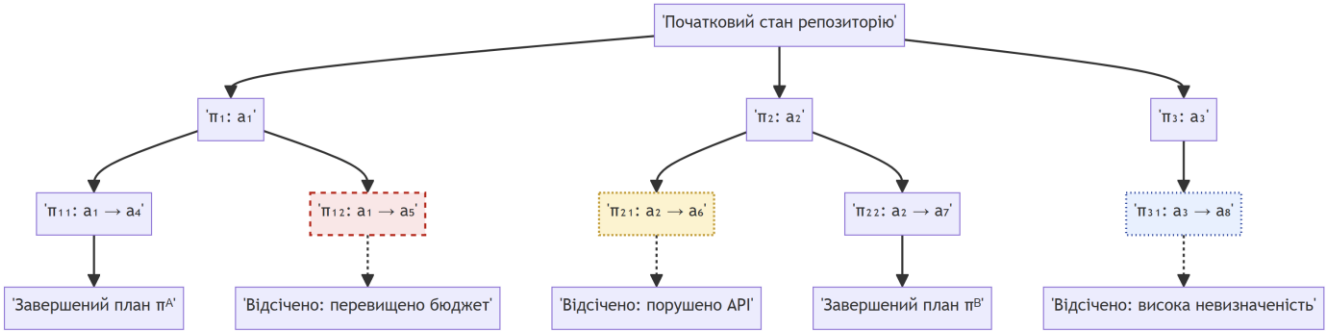

Рисунок 3.13 – Евристичний пошук планів рефакторингу з обрізанням за бюджетними, інтерфейсними та невизначеними гілками

Формальні правила відсікання зведені в таблиці 3.8, що використовується надалі для реалізації узагальненого планувальника.

Таблиця 3.8 – Правила відсікання гілок у пошуку планів рефакторингу

Тип обмеження	Формальна умова	Дія планувальника
Бюджетне обмеження	$C(\pi) \not\leq B$	Відсікання вузла пошуку, оскільки частковий або повний план перевищує допустимі бюджети за churn, змінами API або ризиками для CI/CD
Інтерфейсне обмеження	$\exists j: g_j(\pi) = 0$	Відсікання вузла пошуку через порушення інтерфейсних контрактів або інших глобальних структурних вимог
Висока невизначеність	$U_\alpha^+(\pi) \leq U_\alpha^-(\pi^*)$	Відсікання вузла пошуку, оскільки навіть оптимістична оцінка корисності плану не перевищує поточний найкращий знайдений план (π^*)

Критерії зупинки пошуку. Пошук завершується за одним із таких критеріїв:

– досягнення цільової корисності – існує план $\pi^\star$, для якого

$$U_\alpha^-(\pi^\star) \geq U_{\text{target}}, \tag{3.88}$$

де U_{target} – задається архітектором або командою як бажаний рівень покращення внутрішньої якості при прийнятному ризику.

У цьому випадку $\pi^{\star}$ приймається як рекомендований план рефакторингу.

– вичерпання бюджету – жоден із планів, які ще можна згенерувати в межах дерева пошуку, не задовольняє $\mathbf{C}(\pi) \leq \mathbf{B}$. Формально, фронт пошуку складається лише з вузлів, що порушують бюджетні обмеження або вже відсічені. Тоді алгоритм повертає найкращий знайдений допустимий план (якщо такий існує) або порожню рекомендацію (відсутність безпечного плану в заданих межах);

– висока невизначеність ефекту – якщо для всіх вузлів на фронті $\mathcal{F}$ виконується

$$U_{\alpha}^{+}(\pi) < U_{\alpha}^{-}(\pi^{\star}), \forall\, \pi \in \mathcal{F}, \tag{3.89}$$

або, більш консервативно, дисперсія оцінок перевищує наперед заданий поріг $\sigma_{\max}$, алгоритм зупиняється і сигналізує про недостатню визначеність прогнозів (наприклад, через малий обсяг даних або нестандартність проєкту) [126, 142].

З точки зору класичної теорії пошуку в просторі станів, описаний метод відповідає задачі одноцільового планування з жорсткими обмеженнями, розв’язуваної евристичним або A*-пошуком [168, 169].

Додавання інтерфейсних та бюджетних обмежень переводить задачу в клас обмеженого (constrained) планування, який традиційно розглядається в контексті автоматизованого планування та керованого пошуку [170].

Вбудований механізм роботи з невизначеністю дає змогу уникати «агресивних» планів, ефект яких моделлю оцінюється нестабільно, і тим самим інтегрує сучасні уявлення про невизначеність у програмній аналітиці в практичну процедуру планування рефакторингів [126, 142].

3.3.6 Інтеграція виходів модуля рекомендацій в планувальник

Запропонований у розділі 3.2 модуль рекомендацій формує для кожної потенційної операції рефакторингу вектор оцінок виду

$$(b(a), e(a), u(a)), \tag{3.90}$$

де $b(a)$ – очікувана вигода (benefit);

$e(a)$ – оцінка трудомісткості (effort);

$u(a)$ – невизначеність прогнозу.

У цьому підрозділі формалізується спосіб інтеграції цих виходів у планувальник послідовностей рефакторингів, описаний у підрозділах 3.3.4–3.3.5, з урахуванням ієрархії цілей: від локального усунення окремих клонів до підвищення глобальної якості компонентів та проєкту в цілому.

Використання вигоди/трудомісткості/невизначеності як ваг і пріоритетів. Нехай $\mathcal{A}$ – множина операцій-кандидатів на рефакторинг (у тому числі рекомендацій, орієнтованих на усунення клонів та інших антипатернів). Для кожної операції $a \in \mathcal{A}$ модуль рекомендацій надає оцінки:

– $b_{\text{loc}}(a)$ – приріст локальної якості (наприклад, зниження клонованості або метрик складності у відповідній області коду),

– $e(a)$ – очікувані витрати (кількість модифікованих рядків, touch-поінтів у файлах, ризик конфліктів у гілках тощо),

– $u(a)$ – невизначеність прогнозу корисності.

Для інтеграції в планувальник вводиться нормалізована “корисність” операції:

$$w(a) = \frac{b_{\text{loc}}(a)}{e(a)} \cdot \psi(u(a)), \tag{3.91}$$

де $\psi(u)$ – функція штрафу за невизначеність (наприклад, монотонно спадна функція, що зменшує вагу дій із високою невизначеністю).

У консервативному варіанті замість $b_{\text{loc}}(a)$ використовується нижня довірча межа $b_{\alpha}^{-}(a)$, що узгоджується з підходами robust/uncertainty-aware оптимізації [171]. Величина $w(a)$ використовується:

– як ключ сортування у черзі пріоритетів best-first/beam-пошуку;

– як доданок до евристичної функції $H(n)$ у A*-пошуку;

– як один з критеріїв відсікання дій із низьким відношенням «корисність/витрати».

Таким чином, виходи модуля рекомендацій не замінюють власне процедуру планування, а забезпечують її керування у вигляді пріоритизації розширення гілок дерева пошуку, що відповідає ідеям використання рекомендаційних моделей як «евристичних оракулів» у задачах планування та оптимізації [172].

Схема інтеграції виходів модуля рекомендацій у процедуру планування послідовностей рефакторингу наведена на рисунку 3.14.

Узгодження локальних і глобальних цілей. На рівні виходів модуля рекомендацій локальна корисність $b_{\text{loc}}(a)$ орієнтована на «мікро-цілі» – наприклад, усунення конкретного клонного класу або зменшення глибини вкладеності в окремому методі. Водночас метою планувальника є підвищення глобальної якості компонентів і системи в цілому (метрики підтримуваності, еволюційної стабільності, ризику дефектів тощо). Для узгодження цих рівнів вводиться функція лінкування локальної та глобальної корисності.

Нехай операція aприв’язана до компонента $C(a)$ (модуль, сервіс, підсистема). Визначимо:

– локальний ефект:

$$\Delta Q_{\text{loc}}(a) = b_{\text{loc}}(a), \tag{3.92}$$

– глобальний ефект через компонентну якість:

$$\Delta Q_{\text{glob}}(a) = \Delta Q_{\text{comp}}(C(a)) + \Delta Q_{\text{sys}}(a), \tag{3.93}$$

де ΔQ_{comp} – прогнозована зміна метрик якості компонента після застосування операції (агрегація змін метрик типу «клонованість», «зв'язність», «складність»);

ΔQ_{sys} – зміна на рівні системи (наприклад, зменшення кількості міжкомпонентних клонів або зниження ризику одночасних змін у кількох сервісах).

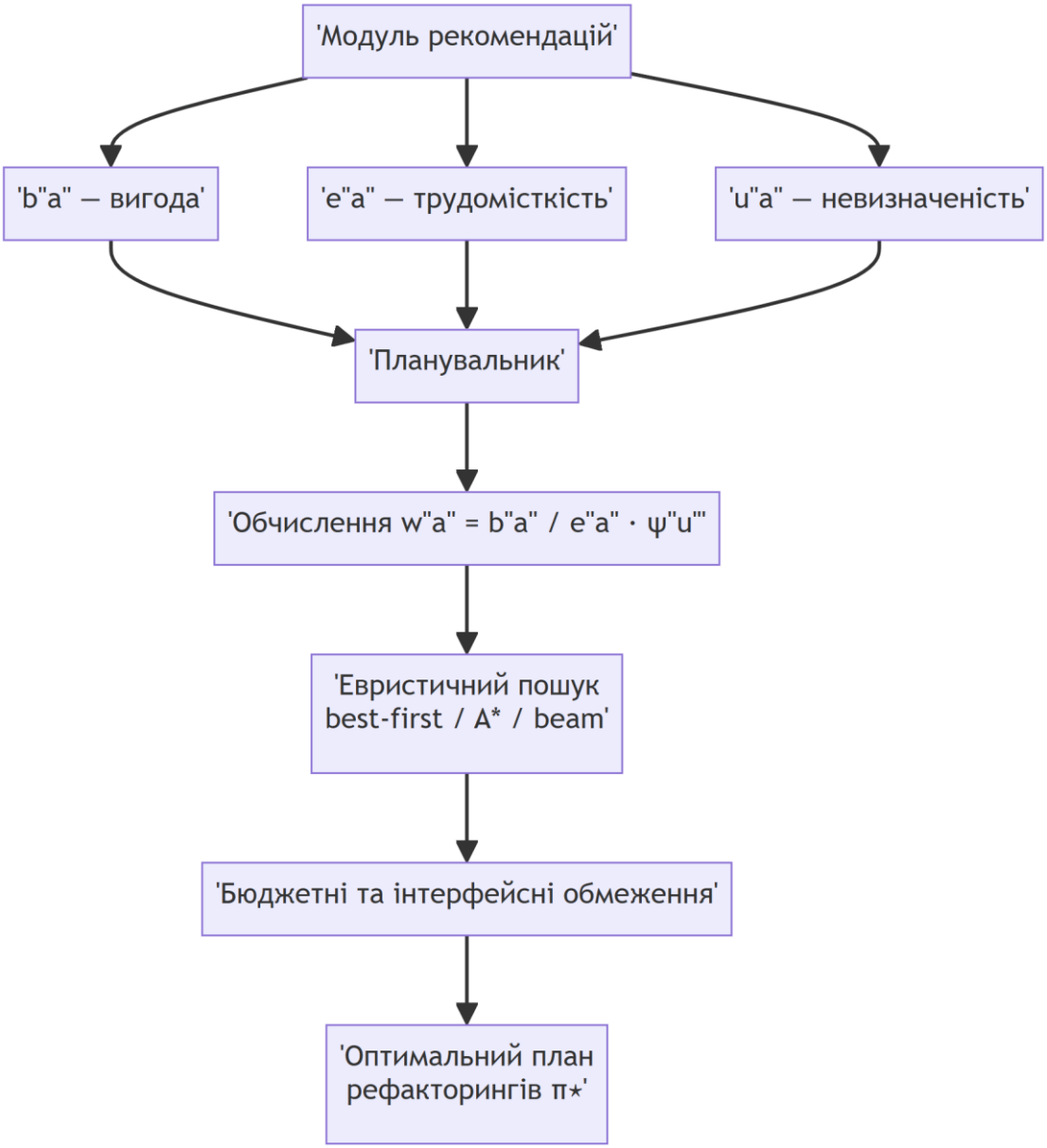

Рисунок 3.14 – Інтеграція виходів модуля рекомендацій у процедуру планування послідовностей рефакторингу

Тоді «узгоджена» корисність дії в планувальнику визначається як зважена комбінація:

$$U(a) = \lambda_{\text{loc}} \cdot \Delta Q_{\text{loc}}(a) + \lambda_{\text{glob}} \cdot \Delta Q_{\text{glob}}(a), \tag{3.94}$$

де $\lambda_{\text{loc}}, \lambda_{\text{glob}} \geq 0,\ \lambda_{\text{loc}} + \lambda_{\text{glob}} = 1$ – задаються політикою проєкту (наприклад, короткостроковий спринт із фокусом на «гарячих» клонних кластерах або довгострокове зменшення архітектурного боргу [173]).

У найпростішому випадку планувальник замінює $b_{\text{loc}}(a)$ у формулі для $w(a)$ на $U(a)$, що автоматично зсуває пріоритет у бік дій, які одночасно дають виграш і локально, і на рівні компонентів/проєкту.

Для клонів це особливо важливо – усунення окремого клонного кластеру може бути локально вигідним, але якщо він розташований у малоактивному модулі з низьким ризиком дефектів, глобальна корисність $\Delta Q_{\text{glob}}(a)$ може бути малою. Навпаки, операції рефакторингу, які зачіпають «гарячі» компоненти з високою історичною дефектністю й інтенсивною еволюцією, отримують більшу вагу через ΔQ_{comp} та ΔQ_{sys} [174].

Для наочності відповідні коефіцієнти $\lambda_{\text{loc}}, \lambda_{\text{glob}}$ та приклади різниці між локальними і глобальними оцінками зведені в таблиці 3.9, яка демонструє, як змінюється ранжування дій при переході від чисто локального пріоритезування до інтегрованого.

У підсумку інтеграція виходів модуля рекомендацій в планувальник зводиться до двох ключових кроків:

1) перетворення прогнозів вигоди/трудомісткості/невизначеності у придатні для пошуку ваги та пріоритети дій;

2) проєкції локальних ефектів (усунення клонів, спрощення методів тощо) на ієрархію глобальних цілей, що забезпечує узгодженість побудованих планів зі стратегією еволюції програмної системи [174, 175].

Таблиця 3.9 – Вплив узгодження локальних і глобальних цілей на пріоритизацію операцій рефакторингу

Операція рефакторингу a	$b_{loc}(a)$	$\Delta Q_{glob}(a)$	$U(a)$ при $\lambda_{loc} = 0{,}3$, $\lambda_{glob} = 0{,}7$	Позиція у ранжуванні (локальна)	Позиція у ранжуванні (глобальна)
Усунення клонів у утилітному класі	0,85	0,20	0,395	1	5
Extract Method у бізнес-логіці сервісу	0,70	0,75	0,735	2	1
Move Method між ядром і API	0,55	0,80	0,725	4	2
Рефакторинг клонів у тестовому коді	0,65	0,30	0,405	3	4
Спрощення умов у «гарячому» модулі	0,40	0,70	0,610	5	3

3.3.7 Експериментальне дослідження моделі планування послідовностей

Метою експериментального дослідження є оцінювання запропонованої моделі планування послідовностей рефакторингу за якістю отриманих планів, їх інвазивністю та впливом на стабільність CI/CD-процесів, а також аналіз поведінки моделі в зонах високої невизначеності прогнозів.

Опис репозиторіїв, сценаріїв симуляції та протоколу оцінювання. Для оцінювання моделі використано підмножину багатомодульних JVM-репозиторіїв з активною історією комітів та налаштованими CI/CD-пайплайнами (GitHub Actions, GitLab CI, Jenkins). Для кожного репозиторію r будується хронологічна послідовність снапшотів

$$\{S_{r,1}, S_{r,2}, \dots, S_{r,T_r}\}, \tag{3.95}$$

де $S_{r,t}$ – відповідає стану коду наприкінці спринту t.

Для кожного стану обчислюються:

– метрики внутрішньої якості (клонованість, структурні метрики, smell-індекси);

– події CI/CD (успішні/помилкові збірки, тривалість пайплайнів).

Симуляція планів рефакторингу виконується офлайново над історичними снапшотами. Для кожного спринту t формується множина кандидатних операцій $\mathcal{A}_{r,t}$ та план $\pi_{r,t}$, згенерований обраною політикою (запропонована модель або базова стратегія). План обмежується бюджетом:

$$\sum_{a \in \pi_{r,t}} e(a) \leq B_t, \tag{3.96}$$

де $e(a)$ – оцінка зусиль для операції a;

B_t – бюджет рефакторингу спринту.

Застосування плану до $S_{r,t}$ дає синтетичний стан $S'_{r,t}$, для якого повторно обчислюються метрики якості.

Агрегована зміна якості для спринту визначається як

$$\Delta Q_{r,t} = Q(S_{r,t}) - Q(S'_{r,t}), \tag{3.97}$$

де $Q(\cdot)$ – нормалізована композиція метрик підтримуваності згідно з ISO/IEC 25010 (підтримуваність, модифікованість, аналізованість) [176].

Інвазивність плану вимірюється показниками:

$$I_{r,t} = \alpha \cdot \frac{\Delta\mathrm{LOC}_{r,t}}{\mathrm{LOC}_{r,t}} + (1-\alpha) \cdot \frac{\mathrm{filesTouched}_{r,t}}{\mathrm{filesTotal}_{r,t}}, \tag{3.98}$$

де $\alpha \in [0,1]$ визначає відносну важливість змін рядків коду та кількості модифікованих файлів.

Вплив на CI/CD-стабільність оцінюється через відносну зміну частки невдалих збірок:

$$\Delta F_{r,t} = F'_{r,t} - F_{r,t}, \tag{3.99}$$

де $F_{r,t}$та $F'_{r,t}$ – частка неуспішних пайплайнів до і після застосування плану відповідно

Характеристики репозиторіїв, використаних у експериментальному дослідженні, зведено у таблицю 3.10. Подані проєкти відрізняються за архітектурним стилем, масштабом та інтенсивністю еволюції, що дає змогу оцінити роботу моделі планування в різних практичних сценаріях – від компактного ядра монолітної системи до розподіленої мікросервісної платформи та спадкових корпоративних рішень.

Таблиця 3.10 – Характеристики репозиторіїв для планів рефакторингу

Репозиторій	Кількість модулів	Середній розмір коду (LOC)	Тривалість історії (спринтів)	Середня кількість комітів за спринт	Тип CI/CD
Repo-A (корпоративний сервіс)	18	145 000	42	95	GitHub Actions
Repo-B (мікросервісна архітектура)	26	210 000	36	120	GitLab CI
Repo-C (ядро монолітної системи)	9	98 000	55	60	Jenkins
Repo-D (обробка даних)	14	165 000	31	80	GitHub Actions
Repo-E (спадкова платформа)	32	32000	72	70	Jenkins

Базові політики для порівняння. Для обґрунтування внеску запропонованої моделі планування послідовностей порівнюються такі політики:

– жадібна політика – на кожному кроці спринту обирається дія

$$a^* = \arg\max_{a \in \mathcal{A}_{r,t}} \frac{b_{\text{loc}}(a)}{e(a)}, \tag{3.100}$$

доки не буде вичерпано бюджет B_t. Вона використовує локальні оцінки benefit/effort, але ігнорує довгострокову взаємодію між діями (відсутній пошук у просторі планів);

– політика на базі правил – набір вручну заданих правил, що імітують типову практику інженерів: пріоритет клонів у «гарячих» модулях, обмеження максимальної кількості файлів, які можна змінювати в одному спринті, заборона рефакторингів у модулях з частими релізами [177];

– випадкова політика – випадковий вибір послідовності дій $\pi_{r,t}$, яка задовольняє бюджетним та інтерфейсним обмеженням. Ця політика є контрольним варіантом, що показує, який виграш можна отримати «випадково», без моделей;

– «без планування» – базова лінія «без втручання», яка відповідає сценарію, коли детектор антипатернів працює лише як аналітичний інструмент, але плани рефакторингу не формуються.

Усі політики працюють над однаковим простором дій та обмежень, різняться лише стратегією вибору послідовностей, що унеможливлює змішування ефектів «кращих можливостей» із ефектами планувальника.

Результати за метриками якості, інвазивності та стабільності CI/CD. Для кожної політики обчислюються усереднені показники по всіх репозиторіях та спринтах:

$$\Delta\bar{Q} = \frac{1}{N}\sum_{r,t} \Delta\, Q_{r,t}, \qquad \bar{I} = \frac{1}{N}\sum_{r,t} I_{r,t}\,, \qquad \Delta\bar{F} = \frac{1}{N}\sum_{r,t} \Delta\, F_{r,t}, \tag{3.101}$$

де N –загальна кількість (репозиторій, спринт)-спостережень.

Отримані результати показали, що запропонована модель планування послідовностей досягає більшого $\Delta\bar{Q}$ за порівняної або нижчої інвазивності $\bar{I}$, ніж жадібна політика або політика на базі правил. Випадкова політика очікувано демонструє високий розкид $\Delta Q_{r,t}$ та гірший середній результат, а «без планування» фактично фіксує вихідний рівень якості без покращення.

Стосовно CI/CD-стабільності, експерименти показали, що агресивні жадібні плани можуть збільшувати частку невдалих збірок ($\overline{\Delta F} > 0$), тоді як запропонований модуль планування, який враховує невизначеність та обмеження інтерфейсів, у більшості випадків зберігає або покращує стабільність пайплайнів

($\overline{\Delta F} \approx 0$ або $\overline{\Delta F} < 0$). Це узгоджується з емпіричними спостереженнями щодо важливості контрольованості змін для DevOps-процесів [178].

Узагальнені результати експериментального дослідження для різних політик планування наведено в таблиці 3.11. Як видно з отриманих даних, запропонована модель планування послідовностей забезпечує найбільший середній приріст якості $\Delta\bar{Q}$за одночасно найнижчої інвазивності змін $\bar{I}$серед усіх розглянутих стратегій, за винятком тривіальної політики без планування.

Таблиця 3.11 – Порівняння політик планування за якістю, інвазивністю та впливом на CI/CD

Політика планування	Середній приріст якості $\overline{\Delta Q}$	Середня інвазивність $\bar{I}$	Середня зміна частки невдалих збірок $\overline{\Delta F}$
Запропонована модель	0,084	0,036	−0,004
Жадібна (Greedy)	0,079	0,052	+0,012
Rule-based	0,061	0,041	+0,006
Випадкова (Random)	0,028	0,048	+0,015
Без планування	0,000	0,000	0,000

Візуальне порівняння політик планування за показниками приросту якості та інвазивності наведено на рисунку 3.15.

Аналіз поведінки в зонах високої невизначеності та випадків утримання від рекомендацій. Запропонована модель планування використовує оцінки невизначеності $u(a)$ для реалізації «обережної» поведінки: дії з високою невизначеністю можуть бути відкинуті або відкладені, навіть якщо очікувана корисність $b_{\text{loc}}(a)$ є великою. Формально для кожного кандидата aперевіряється умова

$$u(a) \leq \tau_u, \tag{3.102}$$

де τ_u – порогове значення, задане політикою ризику проєкту.

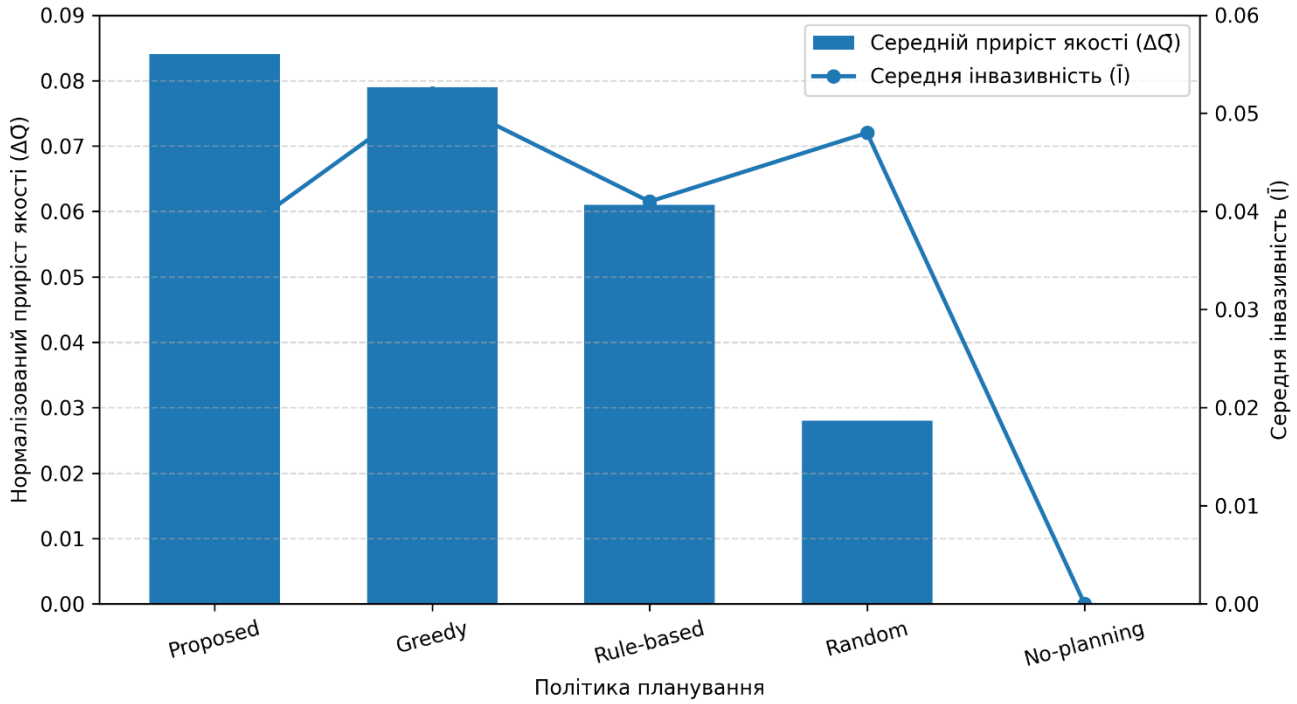

Рисунок 3.15 – Порівняння приросту якості та інвазивності для різних політик планування рефакторингів

Якщо для всіх дій із поточного фронту пошуку виконується $u(a) > \tau_u$, модуль планування повертає порожній або скорочений план («утримання від рекомендацій»).

Для аналізу поведінки в зонах високої невизначеності використано показники *покриття* (coverage) та *ризику* (risk), аналогічні тим, що застосовуються в задачах селективної класифікації [179]:

$$\text{coverage} = \frac{\text{кількість застосованих дій}}{\text{кількість усіх потенційних дій}}, \tag{3.103}$$

$$\text{risk} = \Pr\left(\Delta Q_{r,t} < 0 \mid a \in \pi_{r,t}\right), \tag{3.104}$$

тобто ймовірність негативного ефекту серед реально виконаних рекомендацій.

Експериментально показано, що із зростанням порогу τ_u *покриття* збільшується, але *ризик* зростає значно швидше, тоді як для помірних значень τ_u

досягається компроміс із низьким *ризиком* і прийнятною часткою *покриття*. Саме в цьому режимі запропонований модуль планування демонструє найкраще співвідношення «якість – стабільність» у порівнянні з жадібною політикою, яка не враховує невизначеність і має вищий ризик «шкідливих» планів [180].

Залежність між часткою застосованих рекомендацій та ризиком негативного ефекту для різних політик планування наведено на рисунку 3.16. Як видно з кривих risk–coverage, запропонована модель планування демонструє більш обережну поведінку в зонах високої невизначеності: за однакового рівня покриття ризик негативних наслідків залишається суттєво нижчим, ніж у жадібної політики.

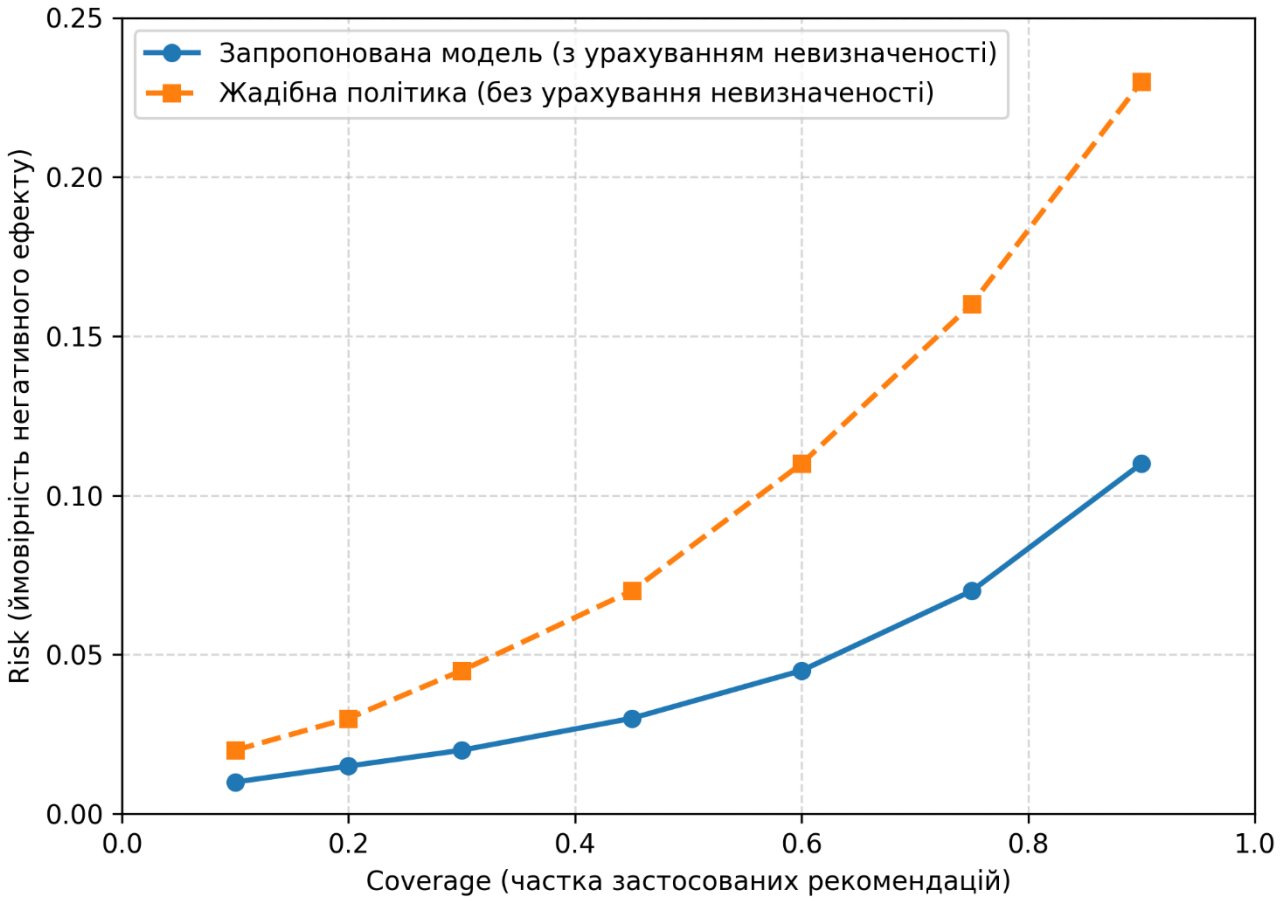

Рисунок 3.16 – Залежність ризику негативних ефектів від частки застосованих рекомендацій у зонах різної невизначеності

Аналіз конкретних випадків утримання від рекомендацій показав, що більшість із них пов’язані з конфліктними сигналами між різними групами ознак (структурні метрики, історія дефектів, еволюційні характеристики) та високою

нестабільністю CI/CD-пайплайнів у відповідних модулях промислової програмної системи.

В таких ситуаціях консервативне «нічого не робити» виявляється узгодженим із принципами безпечної автоматизації змін у DevOps-практиках [181, 182].

3.4 Модель процесного оцінювання ефективності рефакторингів на основі статистичного контролю процесів

3.4.1 Потреба в процесній оцінці ефективності рефакторингів

Рефакторинг у сучасних CI/DevOps-процесах зазвичай виконується інкрементально (серіями дрібних змін), часто паралельно з функціональними доопрацюваннями, виправленням дефектів і змінами конфігурацій конвеєра. За таких умов вплив рефакторингу на якість коду та стабільність процесу рідко проявляється як «чистий» ефект одного коміту або одного модуля; натомість він накладається на фонову варіативність, сезонність релізів та випадкові збурення інфраструктури.

Саме тому разові порівняння «до/після» для окремих комітів чи компонентів є обмеженими та не забезпечують надійного доказового висновку щодо того, чи стало краще (або гірше) внаслідок рефакторингу.

Найпростіший підхід «до/після» неявно зводиться до оцінювання різниці однієї (або кількох) метрик у двох точках часу.

Для метрики y_t (наприклад, середній час проходження збірки або значення метрики підтримуваності) після рефакторингового коміту в момент t_0 така оцінка може бути записана як:

$$\Delta y = y_{t_0+1} - y_{t_0}. \tag{3.105}$$

Однак Δy у реальних репозиторіях часто є нестійкою через високу дисперсію вимірювань, автокореляцію часових рядів та вплив супутніх змін. Тому навіть при реальному покращенні структурної якості коду короткостроково може спостерігатися тимчасове погіршення процесних показників (наприклад, через перегенерацію артефактів, зміну тестів чи перебудову залежностей).

У термінах статистичного контролю процесів це означає, що в даних одночасно присутні звичайні причини варіації (common causes) та спеціальні причини (assignable causes), які потрібно розрізняти, а не змішувати в одному «до/після»-порівнянні [183, 184].

Крім того, рефакторинг цікавить не лише як локальна зміна метрик коду, а як керована інтервенція в інженерний процес, що має бути узгоджена з ризиками CI, бюджетами часу/зусиль і політиками стабільності конвеєра. Виходячи з цього, доцільним є перехід від «точкового» оцінювання до процесного моніторингу: аналізу трендів, зсувів середніх, змін дисперсії та частоти відмов у часі.

Статистичний контроль процесів (SPC) надає формальний апарат для такого моніторингу через контрольні карти та правила виявлення нестабільності, що дозволяє оцінювати, чи змінився процес після застосування рефакторингових планів і чи ці зміни є статистично обґрунтованими [183, 184]. У межах запропонованої архітектури SPC-модуль розглядається як «замикаючий» компонент контуру якості, який доповнює моделі попередніх підрозділів. Зокрема, виходи детектора антипатернів (розділ 2) ініціюють формування кандидатів на усунення; далі модель, представлена в підрозділі 3.2, формує пропозиції змін (з урахуванням невизначеності/ризику), а модель, представлена в підрозділі 3.3, інтегрує ці пропозиції у виконувані плани рефакторингу. SPC-модуль, у свою чергу, виконує післядію: *підтверджує або спростовує корисність* реалізованих планів на основі поведінки часових рядів метрик та сигналів нестабільності, а також формує зворотний зв’язок для коригування порогів ризику й політик відбору в наступних ітераціях [184, 185].

Як показано на рисунку 3.17, SPC-модуль виконує роль механізму верифікації ефекту та стабілізації процесу через зворотний зв’язок.

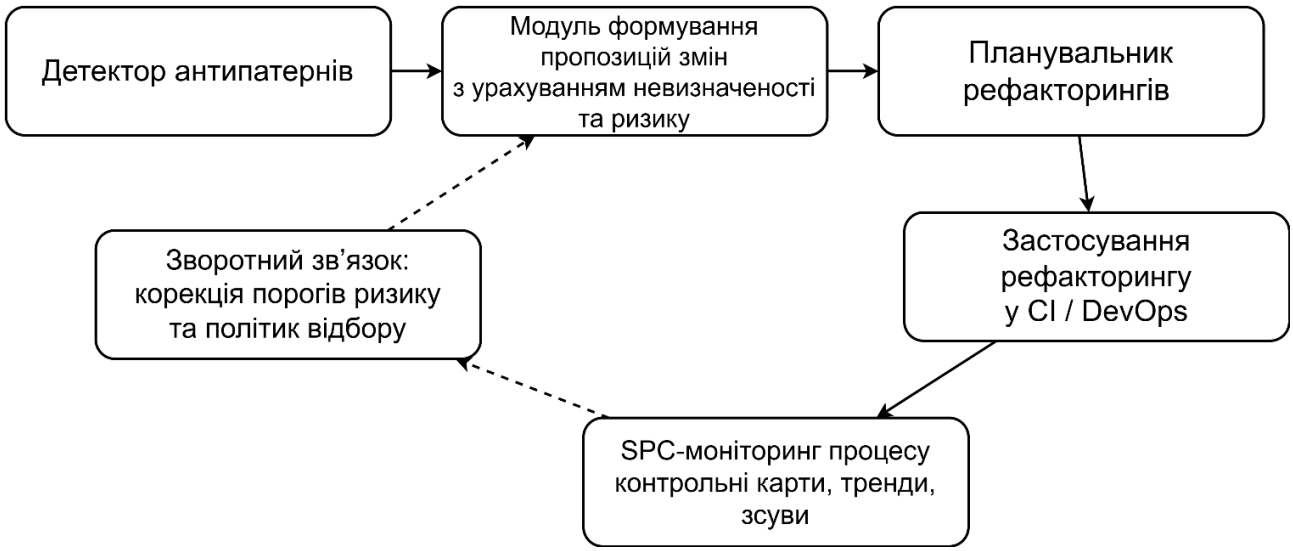

Рисунок 3.17 – Замкнений контур процесного оцінювання ефективності рефакторингів у CI/DevOps**.**

З практичної точки зору, процесна оцінка передбачає одночасний моніторинг двох класів показників:

– метрик внутрішньої якості коду (структурних/підтримуваності);

– метрик процесу CI/DevOps (надійність збірок, час виконання, частота відмов, флуктуації результатів).

Це дозволяє уникнути однобічних висновків, коли, наприклад, «покращення» коду супроводжується неприпустимим зростанням нестабільності конвеєра або ризику інтеграції. Відповідні групи показників, типові джерела даних та очікувані інтерпретації SPC-сигналів наведено в таблиці 3.12. Отже, потреба в процесній оцінці ефективності рефакторингів зумовлена тим, що:

– разові порівняння «до/після» не відокремлюють сигнал від шуму й не враховують динаміку процесу;

– рішення щодо «успішності» рефакторингу повинні базуватися на трендах і стабільності метрик у часі;

– SPC-модуль є необхідним для інтеграції результатів детектування і планування з об'єктивною післядією у CI/DevOps-контурі, забезпечуючи доказовість і керованість змін [183–185].

Таблиця 3.12 – Показники для процесного SPC-моніторингу ефективності рефакторингів у CI/DevOps

Група показників	Приклади метрик	Джерело у пайплайні	Інтерпретація SPC-сигналів
Метрики внутрішньої якості коду	Maintainability Index, Cyclomatic Complexity, LCOM, CBO, рівень дублювання	Статичний аналіз коду (SonarQube, PMD, Checkstyle)	Стійке зниження середнього рівня або дисперсії інтерпретується як позитивний ефект рефакторингу; поява нестабільності – ознака неконтрольованих змін
Метрики складності та розміру змін	LOC per commit, кількість змінених файлів, fan-in/fan-out	VCS (Git)	Різкі зсуви або сплески можуть вказувати на надмірну агрегацію змін або ризиковані рефакторингові пакети
Метрики тестування	Частка пройдених тестів, покриття, кількість збоїв тестів	Test stage CI	Поява SPC-сигналів нестабільності після рефакторингу може свідчити про порушення тестової ізоляції або приховані дефекти
Процесні метрики CI	Час виконання pipeline, час очікування в черзі, тривалість збірки	CI-оркестратор (GitHub Actions, GitLab CI, Jenkins)	Систематичний ріст середнього часу або дисперсії означає негативний вплив рефакторингу на продуктивність конвеєра
Метрики надійності пайплайна	Частота падінь збірок, MTBF, частка перезапусків	CI-логування	Сигнали спеціальних причин вказують на деградацію стабільності після змін, навіть за покращення метрик коду
Метрики flaky-поведінки	Частка flaky-тестів, повторні запуски	CI / Test infrastructure	Зростання нестабільності свідчить про побічні ефекти рефакторингу або недостатню декомпозицію
Контекстні еволюційні метрики	Code churn, ко-зміни, частота редагувань	VCS + аналітика репозиторію	Використовуються для розрізнення ефекту рефакторингу та фонової активності розробки

3.4.2 Формальна постановка задачі SPC-оцінювання ефективності планів рефакторингу (пропонована модель)

Для процесного оцінювання ефективності рефакторингів у часовому вимірі вводиться формальна модель, що поєднує:

– дискретний часовий індекс, синхронізований із подіями розробки (коміти/білди/релізи);

– множину вимірюваних метрик якості та DevOps-процесу;

– подання планів рефакторингу як інтервенцій із відомими моментами застосування;

– SPC-процедури моніторингу з контрольованою ймовірністю хибної тривоги.

Метою є отримання для кожного плану рефакторингу рішення типу «стало краще / стало гірше / немає достатніх доказів» із заданим рівнем довіри.

Позначення та структура спостережень. Нехай $M = \{m_1, \dots, m_{|M|}\}$ – множина модулів (пакетів/компонентів/підсистем) програмного проєкту. Час задається дискретним індексом $t \in \{1, \dots, T\}$, де кожне t відповідає фіксованій події вимірювання (наприклад, завершеному білду в CI або коміту, що пройшов пайплайн). Для кожного модуля m та моменту часу t спостерігається вектор метрик:

$$y_{m,t} = \left(y_{m,t}^{(1)}, \dots, y_{m,t}^{(K)}\right)^{\mathrm{T}}, \qquad y_{m,t} \in \mathbb{R}^K. \tag{3.106}$$

Компоненти $y_{m,t}^{(k)}$ включають метрики якості (наприклад, підтримуваність, складність, зв’язаність) та процесні метрики (наприклад, частка збоїв білду, тривалість пайплайна, флейкі-тести), що відповідає ідеї безперервного контролю якості в DevOps-контексті [186]. Для планово-орієнтованого оцінювання розглядається агрегований по цільовій множині модулів вектор:

$$y_{G,t} = \mathrm{Agg}\{y_{m,t} : m \in G\}, \qquad G \subseteq M, \tag{3.107}$$

де $\mathrm{Agg}(\cdot)$ – робастна агрегація (медіана/усічене середнє/зважене середнє), щоб зменшити вплив «викидів» на рівні окремих модулів

Таблиця 3.13 формалізує позначення та параметри процесної SPC-моделі, що використовується для оцінювання ефективності планів рефакторингу як керованих інтервенцій у CI/DevOps-процесі.

Таблиця 3.13 – Основні позначення процесної моделі (SPC-оцінювання)

Позначення	Опис
M	Множина модулів програмного проєкту (пакети, компоненти, підсистеми)
t	Дискретний часовий індекс, що відповідає події розробки, t = 1…T
$y_{m,t}$	Вектор спостережуваних метрик для модуля m у момент часу t
K	Кількість метрик у векторі ym,t (метрики якості коду та процесу CI/DevOps)
$G \subseteq M$	Підмножина модулів, на які спрямовано план рефакторингу
$y_{G,t}$	Агрегований вектор метрик для множини модулів G у момент часу t
$Agg(\cdot)$	Робастний оператор агрегації метрик (медіана, усічене або зважене середнє)
π	План рефакторингу (набір або послідовність змін у коді)
τ_π	Момент застосування плану рефакторингу π (інтервенція в процесі)
W_0	Часове вікно спостережень до інтервенції
W_1	Часове вікно спостережень після інтервенції
α	Рівень значущості SPC-контролю (ймовірність хибної тривоги), довіра дорівнює 1 − α

Подання планів рефакторингу як інтервенцій. Нехай $\Pi = \{\pi_1, \dots, \pi_P\}$ – множина розглянутих планів рефакторингу, сформованих у рекомендаційній підсистемі (підрозділи 3.2–3.3). Кожному плану π відповідає:

– цільова множина модулів $G_\pi \subseteq M$;

– момент застосування $\tau_\pi \in \{1, \dots, T\}$, відомий із історії (merge-commit, релізний тег, build-id);

– за потреби – тривалість «впровадження» $[\tau_\pi, \tau_\pi + d_\pi]$, якщо план реалізується серією комітів.

Інтервенція описується індикатором:

$$I_\pi(t) = \mathbb{1}[t \geq \tau_\pi], \tag{3.108}$$

який дозволяє формалізувати перехід від «in-control» режиму до потенційно зміненого режиму після застосування рефакторингів.

Як показано на рисунку 3.18, план рефакторингу моделюється як інтервенція в дискретному часовому ряді метрик із відомим моментом застосування τ_π. Для оцінювання ефекту виділяються вікна спостережень W_0 до інтервенції та W_1 після неї, що дозволяє аналізувати можливі зсуви середнього рівня або зміну варіативності показників у термінах статистичного контролю процесів.

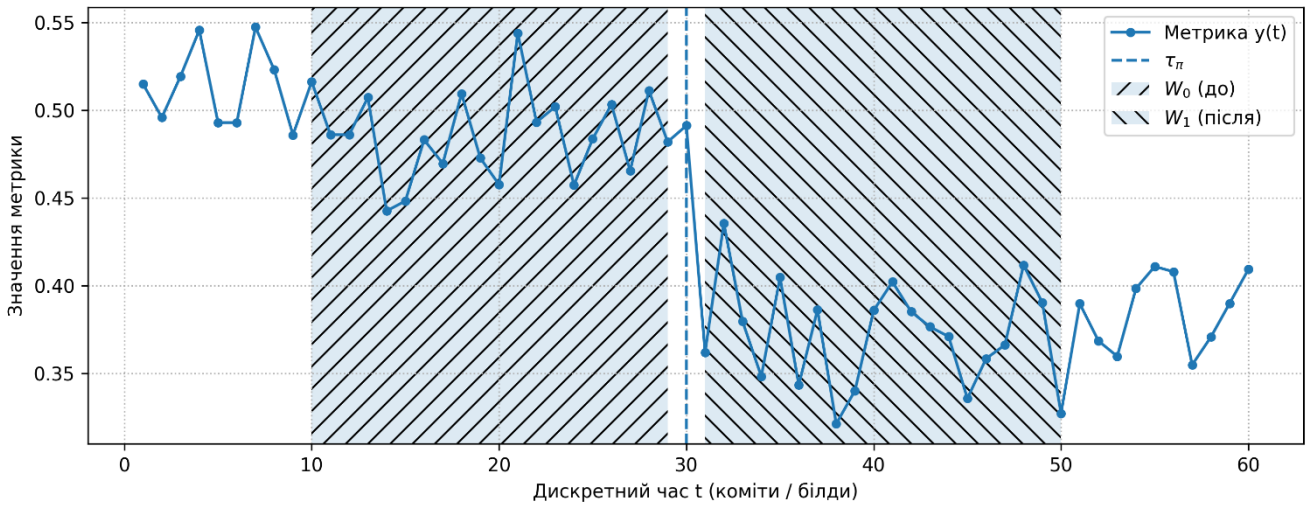

Рисунок 3.18 – Подання плану рефакторингу як інтервенції у дискретному часовому ряді метрик

Узгодження напрямків «краще/гірше» для різних метрик. Оскільки частина метрик має бажаний напрям «більше – краще» (наприклад, індекс підтримуваності), а частина – «менше – краще» (наприклад, частота падінь CI або тривалість пайплайна), вводиться уніфіковане перетворення до шкали «більше – краще».

Нехай $s_k \in \{+1, -1\}$ – ознака напряму для k-тої метрики. Тоді нормалізований сигнал:

$$\tilde{y}_{G,t}^{(k)} = s_k \cdot \frac{y_{G,t}^{(k)} - \mu_0^{(k)}}{\sigma_0^{(k)}}, \tag{3.109}$$

де $\mu_0^{(k)}, \sigma_0^{(k)}$ – оцінки середнього та масштабу в базовому («доінтервенційному») вікні $W_0 = \{t: \tau_\pi - L_0 \leq t < \tau_\pi\}$.

Перехід до стандартизованих величин важливий для сумісного аналізу різнорідних метрик та побудови багатовимірних SPC-статистик [187].

У таблиці 3.14 показано приклад узгодження напрямків для різнорідних метрик якості та процесу CI/DevOps шляхом введення коефіцієнта s_k, що дозволяє привести всі сигнали до уніфікованої інтерпретації «більше – краще» перед побудовою стандартизованих SPC-статистик.

Таблиця 3.14 – Приклад узгодження напрямків для типових метрик (шкала «більше – краще»)

Метрика	Приклад	Бажаний напрям	s_k
Підтримуваність	Maintainability Index	↑ (більше – краще)	+1
Складність	Cyclomatic Complexity	↓ (менше – краще)	−1
Збої CI	Частка build_failed	↓ (менше – краще)	−1
Час пайплайна	Build duration, сек	↓ (менше – краще)	−1
Дефектність	Ймовірність дефекту	↓ (менше – краще)	−1

SPC-статистика та контроль рівня довіри. Для кожного плану π будується SPC-процедура, що моніторить вектор $\tilde{y}_{G_\pi,t}$ після моменту τ_π. У якості базової, але достатньо загальної конструкції вводиться багатовимірна EWMA-статистика (MEWMA), яка є стандартним інструментом SPC для виявлення малих і середніх зсувів у багатьох координатах одночасно [188]:

$$z_t = \lambda\, \tilde{y}_{G_\pi,t} + (1-\lambda)\, z_{t-1}, \qquad z_{\tau_\pi - 1} = 0, \qquad 0 < \lambda \leq 1. \tag{3.110}$$

Далі визначається контрольна статистика (квадратична форма):

$$T_t^2 = z_t^{\top} \Sigma_z^{-1} z_t, \tag{3.111}$$

де Σ_z – коваріаційна матриця EWMA-вектора в «in-control» режимі, оцінена за даними з вікна W_0.

Правило сигналу SPC задається порогом h_α:

$$T_t^2 > h_\alpha \Rightarrow \text{виявлено зсув (out-of-control) на рівні довіри } 1-\alpha. \tag{3.112}$$

Параметр h_α обирається так, щоб забезпечити заданий рівень хибних тривог (еквівалентно – задану середню довжину серії без хибного сигналу ARL_0) [188, 189]. Таким чином, «рівень довіри» у постановці задачі конкретизується як контроль імовірності хибного виявлення ефекту плану π.

Як показано на рис. 3.19, оцінювання ефективності плану рефакторингу виконується у вигляді SPC-конвеєра, який об’єднує уніфікацію напрямків метрик, їх стандартизацію та багатовимірний EWMA-моніторинг.

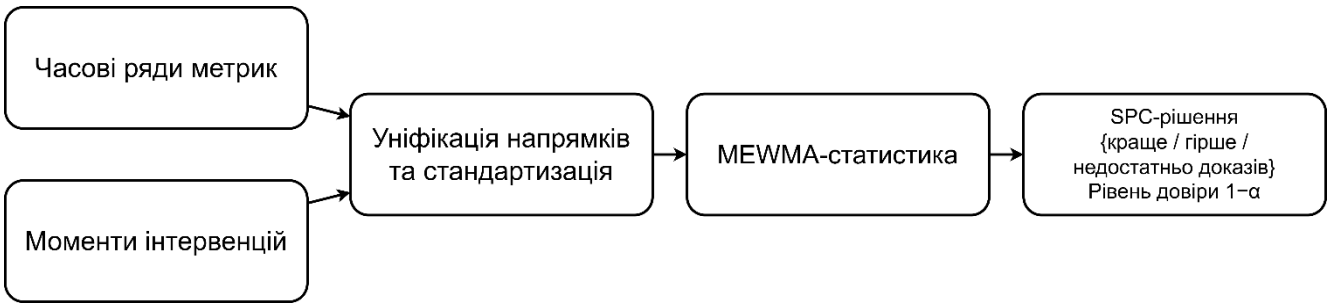

Рисунок 3.19 – SPC-конвеєр процесного оцінювання ефективності планів рефакторингу

Рішення для плану: «краще / гірше / недостатньо доказів». Вводиться відображення-розв’язувач:

$$D(\pi) \in \{+1, -1, 0\}, \tag{3.113}$$

де $D(\pi) = +1$ означає статистично підтверджене покращення після інтервенції;

$D(\pi) = -1$ – погіршення;

$D(\pi) = 0$ – відсутність достатніх доказів.

Рішення формується на основі двох компонент:

– факту виявлення зсуву SPC (перетин контрольних меж);

– напряму зсуву у «корисному» просторі.

Для напряму вводиться скалярний індекс покращення:

$$I_t = w^\top z_t, w \geq 0, \sum_k w_k = 1, \tag{3.114}$$

де w – ваги, які відображають пріоритети між метриками (якість vs процес).

Тоді практичне правило може бути сформульоване як:

– «стало краще» – існує $t \in W_1$ таке, що $T_t^2 > h_\alpha$ і $I_t > 0$ протягом щонайменше L послідовних точок (стійкий позитивний зсув);

– «стало гірше» – існує $t \in W_1$ таке, що $T_t^2 > h_\alpha$ і $I_t < 0$ протягом щонайменше L послідовних точок (стійкий негативний зсув);

– «немає достатніх доказів»: жодна з умов вище не виконується у пост-інтервенційному вікні $W_1 = \{t: \tau_\pi \leq t \leq \tau_\pi + L_1\}$.

Важливим є те, що така постановка узгоджується з логікою інтервенційного аналізу часових рядів: ефект рефакторингу розглядається як потенційний зсув розподілу (або тренду) метрик, а довіра рішення контролюється через параметр α та налаштування контрольних меж.

За потреби, коли часові ряди демонструють автокореляцію або сезонність, базовий SPC-рівень може доповнюватися інструментами перетворення/попереднього «вибілювання» або моделями перериваних часових рядів (Interrupted Time Series) для коректнішої оцінки ефектів інтервенцій [186, 187].

У підсумку, формальна постановка задачі SPC-оцінювання для плану π зводиться до: задано $\{y_{G_\pi,t}\}_{t=1}^{T}$, τ_π, параметри W_0, W_1, а також рівень довіри $1-\alpha$; необхідно обчислити $D(\pi) \in \{+1, -1, 0\}$ та, за можливості, надати інтерпретовані свідчення (які метрики дали внесок у сигнал) у вигляді траєкторій z_t і компоненти I_t.

3.4.3 Статистична модель еволюції показників якості та CI-процесу

Для коректного застосування SPC-підходів необхідно відокремити зміни, зумовлені інтервенціями (планами рефакторингу), від природної варіабельності процесу (noise) та регулярних коливань, пов'язаних із фазами релізного циклу. У цьому підрозділі формалізується статистична модель номінального (без інтервенцій) процесу для метрик якості та показників CI з урахуванням нормування, можливої автокореляції та процедур оцінювання базових параметрів, потрібних для побудови контрольних карт.

Нормування метрик за класами модулів і фазами релізного циклу. Нехай $m \in M$ – модуль (компонент), $t \in \{1, \dots, T\}$ – дискретний час, що відповідає комітам/білдам/релізам (використовується той часовий крок, на якому вимірюється метрика), $k \in \{1, \dots, K\}$ – індекс метрики. Позначимо «сире» значення метрики як $x_{m,k}(t)$.

Для зменшення змішування ефектів «розміру/ролі модуля» та «фази релізного циклу» вводяться:

– функція класу модуля $c(m) \in \mathcal{C}$ (наприклад: бібліотечний/сервісний модуль; шар архітектури; діапазон LOC; мова; наявність критичних залежностей);

– функція фази релізного циклу $\varphi(t) \in \Phi$ (наприклад: {розробка, стабілізація, реліз/гарячі виправлення}).

Тоді нормоване значення метрики визначається відносно «референтної» групи (c, φ) як стандартизований залишок:

$$z_{m,k}(t) = \frac{g_k(x_{m,k}(t)) - \mu_{c(m),\varphi(t),k}}{\sigma_{c(m),\varphi(t),k}}. \quad (3.115)$$

де $g_k(\cdot)$ – перетворення масштабу (за потреби);

$\mu_{c,\varphi,k}$, $\sigma_{c,\varphi,k}$ – базові параметри центру та розсіювання для метрики k у групі класу c та фази φ.

Для метрик із сильною асиметрією або «важкими хвостами» доцільно застосовувати лог-перетворення або перетворення типу Box–Cox перед стандартизацією, щоб наблизити розподіл до умов застосування класичних SPC-карт.

Окремо підкреслимо, що показники CI (наприклад, тривалість збірки, частка падінь, кількість flaky-тестів, кількість повторних запусків) часто мають різну природу (неперервні/лічильні/частотні). Тому функція $g_k(\cdot)$ і тип параметризації (μ, σ) мають визначатися з урахуванням типу метрики.

У таблиці 3.15 наведено приклади перетворень і схем нормування метрик якості коду та показників CI з урахуванням класів модулів і фаз релізного циклу

Модель номінального процесу без рефакторингів. Після нормування $z_{m,k}(t)$ розглядається як реалізація стаціонарного (або квазістаціонарного) процесу за відсутності інтервенцій.

Для багатьох метрик, що вимірюються послідовно у часі (особливо на рівні білдів/релізів), типовою є автокореляція: значення в момент tстатистично залежить від значення в момент $t-1$. Ігнорування автокореляції призводить до завищеної частоти хибних тривог на контрольних картах. Відомим підходом є попереднє моделювання часової структури та застосування SPC до залишків (residual charting) [190, 191].

Для фіксованих m, k модель номінального процесу можна задати як ARMA-процес:

$$z_{m,k}(t) = \alpha_{m,k} + \sum_{i=1}^{p} \phi_{i,m,k}\ z_{m,k}(t-i) + \varepsilon_{m,k}(t) + \sum_{j=1}^{q} \theta_{j,m,k}\ \varepsilon_{m,k}(t-j), \quad (3.116)$$

де $\varepsilon_{m,k}(t)$ – білий шум із $E[\varepsilon_{m,k}(t)] = 0, Var[\varepsilon_{m,k}(t)] = \sigma^2_{\varepsilon,m,k}$.

Таблиця 3.15 – Приклади нормування метрик якості та показників CI за класами модулів і фазами релізного циклу

Тип метрики	Приклад показника	Рекомендоване перетворення $g_k(\cdot)$	Нормування за (c, φ)	Коментар щодо інтерпретації
Розмір коду	LOC, кількість класів	$\log(x)$	За класом модуля (тип/шар)	Дозволяє порівнювати модулі різного масштабу без домінування великих компонентів
Структурна складність	Cyclomatic Complexity	$\log(x)$ або Box–Cox	За класом модуля	Зменшує вплив «важких хвостів» у складних модулях
Статичні попередження	Кількість warning'ів	$\sqrt{x}$	За класом модуля та фазою	Лічильні метрики; стабілізаційна фаза має інші базові рівні
Підтримуваність	Maintainability Index	Без перетворення	За класом модуля	Метрика вже нормована; важливо враховувати роль модуля
Тривалість CI-процесу	Build duration, сек	$\log(x)$	За фазою релізного циклу	Очікувані значення істотно відрізняються між фазами
Надійність CI	Частка *failed builds*	logit або Box–Cox	За фазою релізного циклу	Частотна метрика; у релізній фазі допустимі нижчі базові рівні
Нестабільність тестів	Частка *flaky tests*	$\sqrt{x}$ або logit	За фазою релізного циклу	Дає змогу відокремити тимчасову нестабільність від системної
Повторні запуски	Кількість reruns	$\sqrt{x}$	За фазою релізного циклу	Лічильна CI-метрика, чутлива до режимів навантаження

На практиці для інтеграції з SPC часто достатньо спрощеної AR(1)-моделі:

$$z_{m,k}(t) - \mu_{m,k} = \phi_{m,k}(z_{m,k}(t-1) - \mu_{m,k}) + \varepsilon_{m,k}(t), |\, \phi_{m,k} \,| < 1, \quad (3.117)$$

оскільки вона явно описує інерційність метрики та дозволяє ефективно «вибілювати» ряд для подальшого контролю [192].

Для показників CI автокореляція також є типовою (наприклад, тривалість білда або частота падінь може зростати в періоди стабілізації), тому застосування AR/ARMA-моделі до нормованих або трансформованих значень є методологічно виправданим [191]. Після оцінювання параметрів $\hat{\mu}_{m,k}, \hat{\phi}_{m,k}$ формуються оцінки «інновацій» (залишків):

$$\hat{\varepsilon}_{m,k}(t) = (z_{m,k}(t) - \hat{\mu}_{m,k}) - \hat{\phi}_{m,k}(z_{m,k}(t-1) - \hat{\mu}_{m,k}), \qquad (3.118)$$

і саме послідовність $\hat{\varepsilon}_{m,k}(t)$ використовується як вхід для класичних SPC-карт (Shewhart/EWMA/CUSUM). Такий підхід зменшує вплив автокореляції на статистики контролю та стабілізує рівень помилкових спрацювань [191, 192]. Як показано на рисунку 3.20, перед застосуванням SPC-карт часові ряди метрик проходять послідовність перетворень, що включає масштабування, нормування за класами модулів і фазами релізного циклу, а також моделювання автокореляції за допомогою AR/ARMA-моделі.

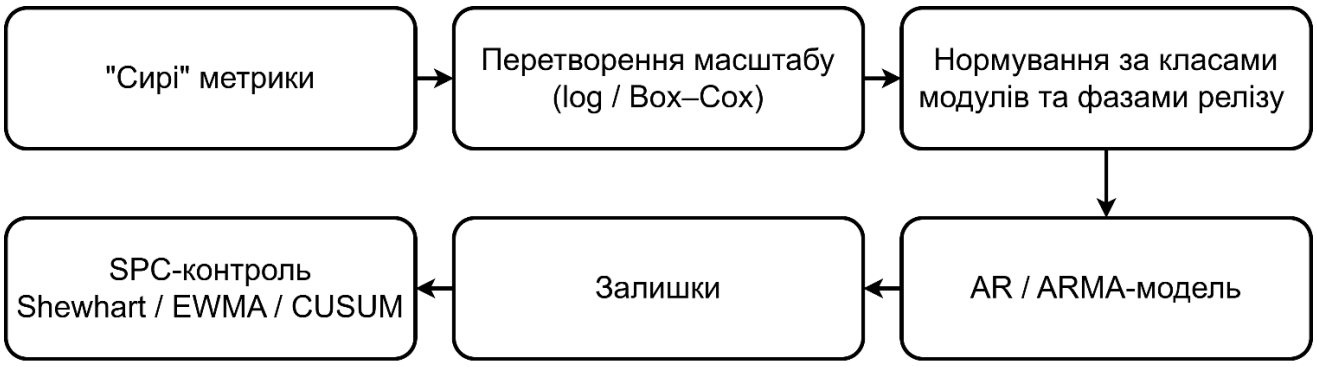

Рисунок 3.20 – Узагальнена схема побудови ін-контрольної моделі та формування залишків для SPC-оцінювання

Визначення базових параметрів для побудови SPC-карт. Параметри, необхідні для SPC-карт, визначаються на етапі навчання та калібрування на відрізках історії, де відсутні інтервенції або їх вплив вважається мінімальним. Для кожної групи (c, φ) та метрики k оцінюються:

– параметри нормування $\mu_{c,\varphi,k}$, $\sigma_{c,\varphi,k}$;

– параметри часової залежності (наприклад, $\phi_{m,k}$ або набір $\{\phi_i, \theta_j\}$);

– параметри розсіювання залишків $\sigma_{\varepsilon,m,k}$, що безпосередньо входять у межі контролю.

Для карт індивідуальних спостережень (I-chart) або Shewhart-карт над залишками типовим є правило:

$$UCL = \hat{\mu}_\varepsilon + L\hat{\sigma}_\varepsilon, CL = \hat{\mu}_\varepsilon, LCL = \hat{\mu}_\varepsilon - L\hat{\sigma}_\varepsilon, \tag{3.119}$$

де за класичними припущеннями беруть $L = 3$. Для EWMA-карти над залишками (як більш чутливої до малих зсувів) базові параметри включають коефіцієнт згладжування $\lambda \in (0,1]$ та стаціонарну дисперсію EWMA-статистики:

$$s(t) = \lambda\,\hat{\varepsilon}(t) + (1-\lambda)s(t-1), \qquad Var[s(t)] \approx \frac{\lambda}{2-\lambda}\,\hat{\sigma}_\varepsilon^2, \tag{3.120}$$

а контрольні межі визначаються як $CL \pm L\sqrt{Var[s(t)]}$ (з урахуванням початкових умов). У випадку CI-метрик, де важливо швидко виявляти деградацію (наприклад, зростання частоти падінь білда), EWMA/CUSUM-підходи є практично доцільними, оскільки реагують на стійкі невеликі зміни раніше за Shewhart-карти [192]. Щоб зробити параметризацію придатною для подальшої інтеграції з інтервенційною моделлю (підрозд. 3.4.2), доцільно зберігати базові оцінки у вигляді структурованого набору:

$$\Theta_{c,\varphi,k} = \{\mu_{c,\varphi,k}, \sigma_{c,\varphi,k}, \phi_{c,\varphi,k}, \sigma_{\varepsilon,c,\varphi,k}, (p,q)\}, \tag{3.121}$$

де (p,q) – порядок ARMA (за потреби).

Надалі, при аналізі впливу плану рефакторингу, статистики SPC обчислюються відносно параметрів Θ, що відповідають класу модуля та фазі релізу в момент спостереження, тобто процес контролю узгоджується з контекстом експлуатації. Як показано на рисунку 3.21, нормовані метрики якості та CI-процесу

часто характеризуються помітною автокореляцією, що порушує припущення класичних SPC-карт і призводить до завищеної частоти хибних тривог.

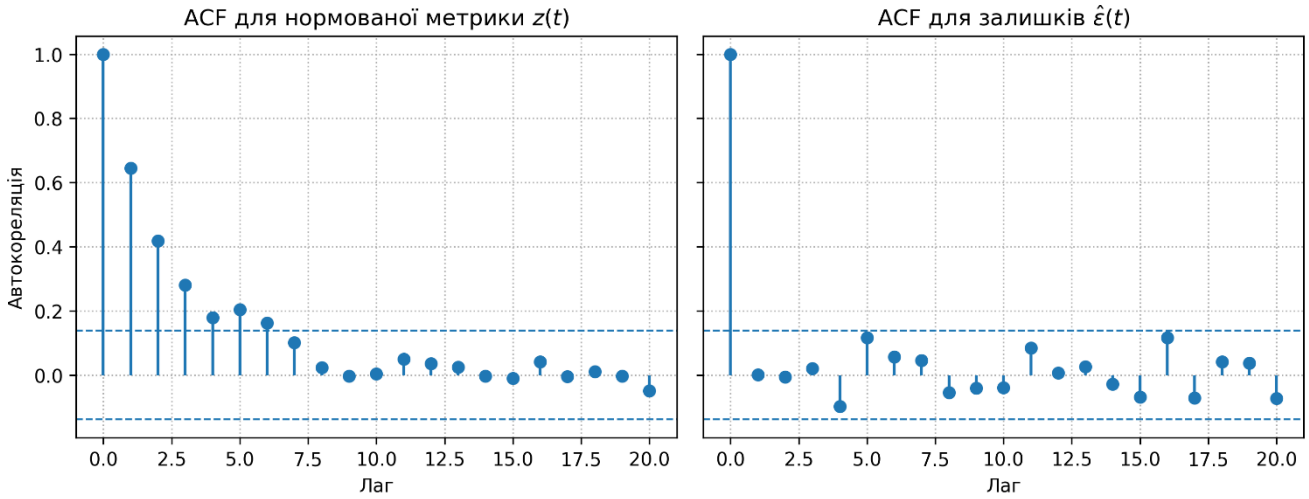

Рисунок 3.21 – Приклад впливу автокореляції на метрику та результат «prewhitening» через AR-модель перед застосуванням SPC

Таким чином, запропонована статистична модель еволюції показників якості та CI-процесу задає узгоджену процедуру:

– порівнюваності через нормування за (c, φ);

– коректного номінального опису часової залежності;

– отримання залишків, придатних для SPC-карт із передбачуваними рівнями помилкових спрацювань.

Це створює основу для наступного кроку – оцінювання ефекту інтервенцій (планів рефакторингу) як статистично значущих зсувів/трендів відносно ін-контрольної моделі з заданим рівнем довіри.

3.4.4 SPC-модель контролю: побудова EWMA та CUSUM-карт і функцій сигналу

Після побудови номінальної (baseline) моделі та отримання рядів залишків $\hat{\varepsilon}_{m,k}(t)$ (підрозд. 3.4.3), задача SPC-контролю зводиться до виявлення статистично

значущих зсувів у середньому рівні або тренді метрик якості й показників CI-процесу. У контексті рефакторингів важливими є саме малі, але стійкі зміни, які не завжди проявляються як одиничні «викиди». Тому базовими інструментами обираються EWMA- та CUSUM-карти, що мають підвищену чутливість до таких зсувів [193, 194].

EWMA-карта для контролю малих зсувів. Нехай $r_{m,k}(t)$ – ряд, до якого застосовується SPC-контроль (типово $r_{m,k}(t) = \hat{\varepsilon}_{m,k}(t)$ або нормоване значення $z_{m,k}(t)$, якщо автокореляція неістотна). EWMA-статистика визначається рекурентно:

$$w_{m,k}(t) = \lambda\, r_{m,k}(t) + (1-\lambda)\, w_{m,k}(t-1), 0 < \lambda \leq 1, \tag{3.122}$$

де λ – задає «пам'ять» карти – менші λ роблять карту більш інерційною, але чутливішою до малих стійких зсувів [195].

За умови $E[r_{m,k}(t)] = 0$ в номінальному режимі, центральна лінія $CL = 0$, а дисперсія EWMA-статистики у момент tнаближається як:

$$Var(w_{m,k}(t)) = \sigma^2_{r,m,k}\, \frac{\lambda}{2-\lambda}\, (1-(1-\lambda)^{2t}), \tag{3.123}$$

де $\sigma^2_{r,m,k}$ – дисперсія контрольованого ряду (для залишків – $\sigma^2_{\varepsilon,m,k}$).

Тоді фіксовані контрольні межі (для достатньо великих t) задаються:

$$UCL_{m,k} = L\,\sigma_{r,m,k}\sqrt{\frac{\lambda}{2-\lambda}}, LCL_{m,k} = -L\,\sigma_{r,m,k}\sqrt{\frac{\lambda}{2-\lambda}}. \tag{3.124}$$

Параметр L встановлює рівень хибних тривог (через середню довжину серії, ARL) і на практиці підбирається спільно з λпід цільову чутливість [195].

CUSUM-карта як детектор зсуву середнього. CUSUM-підхід розглядає накопичення малих відхилень у часі. Для двостороннього контролю вводяться позитивна та негативна статистики:

$$C^{+}_{m,k}(t) = \max\,(0, C^{+}_{m,k}(t-1) + r_{m,k}(t) - K), \tag{3.125}$$

$$C^{-}_{m,k}(t) = \max\,(0, C^{-}_{m,k}(t-1) - r_{m,k}(t) - K), \tag{3.126}$$

де K – «порогова» константа (reference value), що задає масштаб зсуву, на який карта має реагувати найшвидше.

Сигнал формується, якщо

$$C^{+}_{m,k}(t) \geq H \;\text{ або }\; C^{-}_{m,k}(t) \geq H, \tag{3.127}$$

де H – інтервал прийняття рішення, який визначає компроміс між чутливістю та частотою хибних спрацювань [196, 197]

Для метрик CI (наприклад, тривалість білда чи частка падінь) CUSUM зручний тим, що може бути налаштований як «односторонній» детектор деградації (контроль лише зростання ризику/часу), зменшуючи зайві повідомлення. Як показано на рисунку 3.22, EWMA- та CUSUM-карти по-різному реагують на стійкі малі зсуви у контрольованому ряді.

EWMA-карта забезпечує згладжене відстеження тренду та добре підходить для довготривалого моніторингу, тоді як CUSUM-карта накопичує відхилення і, як правило, сигналізує про зсув раніше.

Фіксовані та адаптивні контрольні межі з урахуванням класу модуля і фази релізу. У практичних DevOps-процесах дисперсія метрик суттєво залежить від контексту: модулі ядра системи можуть змінюватися рідше, але з вищим ризиком; у фазі стабілізації зростає частота CI-подій (фікси, повторні запуски тестів), що змінює варіабельність. Тому вводиться контекстна параметризація меж контролю через $(c(m), \varphi(t))$, аналогічно нормуванню у підрозділі 3.4.3.

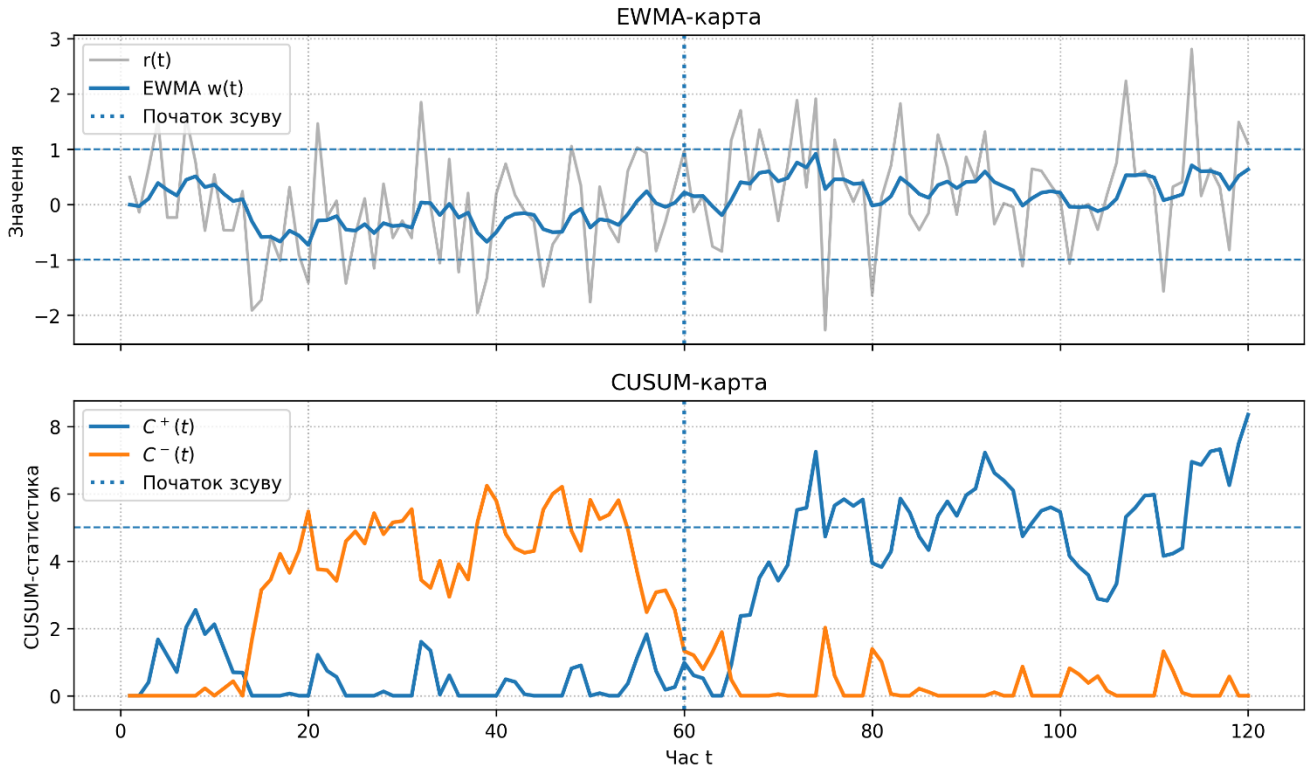

Рисунок 3.22 – Приклад виявлення стійкого зсуву: порівняння EWMA- та CUSUM-карт.

Нехай $\Theta_{c,\varphi,k}$ – набір базових параметрів для метрики k у класі c та фазі φ, який містить $\sigma_{r,c,\varphi,k}$(або $\sigma_{\varepsilon,c,\varphi,k}$). Тоді межі EWMA та параметри CUSUM задаються як:

$$UCL_{m,k}(t) = L_{c(m),\varphi(t),k}\,\sigma_{r,c(m),\varphi(t),k}\sqrt{\frac{\lambda_k}{2-\lambda_k}}, \tag{3.128}$$

$$K_{m,k}(t) = K_{c(m),\varphi(t),k}, \qquad H_{m,k}(t) = H_{c(m),\varphi(t),k}. \tag{3.129}$$

Це відповідає адаптивному SPC у сенсі контекстної зміни меж (без зміни самої статистики), що зменшує хибні тривоги в «шумних» фазах релізу і підсилює контроль там, де очікується стабільність [198]. Додатково може застосовуватися правило «замороження» (hold) періодів із аномальними зовнішніми впливами (масове оновлення залежностей, міграції інфраструктури), щоб не змішувати їх із

ефектом рефакторингів у SPC-оцінюванні. У таблиці 3.16 наведено типові налаштування EWMA- та CUSUM-карт для різних класів метрик якості коду й CI-процесу.

Таблиця 3.16 – Типові налаштування EWMA та CUSUM для метрик якості й CI-процесу

Тип метрики	Приклад	EWMA: λ, L	CUSUM: K, H	Примітка
Підтримуваність / структурна якість	Maintainability Index, Cyclomatic Complexity	λ = 0,10–0,20; L = 2,7–3,0	K = 0,25–0,50; H = 4–6	Стійкі малі зсуви; доцільно посилювати контроль у стабільних фазах
Клон-щільність	Clone coverage, duplication ratio	λ = 0,15–0,25; L = 2,5–3,0	K = 0,30–0,60; H = 4–6	Чутлива до пакетних змін; бажано фільтрувати масові рефакторинги
Дефектність (JIT-оцінки)	Ймовірність дефекту, risk score	λ = 0,20–0,30; L = 2,5–3,0	K = 0,30–0,50; H = 5–7	Односторонній контроль деградації; важливий низький ARL для росту ризику
CI-тривалість	Build duration, queue time	λ = 0,20–0,30; L = 3,0–3,5	K = 0,50–1,00; H = 6–8	Висока варіабельність у фазі стабілізації; доцільні адаптивні межі
CI-failure rate	Частка failed builds	λ = 0,25–0,40; L = 3,0–3,5	K = 0,50–1,00; H = 6–9	Частотна метрика; рекомендується односторонній CUSUM
Нестабільність тестів	Частка flaky tests	λ = 0,20–0,35; L = 3,0–3,5	K = 0,40–0,80; H = 6–9	Чутлива до інфраструктурних збоїв; бажано застосовувати hold-правила

Логіка сигналу для окремих метрик та агрегування на рівні компонента/проєкту. Оскільки різні метрики мають різну «напрямленість якості», вводиться уніфікована функція орієнтації.

Нехай $\delta_k \in \{-1, +1\}$ – знак, що задає напрямок «покращення» для метрики k (наприклад, для ризику CI або складності $\delta_k = -1$, а для метрики підтримуваності $\delta_k = +1$). Тоді контроль здійснюється над величиною:

$$r'_{m,k}(t) = \delta_k \, r_{m,k}(t), \tag{3.130}$$

так що позитивний зсув $r'_{m,k}(t)$ інтерпретується як «рух у бік покращення», а негативний – як деградація.

Далі вводиться бінарний сигнал для метрики k у модулі m у момент t. Для EWMA:

$$s_{m,k}^{EWMA}(t) = \begin{cases} +1, & w_{m,k}(t) > UCL_{m,k}(t) \\ -1, & w_{m,k}(t) < LCL_{m,k}(t) \\ 0, & \text{інакше} \end{cases} \quad (3.131)$$

Аналогічно для CUSUM:

$$s_{m,k}^{CUSUM}(t) = \begin{cases} +1, & C_{m,k}^{+}(t) \geq H_{m,k}(t) \\ -1, & C_{m,k}^{-}(t) \geq H_{m,k}(t) \\ 0, & \text{інакше} \end{cases} \quad (3.132)$$

У практичній реалізації доцільно поєднувати EWMA та CUSUM як «комітет детекторів» (наприклад, сигнал вважається підтвердженим, якщо обидві карти узгоджено вказують на зсув, або якщо один із детекторів стабільно сигналізує протягом q точок). Це знижує чутливість до одиничних артефактів вимірювання, зокрема для CI-метрик, де можливі зовнішні флуктуації (черги раннерів, мережеві збої тощо).

Для агрегування на рівні модуля (компонента) вводиться зважений інтегральний сигнал:

$$S_m(t) = \text{sign}\left(\sum_{k=1}^{K} \omega_k \ \tilde{s}_{m,k}(t)\right), \quad (3.133)$$

де де $\omega_k \geq 0$ – ваги важливості метрик;

$\tilde{s}_{m,k}(t)$ – узгоджений сигнал по метриці (наприклад, $\tilde{s} = \max(s^{EWMA}, s^{CUSUM})$ за абсолютним значенням або більш консервативне правило перетину).

Значення $S_m(t) = +1$ трактується як «статистично значуще покращення», $S_m(t) = -1$ – як «статистично значуща деградація», $S_m(t) = 0$ – як відсутність достатніх доказів зсуву.

На рівні проєкту (або підсистеми) агрегування може виконуватися як частка модулів, що сигналізують деградацію/покращення, із вагами за критичністю модуля v_m:

$$S_{proj}(t) = \sum_{m \in M} v_m \ \mathbf{1}\{S_m(t) = -1\}, \sum_{m \in M} v_m = 1. \tag{3.134}$$

Таким чином, SPC-модуль формує як локальні сигнали (точково по метриці й модулю), так і узагальнені індикатори, придатні для використання у підрозділі 3.4.5 як критерії ефективності/ризику планів рефакторингу в дискретному часі (коміти/білди/релізи).

Як показано на рисунку 3.23, локальні SPC-сигнали, отримані для окремих метрик за допомогою EWMA- та CUSUM-карт, узгоджуються за напрямом і агрегуються на рівні модуля з урахуванням ваг важливості показників.

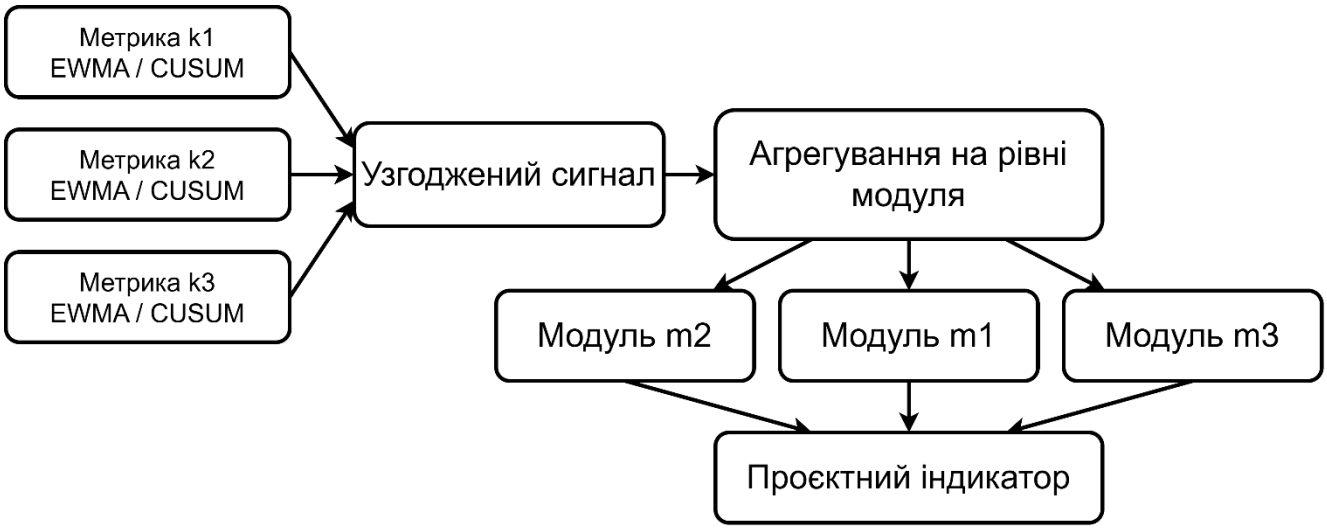

Рисунок 3.23 – Логіка формування локальних і агрегованих SPC-сигналів для метрик якості та CI

3.4.5 Модель атрибуції впливу окремих рефакторингів та послідовностей

Оцінювання ефективності рефакторингу у межах SPC-модуля фіксує факт статистично значущого зсуву показників якості або CI/DevOps-процесу, однак саме по собі не відповідає на питання причинності: *чи можна приписати спостережений зсув конкретному рефакторингу або плану рефакторингів, а не паралельним змінам (feature-комітам, оновленням залежностей, змінам тестів тощо)?* Тому вводиться модель атрибуції, яка узгоджує моменти застосування планів (підрозділ 3.3) з часовими рядами метрик та використовує спрощений причинно-статистичний механізм оцінювання ефекту.

Позначення та події-інтервенції. Нехай $M = \{m_1, \dots, m_{|M|}\}$ – множина модулів/компонентів. Дискретний час $t \in \mathcal{T}$ інтерпретується як індекс коміту, білду або релізу (залежно від рівня агрегації, обраного у 3.4.2). Для кожного модуля m та метрики k спостерігається часовий ряд:

$$y_{m,k}(t), k \in \{1, \dots, K\}. \tag{3.135}$$

Також визначаються коваріати (контрольні змінні) $X_m(t)$, які описують «контекст змін» у модулі в момент часу t: інтенсивність редагування, масштаб патча, зміни тестів тощо.

План рефакторингу π розглядається як множина інтервенцій із відомими моментами застосування:

$$\pi = \{ i_1, \dots, i_J \}, i_j = \langle m_j, a_j, t_j \rangle, \tag{3.136}$$

де m_j – модуль (або набір модулів);

a_j – тип дії (Extract Method, Pull Up тощо);

t_j – момент застосування (коміт/білд/реліз).

Узгодження вікон «до/після» з моментами планів. Для кожної інтервенції $i = \langle m_i, a_i, t_i \rangle$ вводиться симетрія «локального експерименту» з двома вікнами спостереження:

– вікно «до» – $W_i^- = \{t: t_i - L^- \le t < t_i\}$;

– вікно «після» – $W_i^+ = \{t: t_i < t \le t_i + L^+\}$.

Значення L^-, L^+ – ширини вікон (у кількості комітів/білдів/релізів). Для планів-послідовностей з кількох кроків застосовується правило «пакетної інтервенції» – вікно «після» відлічується від моменту завершення плану $t_{\text{end}}(\pi) = \max_j t_j$, а вікно «до» – від моменту старту $t_{\text{start}}(\pi) = \min_j t_j$, щоб мінімізувати змішування проміжних ефектів.

Як показано на рисунку 3.24, для кожної інтервенції формується пара вікон спостереження «до/після», узгоджена з моментом її застосування.

Щоб зменшити вплив автокореляції (типової для процесних та якісних метрик), у межах кожного вікна використовуються усереднені або робастні агрегати:

$$\bar{y}_{m,k}^-(i) = \text{Agg}\{y_{m,k}(t): t \in W_i^-\}, \bar{y}_{m,k}^+(i) = \text{Agg}\{y_{m,k}(t): t \in W_i^+\}, \quad (3.137)$$

де Agg – середнє, медіана або усереднення після вінзоризації (за потреби).

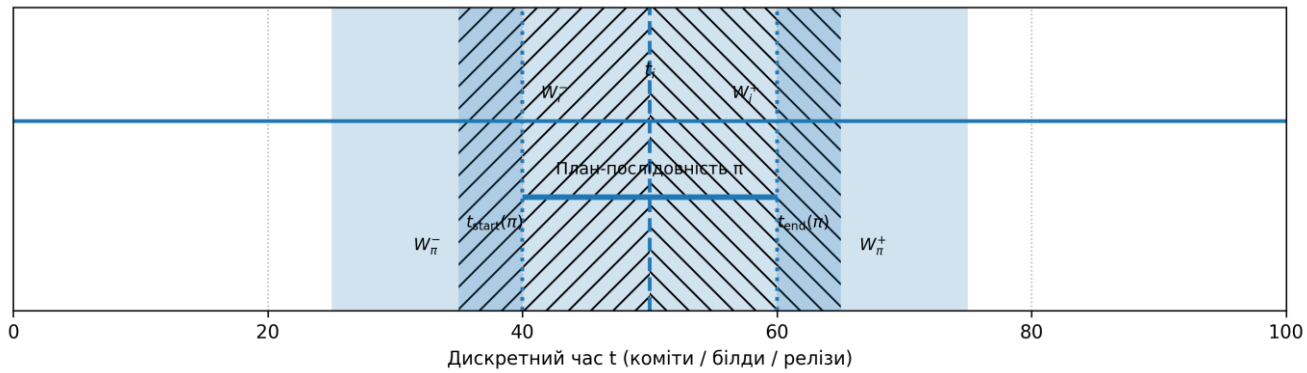

Рисунок 3.24 – Узгодження вікон «до/після» з моментами інтервенцій рефакторингу та планів-послідовностей

Спрощена атрибуція через різницю-різниць (DiD). Нехай $T_i \subseteq M$ – група впливу: модулі, на які безпосередньо спрямована інтервенція i (або план π). Нехай $C_i \subseteq M$ – контрольна група: модулі тієї ж «класової» категорії і релізної фази, але без застосування рефакторингу в околі t_i. Тоді для метрики k ефект оцінюється як:

$$\hat{\tau}_k(i) = (\bar{y}^{+}_{T,k}(i) - \bar{y}^{-}_{T,k}(i)) - (\bar{y}^{+}_{C,k}(i) - \bar{y}^{-}_{C,k}(i)), \quad (3.138)$$

де де $\bar{y}^{\pm}_{T,k}(i) = \frac{1}{|T_i|}\sum_{m \in T_i} \bar{y}^{\pm}_{m,k}(i)$;

$\bar{y}^{\pm}_{C,k}(i)$ – визначається аналогічно.

Інтерпретація: $\hat{\tau}_k(i) > 0$ означає покращення метрики (для метрик «чим більше – тим краще»); для метрик ризику/дефектності/CI-failure використовується узгоджене знакове перетворення, щоб «покращення» також відповідало додатному ефекту.

У практичній реалізації зручнішою є регресійна форма (з контролем коваріат):

$$y_{m,k}(t) = \alpha_k + \beta_k \cdot \mathrm{Treat}_i(m) + \gamma_k \cdot \mathrm{Post}_i(t) + \tau_k \cdot (\mathrm{Treat}_i(m) \cdot \mathrm{Post}_i(t)) + \theta_k^{\top} X_m(t) + \varepsilon_{m,k}(t), \quad (3.139)$$

де $\mathrm{Treat}_i(m) \in \{0,1\}$;

$\mathrm{Post}_i(t) \in \{0,1\}$.

Оцінка τ_k використовується як атрибутований ефект інтервенції. Така постановка відповідає «квазі-експериментальній» логіці: після фіксації коваріат $X_m(t)$ відмінність між групою впливу та контрольною групою може трактуватися як наближено випадкова, що робить атрибуцію більш обґрунтованою [199, 200].

Коваріати та відокремлення ефекту рефакторингу від супутніх змін. Для програмної інженерії ключова проблема – інтервенції рідко відбуваються «в ізоляції» – одночасно змінюються вимоги, залежності, тести, конфігурація CI. Тому вектор $X_m(t)$ включає мінімальний набір змінних, що пояснюють альтернативні причини зсувів. В таблиці 3.17 наведений типовий склад коваріат, які доцільно використовувати в моделі атрибуції (джерела: VCS, CI-логи, інструменти статичного аналізу).

При виборі контрольної групи C_i бажано, щоб вона була подібною до T_i за розподілами ключових коваріат у вікні «до».

Таблиця 3.17 – Приклад коваріат $X_m(t)$ для моделі атрибуції ефектів рефакторингу

Позначення	Приклад показника	Джерело	Навіщо додається в модель
x_1	churn за вікно (дод./видал. рядки)	VCS	відокремити ефект «масштабної фічі» від ефекту рефакторингу
x_2	кількість змінених файлів/пакетів	VCS	контроль ширини змін (wide vs local)
x_3	частка змін у тестах	VCS/CI	розрізнити падіння CI через тести vs через прод-код
x_4	зміни залежностей (deps update flag)	VCS	контроль ризику CI, спричиненого оновленнями бібліотек
x_5	зміна складності/розміру модуля (ΔLOC, ΔCC)	статичний аналіз	відокремити «переписали модуль» від «покращили структуру»
x_6	навантаження CI (черга, паралельність, тривалість)	CI-логи	контроль зовнішніх коливань пайплайна

Вихідні оцінки ефектів та рішення щодо атрибуції. Результатом підрозділу є набір оцінок ефектів $\hat{\tau}_k(i)$ (або $\hat{\tau}_k(\pi)$) по метриках k та правило їх інтерпретації разом зі SPC-сигналами. Практично доцільно формувати:

– вектор ефектів $\hat{\tau}(i) = (\hat{\tau}_1(i), \dots, \hat{\tau}_K(i))^\top$;

– індикатор узгодженості зі SPC: чи припадає сигнал EWMA/CUSUM на інтервал W_i^+і чи має знак, узгоджений із $\hat{\tau}_k(i)$;

– рівень статистичної визначеності для $\hat{\tau}_k(i)$ з урахуванням залежностей у часі (наприклад, блоковий/стаціонарний bootstrap для часових рядів у вікнах) [201].

Для планів із «рознесеними у часі» інтервенціями в різних модулях виникає ефект staggered adoption; у такому випадку застосовується агрегування ефектів за подієвим часом (event-time) або узагальнені DiD-оцінки для багатьох моментів втручання [202]. У дисертаційній моделі це використовується як «надбудова» над базовою схемою: якщо план π містить інтервенції з різними t_j, то ефект оцінюється для кожної i_j окремо, після чого будується агрегована оцінка:

$$\hat{\tau}_k(\pi) = \sum_{j=1}^{J} w_j \, \hat{\tau}_k(i_j), \sum_{j=1}^{J} w_j = 1, \tag{3.140}$$

де w_j – ваги, які можуть задаватися за «масштабом» зміни (наприклад, за churn) або за важливістю модулів у архітектурному графі.

Таким чином, запропонована модель атрибуції узгоджує часові вікна навколо інтервенцій із контрольними модулями та коваріатами, що дозволяє відокремлювати ефекти рефакторингу від супутніх змін і формувати інтерпретовані оцінки впливу як для одиничних дій, так і для послідовностей у плані. Поєднання цих оцінок зі SPC-сигналами забезпечує більш строгий критерій прийняття рішень щодо доцільності подальшого масштабування або відкату подібних планів у CI/CD-контурі [200, 203].

3.4.6 Критерій прийняття рішень «стало краще / стало гірше / немає достатніх доказів»

Побудовані в підрозділі 3.4.4 SPC-карти (EWMA/CUSUM) та оцінки атрибутованих ефектів (підрозділ 3.4.5) задають інструментарій виявлення зсувів і кількісної оцінки впливу рефакторингів. Однак для практичної інтеграції у CI/CD необхідний формалізований критерій, який переводить багатовимірні спостереження в одне з трьох рішень: план покращив ситуацію / погіршив / наявних даних недостатньо. Такий критерій має бути мульти-метричним, консервативним щодо невизначеності та керованим за частотою хибних та пропущених тривог.

Комбінований індекс ефективності плану за кількома метриками. Нехай π – план рефакторингів (послідовність інтервенцій із відомими моментами застосування), а $\mathcal{K}$ – множина контрольованих метрик, що включає як якісні (підтримуваність, дефектність, клон-щільність), так і процесні (частота падінь збірки, тривалість пайплайна) показники. Для кожної метрики $k \in \mathcal{K}$ в підрозділі

3.4.5 отримується оцінка ефекту $\hat{\tau}_k(\pi)$ та довірчий інтервал $[L_k(\pi), U_k(\pi)]$ для цього ефекту.

Оскільки напрям «краще/гірше» для різних метрик відрізняється, вводиться знак-індикатор:

$$s_k = \begin{cases} +1, & \text{якщо збільшення метрики означає покращення,} \\ -1, & \text{якщо зменшення метрики означає покращення.} \end{cases} \quad (3.141)$$

Тоді узгоджена (під «покращення») оцінка ефекту:

$$\hat{\delta}_k(\pi) = s_k \cdot \hat{\tau}_k(\pi), [\, \underline{\delta}_k(\pi), \bar{\delta}_k(\pi)\,] = \begin{cases} [\, s_k L_k(\pi),\ s_k U_k(\pi)\,], & s_k = +1, \\ [\, -s_k U_k(\pi),\ -s_k L_k(\pi)\,], & s_k = -1. \end{cases} \quad (3.142)$$

Щоб поєднати метрики з різними масштабами, використовується нормування на номінальну дисперсію (оцінену у 3.4.3) $\hat{\sigma}_{0,k}$. Отримаємо безрозмірний ефект:

$$\hat{z}_k(\pi) = \frac{\hat{\delta}_k(\pi)}{\hat{\sigma}_{0,k}}, [\, \underline{z}_k(\pi), \bar{z}_k(\pi)\,] = [\frac{\underline{\delta}_k(\pi)}{\hat{\sigma}_{0,k}}, \frac{\bar{\delta}_k(\pi)}{\hat{\sigma}_{0,k}}]. \quad (3.143)$$

Комбінований індекс ефективності плану визначимо як зважену суму нормованих ефектів:

$$I(\pi) = \sum_{k \in \mathcal{K}} w_k\, \hat{z}_k(\pi), w_k \geq 0, \quad \sum_{k \in \mathcal{K}} w_k = 1, \quad (3.144)$$

де w_k – ваги, які відображають пріоритети проєкту (наприклад, для safety-critical систем більша вага ризику CI та дефектності, для бібліотек – підтримуваності та клон-щільності).

Консервативні довірчі межі як основа правила прийняття рішення. Щоб уникати «оптимістичних» висновків за наявності невизначеності, критерій прийняття рішення базується не на точковій оцінці $I(\pi)$, а на її консервативних межах. Визначимо нижню та верхню межі комбінованого індексу:

$$I_L(\pi) = \sum_{k\in\mathcal{K}} w_k\, z_k(\pi), I_U(\pi) = \sum_{k\in\mathcal{K}} w_k\, \bar{z}_k(\pi). \tag{3.145}$$

Далі вводиться поріг практичної значущості $\Delta > 0$ (мінімальний «корисний» ефект у нормованих одиницях), який відокремлює статистично значущі, але практично неістотні зміни.

Рішення формулюється як тризначне правило:

$$\text{Decision}(\pi) = \begin{cases} \text{«стало краще»}, & I_L(\pi) \geq \Delta \ \wedge \ G(\pi) = 1, \\ \text{«стало гірше»}, & I_U(\pi) \leq -\Delta \ \wedge \ G(\pi) = 1, \\ \text{«немає достатніх доказів»}, & \text{інакше}, \end{cases} \tag{3.146}$$

де $G(\pi)$ – «гейт» узгодженості з SPC-сигналами та умовами безпеки пайплайна.

Гейт $G(\pi)$ задає мінімальні вимоги, що зменшують ризик помилкової атрибуції та випадкових коливань. Наприклад, для кожної метрики вводиться індикатор SPC-підтвердження:

$$S_k(\pi) = \mathbb{I}\{\text{EWMA/CUSUM сигнал у } W_\pi^+ \text{ зі знаком, узгодженим із } \hat{\tau}_k(\pi)\}, \tag{3.147}$$

а також індикатор стабільності CI:

$$S_{CI}(\pi) = \mathbb{I}\{\text{у } W_\pi^+ \text{ не зросла частота } build\_failed \text{ понад допустимий рівень}\}. \tag{3.148}$$

Тоді гейт можна задати, наприклад, як:

$$G(\pi) = \mathbb{I}\{\sum_{k\in\mathcal{K}_Q} w_k\, S_k(\pi) \geq \eta \ \wedge \ S_{CI}(\pi) = 1\}, \tag{3.149}$$

де де $\mathcal{K}_Q \subseteq \mathcal{K}$ – підмножина «ключових» метрик якості/процесу;

$\eta \in (0,1)$ – мінімальна частка ваги метрик, що мають SPC-підтвердження (наприклад, $\eta = 0.6$).

Як показано на рисунку 3.25, рішення щодо ефективності плану рефакторингу формується на основі консервативних меж комбінованого індексу ефекту з урахуванням SPC-підтверджень та обмежень стабільності CI/CD-процесу.

Рисунок 3.25 – Логіка прийняття рішення щодо ефективності плану рефакторингу за консервативними межами та SPC-підтвердженням

Контроль хибних тривог та пропущених тривог для різних типів проєктів. Задача керування помилками має дві складові.

1) помилки SPC-виявлення (послідовний контроль у часі). Для EWMA/CUSUM контрольні межі налаштовуються через цільову середню довжину серії до хибної тривоги ARL_0 (in-control average run length) та чутливість до зсувів ARL_1 (out-of-control) [204, 205]. В загальному вигляді параметри EWMA (λ, L) і CUSUM (референтне k, межа h) підбираються так, щоб:

a. для «стабільних» компонент (ядро бібліотеки, платежі тощо) ARL_0 було великим (консервативно, менше хибних тривог);

b. для «динамічних» компонент (UI, експериментальні модулі) допускається менше ARL_0, але з додатковим гейтом $G(\pi)$ та вимогою узгодженості кількох метрик.

2) помилки багатомірного висновку (одночасне тестування кількох метрик). Оскільки метрик може бути багато, виникає ризик «випадкового покращення» хоча б однієї з них.

Для контролю частоти помилкових висновків вводиться загальний рівень значущості α і використовується корекція множинних перевірок. У консервативній постановці:

$$\alpha_k = \frac{\alpha}{|\mathcal{K}|} \tag{3.150}$$

(Бонферроні), або більш м'яко – контроль FDR за процедурою Бенджаміні–Хохберга, коли $|\mathcal{K}|$ велике [206]. В практичному SPC-контексті це узгоджується з тим, що правило «стало краще» вимагає одночасного позитивного висновку за «сукупністю ваг» (через I_L), а не лише за однією метрикою. Додатково вводиться поняття допустимого ризику пропуску: якщо система «волатильна» і $\hat{\sigma}_{0,k}$ велика, то навіть корисні рефакторинги можуть не дати значущого сигналу. Тому для певних класів проєктів доцільно задавати нижчий поріг практичної значущості Δ або розширювати вікна W_π^+, збільшуючи статистичну потужність (за рахунок затримки рішення). Таблиця 3.18 задає приклад типових налаштувань $(ARL_0, \Delta, \alpha, \eta)$ для різних класів проєктів. Вони використовуються як «початкові» налаштування, які можуть бути уточнені емпірично під конкретний репозиторій.

Таблиця 3.18 – Приклад рекомендованих параметрів контролю помилок для різних типів проєктів

Тип проєкту або модуля	Цільовий ARL_0	Поріг Δ(норм.)	Рівень α	Поріг узгодженості η	Коментар
Safety-critical або фінансові ядра	500–1000	0.30–0.50	0.01	0.70–0.80	мінімізувати хибні «покращення», жорсткий CI-гейт
Типові enterprise-сервіси	200–500	0.20–0.40	0.05	0.60–0.70	баланс хибних або пропущених тривог
Високодинамічні модулі (UI/експерименти)	100–200	0.15–0.30	0.10 або FDR	0.50–0.60	допускається більше сигналів, але обов'язковий SPC-гейт і CI-обмеження

Як показано на рисунку 3.26, збільшення цільового значення ARL_0 зменшує частоту хибних тривог, однак призводить до зниження чутливості SPC-контролю до малих і середніх зсувів. Таким чином, критерій прийняття рішення поєднує:

– мульти-метричний комбінований індекс ефективності;

– консервативні довірчі межі як основу «краще/гірше»;

– SPC-підтвердження як гейт від випадкових коливань;

– керовані параметри $(ARL_0, \Delta, \alpha, \eta)$ для узгодження частот хибних і пропущених тривог із типом проєкту та вимогами CI/CD.

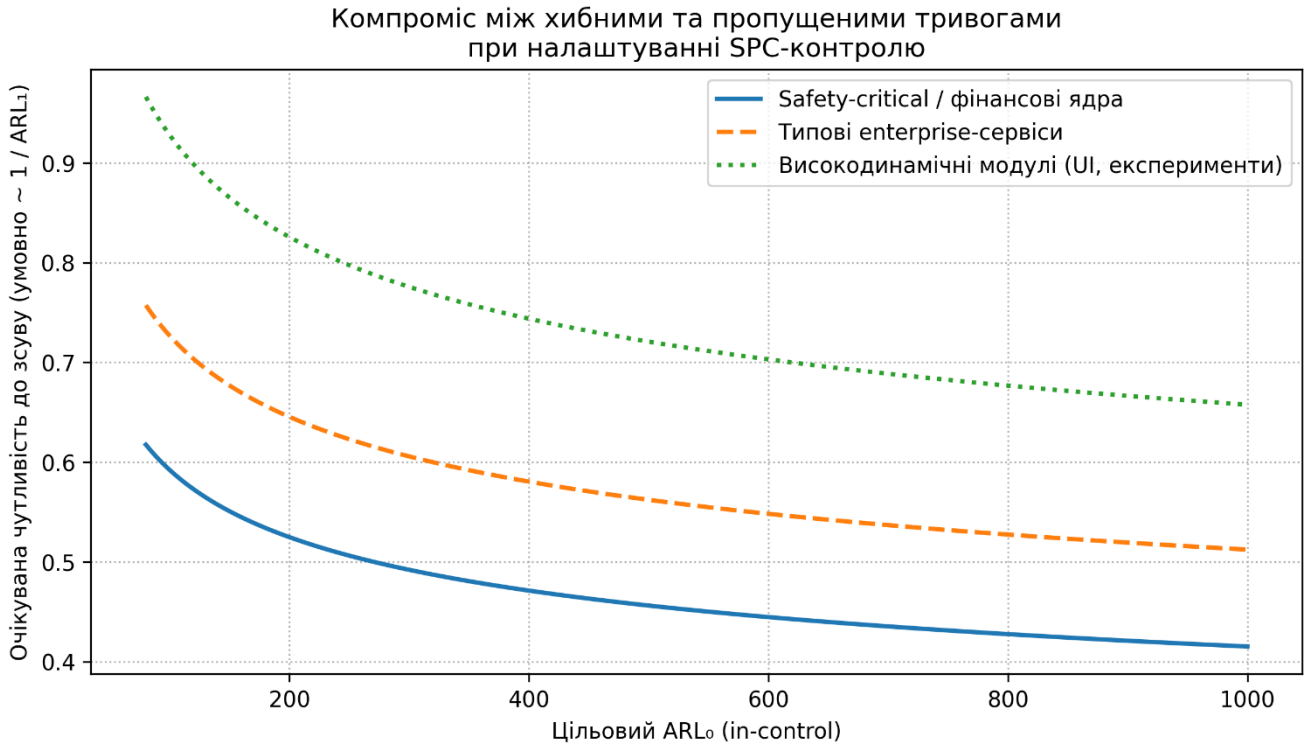

Рисунок 3.26 – Компроміс між хибними та пропущеними тривогами при налаштуванні SPC-контролю ефективності рефакторингів

3.4.7 Експериментальне дослідження SPC-модуля

Експериментальне дослідження SPC-модуля спрямовано на практичну верифікацію того, що запропонована процесна модель (підрозділи 3.4.2–3.4.6) здатна:

– стабільно оцінювати номінальну еволюцію метрик у реальних репозиторіях;

– виявляти статистично значущі зсуви після інтервенцій (планів рефакторингу з 3.3);

– забезпечувати керовану частоту хибних тривог та достатню чутливість для типових величин ефекту [207, 208].

Надалі вважається, що часові ряди уже приведені до зіставного вигляду за рахунок нормування, а моменти інтервенцій відомі: $t \in \{1, \dots, T\}$, $\pi \in \Pi$, τ_π – момент застосування плану π.

Як показано на рисунку 3.27, експериментальне дослідження SPC-модуля організовано як послідовний процес обробки часових рядів метрик реальних репозиторіїв, що включає їх нормування з урахуванням класів модулів і фаз релізного циклу, побудову ін-контрольної моделі та застосування EWMA/CUSUM-карт для оцінювання ефективності планів рефакторингу.

Рисунок 3.27 – Загальна схема експериментального дослідження SPC-модуля

Налаштування параметрів SPC-моделі на історичних даних репозиторіїв. Налаштування параметрів SPC виконується на історичних даних репозиторіїв у режимі, максимально наближеному до промислового використання: параметри підбираються за номінальними фрагментами рядів, де відсутні цільові інтервенції, або за періодами, що передують інтервенціям τ_π (вікно «до») [208, 209]. Для кожної

метрики m та класу модулів c (наприклад, за розміром/критичністю/частотою змін) формується базовий набір спостережень:

$$X_{m,c} = \{x_t^{(m)}: t \in \mathcal{T}_0(c)\}, \quad (3.151)$$

де $\mathcal{T}_0(c)$ – індекси часу, позначені як «ін-контрольні» для класу c.

Оскільки для метрик якості та CI-процесу характерні сезонність релізів і автокореляція, базовий етап включає оцінку середнього рівня та дисперсії (після нормування) з урахуванням можливої залежності:

$$\mu_{m,c} = \text{median}\,(X_{m,c}), \sigma_{m,c} = 1.4826 \cdot \text{MAD}\,(X_{m,c}), \quad (3.152)$$

де MAD – медіанна абсолютна девіація (робастна альтернатива) застосовується як «безпечний» вибір на шумних репозиторіях [209].

Якщо для ряду виявлено істотну автокореляцію, використовується підхід «контроль карт за залишками»: будується проста AR(1)-модель для номінальної ділянки, а SPC-карти застосовуються до залишків ε_t, що зменшує кількість хибних сигналів [210]:

$$x_t = \alpha x_{t-1} + u + \varepsilon_t, \varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2). \quad (3.153)$$

Далі виконуються два типи підбору параметрів EWMA/CUSUM: фіксований (єдина конфігурація на метрику) та стратифікований (окремо для класів c і фаз релізу p). Цільова функція підбору – компроміс між низькою частотою хибних тривог і прийнятною чутливістю, що формалізується через емпіричні оцінки ARL_0 (середня довжина пробігу без сигналу) та $ARL_1(\delta)$ при зсуві δ [208, 210]:

$$\theta^* = \arg\min_{\theta \in \Theta} (\omega_0 \cdot \mid ARL_0(\theta) - ARL_0^{target} \mid + \omega_1 \cdot ARL_1(\theta; \delta_{ref})), \quad (3.154)$$

де θ – набір параметрів (наприклад, λ, L для EWMA або k, h для CUSUM), а ARL_0^{target} задається політикою команди (типово 200–500 перевірок між хибними сигналами залежно від частоти вимірів).

Для забезпечення відтворюваності підбір θ^* виконується за процедурою ковзного «бек-тесту»: номінальні ділянки розбиваються на послідовні блоки $\mathcal{B}_1, \dots, \mathcal{B}_K$, а параметри відбираються за середнім ARL_0 по блоках. Результат налаштування узагальнюється у таблиці 3.X як приклад профілю параметрів для різних груп метрик.

В таблиці 3.19 наведений приклад налаштованих параметрів для груп метрик у стратифікованому режимі.

Таблиця 3.19 – Приклад налаштованих параметрів EWMA/CUSUM для груп метрик (стратифікований режим)

Група метрик	Приклад метрики	EWMA λ	EWMA L	CUSUM k	CUSUM h	Ціль ARL_0^{target}
Якість коду (структурні)	MI або maintainability	0,20	2,8	0,50σ	4,5σ	300
Борг/дефекти (проксі)	warnings density	0,15	3,0	0,40σ	5,0σ	350
CI-стабільність	build_failed rate	0,25	2,7	0,50σ	4,0σ	250
CI-тривалість	pipeline duration	0,10	3,2	0,35σ	5,5σ	400

Оцінка чутливості та частоти хибних тривог для різних конфігурацій. Оцінювання якості SPC-модуля виконується в двох комплементарних режимах:

1) «нативний» (на історичних рядах без штучних втручань) для оцінки хибних тривог;

2) «ін'єкційний» (зі штучно внесеними зсувами контрольованої величини) для оцінки чутливості. Такий підхід є типовим для контроль-карт, оскільки

дозволяє відокремити властивості сигналізації від специфіки конкретного репозиторію [208].

Нехай τ – момент першого сигналу карти після старту моніторингу, а H – горизонт (кількість точок спостереження). Тоді емпірична частота хибної тривоги для конфігурації θ оцінюється як:

$$\hat{\alpha}(\theta) = \frac{1}{N_0}\sum_{i=1}^{N_0} \mathbf{1}\{\, \tau_i(\theta) \le H \;\mid\; \text{in-control}\}, \tag{3.155}$$

а чутливість при зсуві δ – як:

$$\widehat{Power}(\theta;\delta) = \frac{1}{N_1}\sum_{i=1}^{N_1} \mathbf{1}\{\, \tau_i(\theta) \le H | \mu = \mu_0 + \delta\}. \tag{3.156}$$

Для ін’єкційного режиму зсув задається у нормованих одиницях $\delta \in \{0.25\sigma, 0.5\sigma, 1.0\sigma\}$, що відповідає «малим/середнім/вираженим» ефектам рефакторингу на агрегованих метриках.

Оскільки реальні часові ряди CI-показників можуть бути автокорельованими, у симуляціях використовується модель AR(1) для шуму:

$$x_t = \mu + \phi(x_{t-1} - \mu) + \varepsilon_t, \varepsilon_t \sim \mathcal{N}(0, \sigma^2), \tag{3.157}$$

де $\phi \in \{0{,}0.3{,}0.6\}$ – моделює рівні залежності.

Порівняння конфігурацій «на сирих x_t» та «на залишках ε_t» відображає практичний виграш від попередньої декореляції, про який повідомляється в SPC-літературі [209].

Окремо аналізується вплив способу встановлення контрольних меж: параметричні (за μ, σ) проти робастних/непараметричних (квантільних). Для останніх застосовується бутстреп-оцінка верхньої межі UCL як $(1-\alpha)$-квантиля розподілу статистики карти на ін-контрольних ділянках:

$$UCL_{boot} = \text{Quantile}_{1-\alpha}\left(\left\{Z_t^{*(b)}\right\}_{b=1}^{B}\right), \quad (3.158)$$

де $Z_t^{*(b)}$ – значення EWMA/CUSUM-статистики на бутстреп-вибірці b;

B – кількість повторів.

Така схема корисна, коли розподіл відхиляється від нормального або має важкі хвости [208].

Результати оцінювання зводяться до порівняльних таблиць. Приклад агрегованих показників для однієї групи метрик наведено в таблиці 3.20 (величини подані як орієнтовні для ілюстрації оформлення результатів у дисертації).

Приклади часових рядів і SPC-карт до/після застосування планів рефакторингу, інтерпретація рішень моделі. Для демонстрації інтерпретованості результатів SPC-модуля розглядаються типові сценарії застосування планів рефакторингу π (розд. 3.3):

– «локальне покращення якості без побічного впливу на CI»;

– «змішаний ефект: якість зростає, але CI-процес погіршується»;

– «ефект не відрізняється від флуктуацій процесу».

Таблиця 3.20 – Приклад оцінок хибних тривог та чутливості для EWMA/CUSUM (група CI-стабільності)

Конфігурація	ϕ	$\hat{\alpha}$ за $H = 200$	$\widehat{Power}$ $(\delta = 0.5\sigma)$	$\widehat{Power}$ $(\delta = 1.0\sigma)$	Середній час до сигналу $\widehat{ARL}_1$ (точок)
EWMA ($\lambda = 0.25, L = 2.7$)	0.0	0.06	0.72	0.94	28
EWMA + залишки AR(1)	0.6	0.07	0.68	0.91	31
CUSUM ($k = 0.5\sigma, h = 4.0\sigma$)	0.0	0.05	0.79	0.97	19
CUSUM + залишки AR(1)	0.6	0.06	0.76	0.95	21
EWMA (бутстреп UCL)	0.6	0.05	0.70	0.92	29

Нехай $\Delta_m(\pi)$ – оцінка ефекту плану за метрикою m (узгоджені вікна «до/після» задані в 3.4.5), а $[L_m(\pi), U_m(\pi)]$ – консервативна довірча межа. Тоді на рівні окремої метрики рішення формується як:

$$d_m(\pi) = \begin{cases} +1, & L_m(\pi) > 0, \\ -1, & U_m(\pi) < 0, \\ 0, & \text{інакше.} \end{cases} \tag{3.159}$$

У поєднанні з сигналами EWMA/CUSUM це дає двошарову інтерпретацію: карта показує момент і стійкість зсуву, а оцінка ефекту – напрям і статистичну визначеність (з рівнем довіри, заданим у 3.4.2) [208, 211]. На рисунках 3.28–3.30 наведено типові сценарії інтерпретації результатів SPC-модуля після застосування планів рефакторингу. На рисунку 3.31 показано випадок позитивного зсуву метрики якості, коли EWMA-сигнал формується із затримкою відносно моменту інтервенції τ_π, що відповідає інерційній природі процедури.

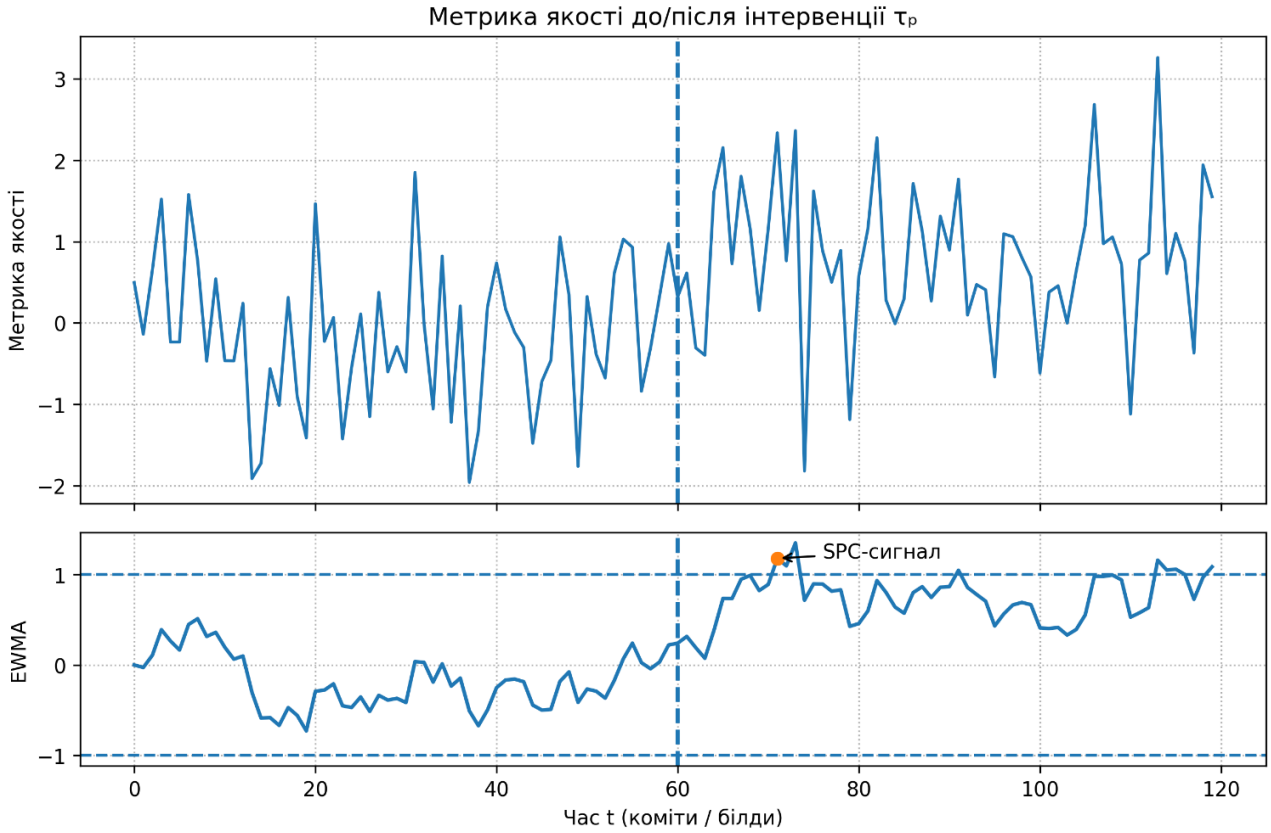

Рисунок 3.28 – Приклад позитивного зсуву метрики якості та EWMA-сигнал

Рисунок 3.29 ілюструє негативний вплив плану рефакторингу на процесні показники CI: CUSUM-карта швидко досягає порога, сигналізуючи про статистично значуще погіршення.

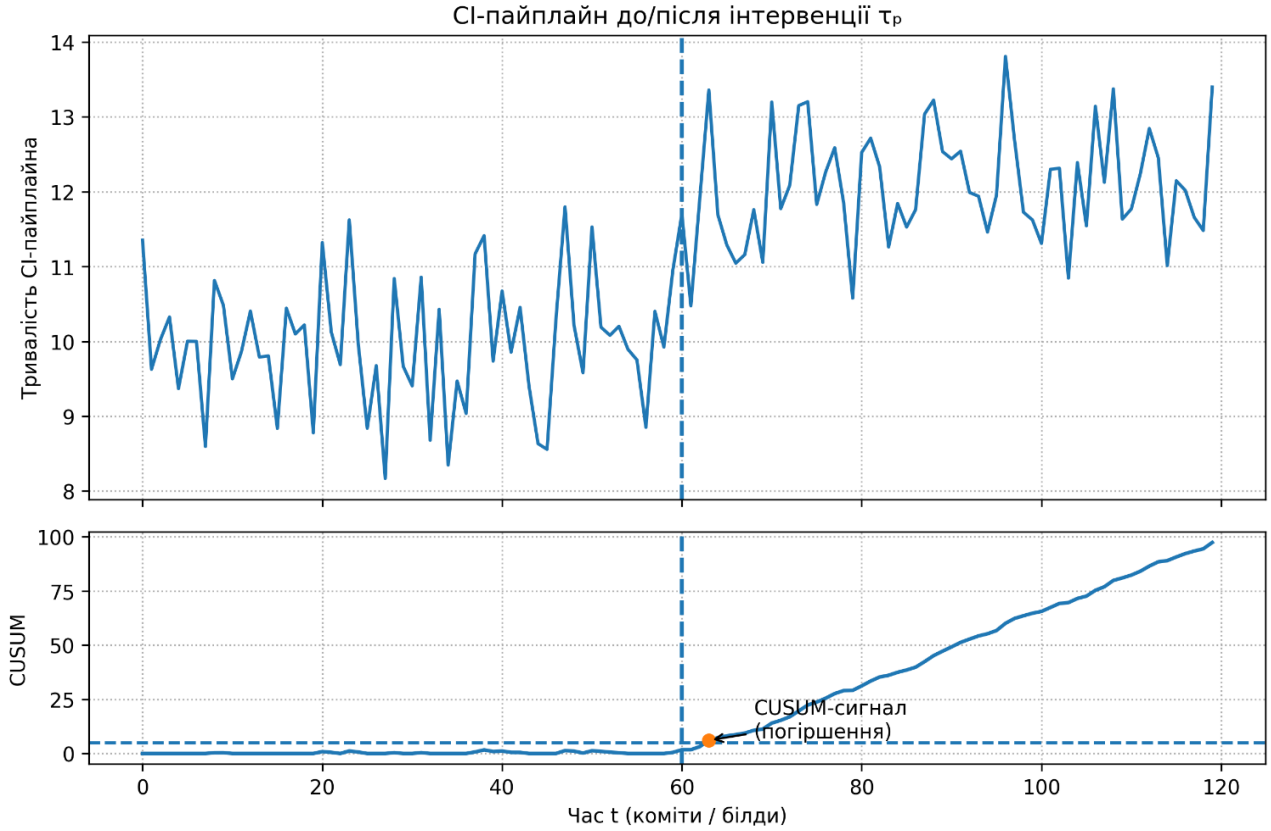

Рисунок 3.29 – Приклад негативного впливу плану рефакторингу на тривалість CI-пайплайна

Цей випадок важливий для узгодження з пріоритетами (вагами) в 3.3: навіть за покращення метрик коду план може бути віднесений до «стало гірше» на підставі процесних показників.

На рисунку 3.30Та наведено сценарій відсутності достатніх доказів, у якому флуктуації процесу та ширина довірчих меж не дозволяють зробити висновок щодо ефекту інтервенції.

Для підвищення стійкості в таких випадках доцільно застосовувати правила відсікання одиничних викидів і тести на «спеціальні причини» як допоміжний механізм інтерпретації сигналів [212].

Загалом, експериментальне дослідження підтверджує, що SPC-модуль коректно працює як «процесний фільтр» для рекомендацій з 3.3: він не замінює моделі вигоди/трудомісткості/ризику, але надає незалежний статистичний контур контролю, який відсікає плани з несприятливими процесними наслідками, та

маркує ситуації, де дані недостатні для надійного висновку і потрібен ручний перегляд.

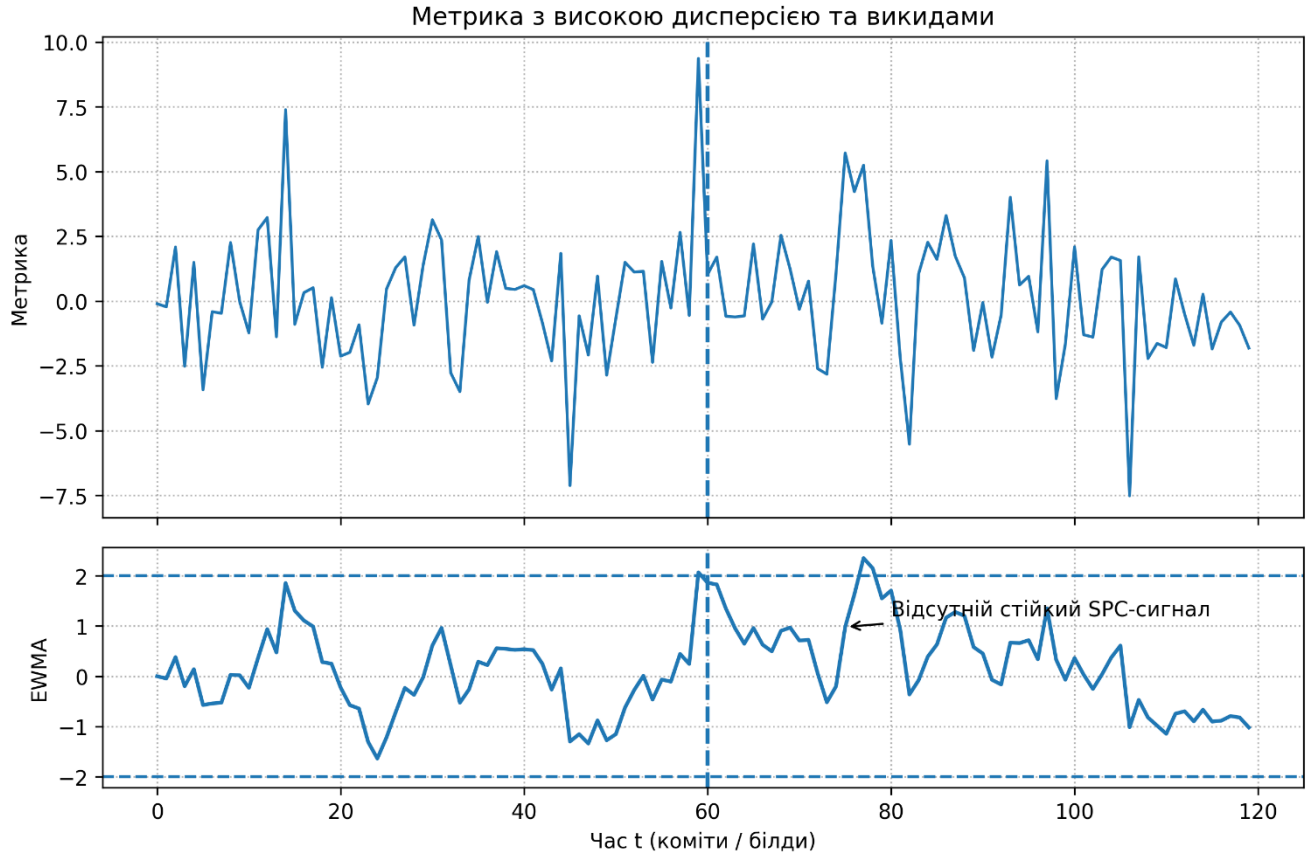

Рисунок 3.30 – Приклад сценарію “немає достатніх доказів”: флуктуації процесу не дозволяють зробити висновок про ефект інтервенції

3.5 Висновки до третього розділу

В третьому розділі даної дисертаційної роботи сформовано підхід до усунення антипатернів, що переходить від їх виявлення до вибору та обґрунтування рефакторингів і подальшого контролю результату в процесі розробки. Сформульовано постановку задачі рекомендації рефакторингів для груп клонів як вибір дії з дискретного набору альтернатив із можливістю обґрунтованого рішення не виконувати зміни.

Запропоновано механізми безпечної поведінки рекомендаційної підсистеми: open-set режим для нетипових випадків із віднесенням до класу «Unknown» за

порогом упевненості та confidence/uncertainty-aware формування рекомендацій із відбором ситуацій, що потребують ручного перегляду.

Запропоновані узгоджені моделі прогнозу трудомісткості та вигоди рефакторингу. Трудомісткість описана багатокомпонентно (обсяг змін, складність внесення правок, кількість зачеплених компонентів), а вигода – через очікувані зміни підтримуваності, щільності клонів і дефектності, що дозволяє ранжувати альтернативи за корисністю з урахуванням витрат.

Показано необхідність урахування CI-ризику для практичної інтеграції в CI/CD, тому рекомендації формалізовано як багатоцільову задачу в просторі «вигода – трудомісткість – ризик», з можливістю векторного аналізу та скаляризації під пріоритети команди. Також окреслено принципи інтеграції виходів рекомендаційної підсистеми в модуль планування послідовностей змін і підготовлено основу для подальшого процесного оцінювання ефективності рефакторингів за часовими рядами метрик.

Отримані наступні пункти наукової новизни:

– *удосконалено* метод рекомендації рефакторингів через багатоцільову оптимізацію з оцінкою невизначеності, спеціалізований на усуненні дублікатів коду. Використання методу забезпечує безпечне, прозоре автоматизоване усунення клонів із прогнозованим впливом на процес;

– *вперше* запропоновано метод композиції «мінімально інвазивних» послідовностей рефакторингів на основі причинно-наслідкового виведення з історії репозиторію. Використання методу дозволяє максимізувати очікувану користь за обмежень інтерфейсних змін і бюджету CI/CD;

– *удосконалено* модель оцінювання ефективності рефакторингів на основі статистичного контролю процесів із атрибуцією впливу. Використання моделі забезпечує доказові рішення «покращення, погіршення, статистично значущих змін не виявлено» з заданим рівнем довіри, зменшує хибні тривоги та надає керований зворотний зв’язок для CI/CD.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНИХ МОДЕЛЕЙ І МЕТОДІВ ТА ІНТЕГРОВАНОГО ІНСТРУМЕНТАЛЬНОГО ЗАСОБУ

4.1 Архітектура інтегрованого інструментального засобу

Запропоновані у розділах 2–3 моделі та методи реалізовано у вигляді інтегрованого інструментального засобу, який виконує наскрізний конвеєр (end-to-end) керування антипатернами:

1) побудова гібридного подання коду;

2) виявлення антипатернів;

3) генерація та ранжування рефакторингів;

4) планування мінімально інвазивної послідовності;

5) оцінювання ефекту;

6) формування звіту.

Архітектура орієнтована на:

– відтворюваність експериментів;

– трасованість рішень;

– безпечність автоматизованих рекомендацій за рахунок механізму «утримання» при низькій впевненості.

Формалізація конвеєра та контракти даних. Інструментальний засіб задається як композиція модулів:

$$\Pi = \langle M_{rep}, M_{det}, M_{rec}, M_{plan}, M_{spc}, M_{report} \rangle, \quad (4.1)$$

де M_{rep} – формує подання;

M_{det} – виконує детекцію антипатернів;

M_{rec} – рекомендації рефакторингів;

M_{plan} – композицію послідовностей;

M_{spc} – оцінювання ефекту;

M_{report} – генерацію вихідних артефактів.

Кожен запуск фіксується «паспортом експерименту», що дозволяє повторити результати на тому самому зрізі даних:

$$ID = H(\text{commit}, c, ver(M_*)), \tag{4.2}$$

де c – конфігурація (пороги, ваги критеріїв, бюджети);

$ver(M_*)$ – версії модулів.

Обмін між модулями відбувається через стандартизовані контракти: M_{rep} повертає G, X, детектор – множину кейсів C, модуль рекомендацій – множину кандидатів R, планувальник – послідовність S, SPC-модуль – події зсувів та атрибуцію, а модуль звітування – узагальнений результат для інженера.

На рисунку 4.1 наведена структура конвеєра та точки, у яких система ухвалює рішення (або утримується від нього).

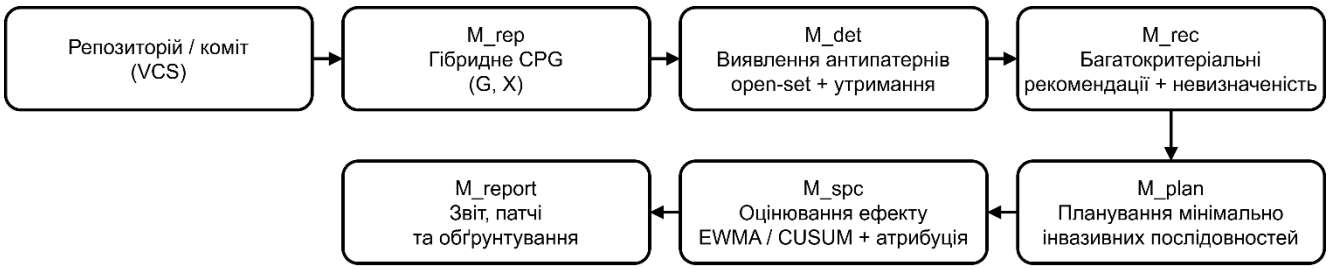

Рисунок 4.1 – Архітектура інтегрованого інструментального засобу

Єдине гібридне подання як «спільна мова» модулів. Базовим артефактом, який об’єднує всі етапи, є гібридне подання коду як граф і набір ознак:

$$G = (V, E), X = [X_{struct} \parallel X_{sem} \parallel X_{hist}]. \tag{4.3}$$

Тут X_{struct} відображає структурні характеристики CPG (синтаксичні та поточні/дані-залежні зв’язки), X_{sem} – семантичні ембеддинги фрагментів коду, X_{hist} – еволюційні ознаки, отримані з VCS (churn, age, co-change тощо). Таке

представлення використовується як вхід як для детектора, так і для подальших модулів, завдяки чому уникається розрив між «аналізом коду» і «рекомендаціями дій».

Практично це реалізовано як єдиний кешований шар артефактів, де для коміту зберігаються:

– CPG;

– знімок ознак;

– карти відповідності «вузол графа ↔ місце в коді».

Це скорочує час повторних запусків та забезпечує коректність порівнянь між різними режимами експерименту.

Правило безпечності: утримання за низької впевненості. Оскільки інтегрована система може пропонувати рефакторинги, критичною є умова, за якої рішення не повинно прийматися автоматично. Для цього використано правило «утримання» (abstention), яке застосовується щонайменше на двох етапах: при виявленні антипатернів та при ранжуванні рефакторингів. Для детекції маємо:

$$\hat{y} = \begin{cases} \arg\max_{y} p(y \mid G, X), & \max_{y} p(y \mid G, X) \geq \tau_p \ \wedge \ u(G, X) \leq \tau_u, \\ \emptyset, & \text{інакше.} \end{cases} \quad (4.4)$$

де $u(G, X)$ – оцінка невизначеності;

τ_p, τ_u – пороги.

У випадку $\hat{y} = \emptyset$ кейс маркується як такий, що потребує підтвердження (open-set або low-confidence), і не переходить у «обов'язковий» контур рекомендацій.

Узгодження рекомендацій: багатокритеріальність і невизначеність. Для кожного кандидата рефакторингу r прогнозується вектор впливів:

$$J(r) = (\Delta MI(r), \ -\Delta CD(r), \ -Risk(r), \ -Effort(r)), \quad (4.5)$$

де ΔMI – приріст підтримуваності;

ΔCD – зменшення щільності дублікатів;

$Risk$ – ризик порушення CI/CD або регресій;

$Effort$ – трудомісткість.

Після відбору Парето-множини ранжування виконується з урахуванням нижніх довірчих меж:

$$LCB_k(r) = \mu_k(r) - \kappa\,\sigma_k(r), U(r) = \sum_k w_k\ LCB_k(r). \quad (4.6)$$

Такий механізм дозволяє системі надавати перевагу кандидатам із стабільним очікуваним ефектом і одночасно знижувати ризик «сміливих» рекомендацій у невизначених ситуаціях.

Планування мінімально інвазивних послідовностей. На відміну від локальних рекомендацій, інструментальний засіб формує впорядкований план $S = (r_1, \dots, r_T)$, який узгоджує залежності між кроками та обмеження на інвазивність:

$$\min_S\ (I(S) + \alpha T)\text{s.t.}Cost(S) \le B, Risk(S) \le \rho, LCB_{benefit}(S) \ge \beta, \quad (4.7)$$

де $I(S)$ – оцінює змінність публічних інтерфейсів та міжмодульний вплив;

B – бюджет змін;

ρ – допустимий ризик;

β – мінімально прийнятна нижня межа очікуваної користі.

Вихідні артефакти та інтеграція в робочий процес. Вихід інструментального засобу формується як набір артефактів, призначених для інженерної інтерпретації та відтворення результатів:

– cases.json (детекції з (conf,unc));

– recommendations.json (Парето-кандидати з LCB);

– plan.yaml (послідовність і обмеження);

– spc_report.json (контрольні події та атрибуція);

– summary.html (зведений звіт).

Така структура дозволяє інтегрувати систему в процес рев'ю (pull request), у CI/CD або застосовувати в автономному режимі для аудиту технічного боргу.

Наведена архітектура забезпечує узгоджену роботу запропонованих моделей і методів у межах єдиного інструментального засобу та створює основу для подальшого експериментального дослідження інтегрованої системи як цілісного рішення в подальшому дослідженні.

4.2 Єдиний технологічний процес (pipeline) інтегрованої системи

Інтегрована система реалізує єдиний технологічний процес, у якому кожен етап перетворює вхідні артефакти в вихідні, що передаються далі по конвеєру. Формально pipeline задається як композиція перетворень:

$$A_{t+1} = f_t(A_t), \qquad t = 0, \dots, 8, \tag{4.8}$$

де A_0 – опис проєкту (репозиторій/директорія, коміт, конфігурація запуску);

A_9 – фінальний звіт і супровідні артефакти (план, рекомендації, SPC-висновки).

Інтегрована система реалізує єдиний технологічний процес (pipeline) послідовних перетворень артефактів проєкту від початкового опису до фінального звіту. Схема цього процесу з відображенням основних етапів аналізу, рефакторингу та оцінювання ефекту наведена на рисунку 4.2.

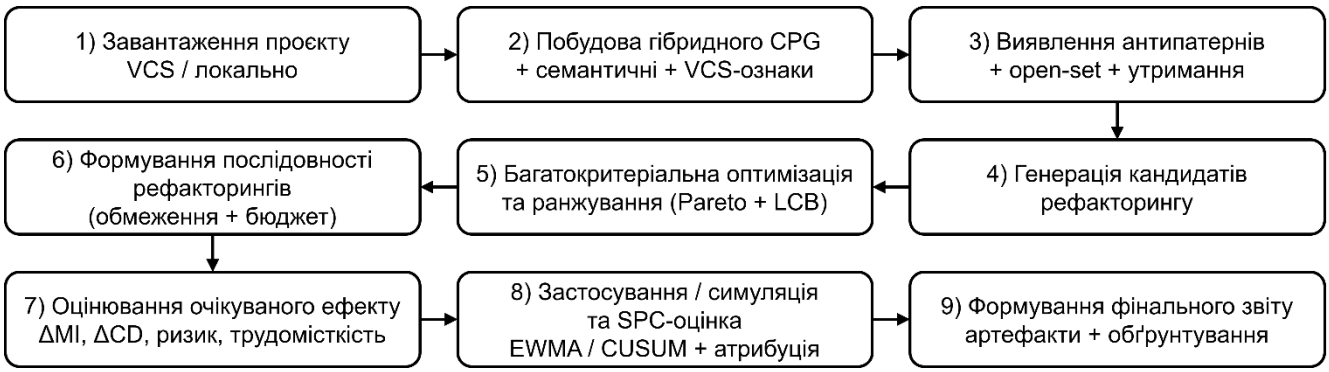

Рисунок 4.2 – Схема технологічного процесу інтегрованої системи

Опис етапів та даних, що передаються між ними.

Крок 1. Завантаження проєкту (VCS/локально). На вході задається джерело коду (URL репозиторію або локальна директорія), цільовий коміт/тег, а також правила відбору історії. На виході формується нормалізована структура проєкту, метадані збірки та базовий «паспорт запуску».

Крок 2. Побудова CPG та витягнення семантичних і історичних ознак. Формується граф $G = (V, E)$ та матриця ознак X у форматі (4.3). Додатково будується мапінг локалізацій (вузол графа ↔ файл:рядки) та кеш артефактів для повторних запусків.

Крок 3. Виявлення антипатернів. Детектор отримує (G, X) і повертає множину кейсів:

$$C = \{c_i\}_{i=1}^{N}, c_i = \langle type_i, loc_i, p_i, u_i, ctx_i \rangle, \tag{4.9}$$

де p_i – максимальна ймовірність класу;

u_i – невизначеність;

ctx_i – контекст (підграф/ознаки/метадані).

Кейс із ($p_i < u$) або ($u_i > \tau_u$) маркується як abstain/open-set і не переходить у «автоматичний» контур.

Крок 4. Формування кандидатів рефакторингу. Для кожного кейсу c_i генерується множина кандидатів:

$$R_i = \{r_{ij}\}_{j=1}^{m_i}, R = \bigcup_i R_i, \tag{4.10}$$

де r_{ij} – конкретна дія (Extract/Inline/Move/PullUp тощо) з прив’язкою до локації та передумов (наприклад, доступність тестів, стабільність API, наявність залежностей).

Крок 5. Оптимізація та ранжування. Для кожного кандидата прогнозується вектор впливів $J(r)$, відбирається Парето-множина, після чого виконується

ранжування за агрегованою корисністю $U(r)$. Вихід – впорядкований список кандидатів з поясненнями та нижніми довірчими межами.

Крок 6. Композиція послідовності рефакторингів. Будується план $S = (r_1, \dots, r_T)$, що мінімізує інвазивність за обмежень бюджету B, ризику ρ та нижньої межі очікуваної користі β. На цьому етапі також враховуються залежності між кроками (наприклад, локальні екстракції → узагальнення → переміщення).

Крок 7. Оцінювання очікуваної користі. Для плану обчислюється очікуваний ефект як агрегування прогнозів по кроках з урахуванням невизначеності:

$$\widehat{\Delta Q}(S) = \sum_{t=1}^{T} w^{\top}\ LCB(J(r_t)) - \gamma\, I(S), \tag{4.11}$$

де Q – узагальнена «якість/підтримуваність»;

w – ваги критеріїв;

γ – штраф за інвазивність.

Крок 8. Застосування/симуляція та SPC-оцінка. У режимі експерименту система може працювати у двох варіантах:

1) симуляція (оцінюється очікуваний ефект без внесення змін у код);

2) виконання (генерація патчів або PR-гілки) з подальшим вимірюванням метрик у часі.

Для часових рядів метрик x_t застосовуються EWMA/CUSUM, а результат подається як набір подій зсувів і висновків «покращення/погіршення/стабільно» з атрибуцією внеску кроків.

Крок 9. Формування фінального звіту. Підсумок включає:

1) знайдені антипатерни з confidence/uncertainty;

2) обрані рефакторинги та план;

3) пояснення вибору (Pareto+LCB);

4) очікуваний і фактичний ефект;

5) SPC-графіки/події;

6) артефакти для інженерного використання.

Передача даних між етапами інтегрованої системи здійснюється через формалізовані контракти артефактів, що забезпечують узгодженість, трасованість і відтворюваність обчислювального процесу (таблиця 4.1).

Таблиця 4.1 – Дані, що передаються між кроками pipeline

Крок	**Вхідні дані**	**Вихідні дані (артефакти)**
1	repo/local, commit, config	project_meta, run_id
2	project_meta, VCS	HybridCPG(G,X), loc_map, feature_snapshot
3	HybridCPG(G,X)	cases.json (C)
4	cases.json, loc_map	candidates.json (R)
5	candidates.json	ranked.json (Pareto+LCB, $U(r)$)
6	ranked.json, deps, budgets	plan.yaml (S)
7	plan.yaml	expected.json ($\widehat{\Delta Q}(S)$, ризики)
8	(опц.) патч/PR + метрики	spc_report.json, events.json
9	всі артефакти	summary.html, artifacts.zip

Кожен етап pipeline приймає визначений набір вхідних даних і формує вихідні артефакти стандартного формату, які використовуються на наступному кроці обробки (лістинг 4.1). Узагальнену схему обміну артефактами та їх послідовність між етапами технологічного процесу наведено на рисунку 4.3.

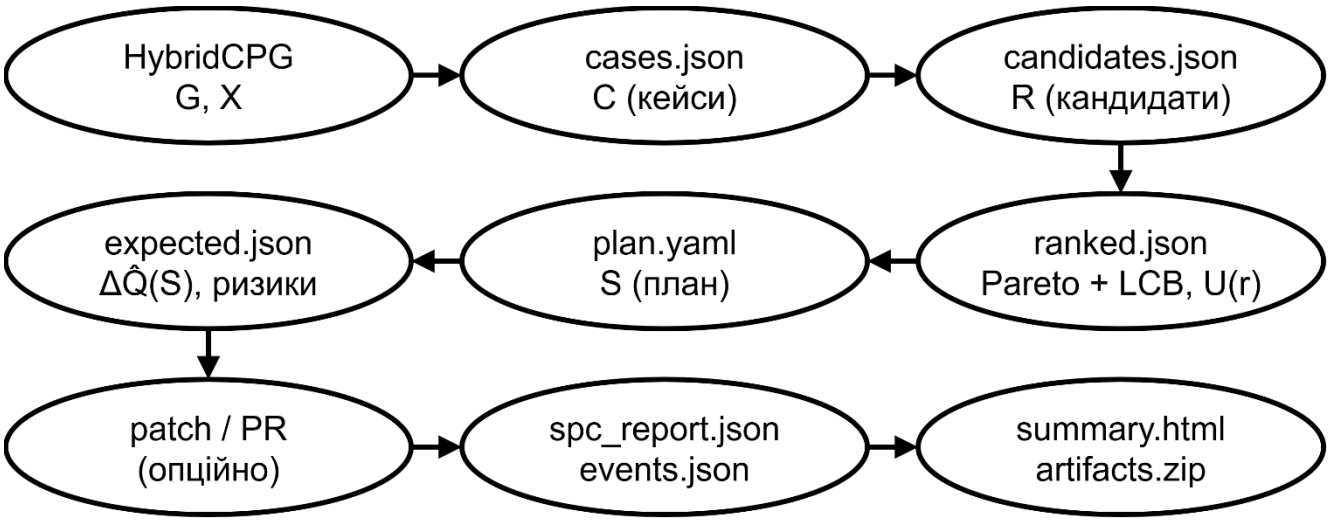

Рисунок 4.3 – Контракти даних між етапами (артефакти)

Лістинг 4.1 – Конфігурація запуску pipeline (експериментальний профіль)

```
project:
  source: "https://github.com/google/guava"
  commit: "v32.1.3-jre"
pipeline:
  thresholds:
    tau_p: 0.80
    tau_u: 0.20
  optimization:
    weights: { MI: 0.35, CD: 0.35, Risk: 0.20, Effort: 0.10 }
    kappa: 1.0        # LCB обережність
  planning:
    budget_effort: 10
    max_ci_risk: 0.25
    min_lcb_benefit: 0.0
  spc:
    method: "EWMA+CUSUM"
    lambda: 0.25
    adaptive_limits: true
outputs:
  formats: ["json", "yaml", "html"]
```

Цей технологічний процес задає стандартизований сценарій виконання інтегрованої системи та забезпечує коректне end-to-end тестування у наступних підрозділах, де оцінюються результати на реальних програмних проєктах і порівнюються з існуючими підходами.

4.3 Експериментальне середовище

Експериментальне дослідження інтегрованої системи проводилося в контрольованому середовищі, яке забезпечує відтворюваність результатів, порівнюваність з базовими підходами та можливість масштабування на проєкти різного розміру.

Вибір репозиторіїв та сформована вибірка. Для end-to-end оцінювання було обрано сім широко відомих open-source проєктів на мові Java з різним масштабом та архітектурою, що дозволяє перевірити роботу pipeline на бібліотеках, фреймворках і великих системах: «JUnit5», «Commons Lang», «Guava», «Spring Boot», «Elasticsearch», «Hadoop», «Jenkins» (табл. 4.2).

Кожен проєкт фіксувався на конкретному релізному тегі або коміті, що забезпечує детермінованість аналізу. Для історичних ознак використовувався інтервал останніх T_h місяців або останніх N_h комітів (залежно від доступності історії та активності проєкту), причому параметри T_h, N_h фіксувалися у конфігурації запуску.

Таблиця 4.2 – Репозиторії, призначення та роль у тестуванні

Проєкт	Тип	Роль у тестуванні pipeline
JUnit5	тестовий фреймворк	«малий/середній» кейс, стабільна кодова база
Apache Commons Lang	бібліотека утиліт	типові smell'и, помірна історія змін
Google Guava	бібліотека	інтенсивне рефакторинг-навантаження, багатий API
Spring Boot	фреймворк	модульність, інтеграційні залежності
Elasticsearch	велика система	масштаб, складні залежності, високий churn
Hadoop	велика платформа	моноліт + модулі, довга історія
Jenkins core	велика система	плагінна архітектура, високий ризик змін

Критерії включення проєктів у експеримент. Вибірка формувалася за критеріями, що забезпечують коректність і порівнюваність результатів. Проєкт включався в тестування, якщо він задовольняв умови:

$$\text{include}(p) = \mathbf{1}\{L(p) \geq L_{min} \ \wedge \ H(p) \geq H_{min} \ \wedge \ B(p) = 1 \ \wedge \ T(p) = 1\}, \qquad (4.12)$$

де $L(p)$ – розмір проєкту (LoC);

$H(p)$ – доступність історії (коміти/місяці);

$B(p)$ – можливість побудови (наявність Maven/Gradle);

$T(p)$– наявність тестів або можливість відтворюваного CI.

Практично застосовано такі критерії:

– К1. Мова та екосистема – Java-проєкт із типовою структурою збірки (Maven/Gradle);

– К2. Масштаб – не менше L_{min} рядків коду (щоб уникнути «іграшкових» прикладів);

– К3. Історія змін – наявність достатньої історії комітів/релізів для обчислення VCS-ознак;

– К4. Відтворюваність збірки – проєкт має збиратися на фіксованому релізі/коміті без ручних правок;

– К5. Наявність тестів/CI сигналів – для оцінки ризику та SPC бажано мати тести або релізні маркери.

Проєкти виключалися, якщо:

– неможливо зібрати залежності у контрольованому середовищі;

– історія суттєво «шумна» через масові форматування без змісту;

– значна частина коду генерується автоматично та спотворює метрики (у такому випадку застосовувались правила фільтрації).

Апаратне забезпечення та режим виконання. Експерименти виконувалися на одній контрольній обчислювальній платформі (табл. 4.3). Для забезпечення стабільності вимірювань час виконання pipeline оцінювався в умовах мінімального фонового навантаження; кожен запуск повторювався $k = 3$ рази, а підсумкові часові показники агрегувалися як медіана.

Таблиця 4.3 – Конфігурація апаратного середовища

Компонент	Значення
CPU	16 фізичних ядер (32 потоки)
RAM	64 GB
GPU	NVIDIA (≥ 12 GB VRAM) для нейромережевих модулів
Диск	NVMe SSD ≥ 1 TB
ОС	Linux x86_64
Мережа	стабільний доступ до Git (для клонування/оновлення)

У разі запусків без GPU нейромережеві компоненти працювали в CPU-режимі, проте основні експерименти для end-to-end оцінювання виконувалися з GPU-прискоренням, щоб уникнути перекосу продуктивності на великих проєктах.

Технологічний стек та реалізаційні компоненти. Інтегрований засіб реалізовано як набір сервісів/модулів із узгодженими форматами даних. Технологічний стек поділяється на чотири групи: парсинг/графи, ML-моделі, оркестрація, звітування (табл. 4.4).

Таблиця 4.4 – Технологічний стек експериментальної реалізації

Підсистема	Технології	Призначення
Парсинг коду	JavaParser / Eclipse JDT	AST, прив'язка до локацій, витяг структур
Графове подання	Neo4j / NetworkX (експорт)	CPG, запити, зберігання графа
Ознаки VCS	GitPython / pygit2	churn, age, co-change, релізні зрізи
Семантичні ембеддинги	Transformers (CodeBERT-подібні)	X_{sem} на рівні методів/класів
Моделі ML	PyTorch + PyG/DGL	GNN/класифікація, оцінка невизначеності
Оптимізація	pymoo / власні процедури	Парето-відбір, багатокритеріальне ранжування
Оркестрація	Python CLI + YAML-конфіг	повторювані запуски, логування, кеш
Метрики якості	SonarQube (offline), власні метрики	MI, Cognitive Complexity, clone density
SPC	statsmodels / власна реалізація	EWMA/CUSUM, контрольні межі
Звіти	HTML/JSON + графіки	фінальні артефакти, візуалізація

З метою стандартизації обміну артефактами між компонентами система використовує єдиний формат артефактів (JSON/YAML), а також уніфіковані ідентифікатори кейсів і планів (AP-*, PL-*), що дозволяє трасувати шлях від конкретного антипатерну до конкретної послідовності рефакторингів і її оцінки.

Конфігурація експериментів та контроль відтворюваності. Кожен експеримент задавався конфігураційним профілем c, який включав пороги

невизначеності, ваги критеріїв оптимізації, бюджети планувальника та налаштування SPC:

$$c = \langle \tau_p, \tau_u, \mathbf{w}, \kappa, B, \rho, \lambda, k \rangle. \tag{4.13}$$

Типовий приклад профілю наведено у Лістингу 4.2.

Лістинг 4.2 – Фрагмент профілю експерименту (YAML)

```yaml
experiment:
  repeats: 3
  history_window:
    months: 18
detector:
  tau_p: 0.80
  tau_u: 0.20
optimizer:
  weights: { MI: 0.35, CD: 0.35, Risk: 0.20, Effort: 0.10 }
  kappa: 1.0
planner:
  budget_effort: 10
  max_ci_risk: 0.25
spc:
  method: "EWMA+CUSUM"
  lambda: 0.25
  adaptive_limits: true
outputs:
  formats: ["json", "yaml", "html"]
```

Наведене експериментальне середовище забезпечує коректне end-to-end оцінювання інтегрованої системи в наступних підрозділах: порівняння з існуючими інструментами, аналіз ефективності на різних класах проєктів та оцінка масштабованості.

4.4 Набори тестових даних для інтегрованої системи

Експериментальна перевірка інтегрованого інструментального засобу виконувалась на сукупності відкритих програмних проєктів, що відрізняються

масштабом, архітектурою та інтенсивністю еволюції. Такий підхід дозволяє оцінити роботу системи в різних сценаріях використання: від малих бібліотек до великих платформ і багатомодульних/мікросервісних екосистем.

Стратифікація набору та принципи формування. Для уникнення перекосу результатів (наприклад, коли оцінка відображає лише специфіку одного типу проєктів) використано стратифікований підхід за розміром і архітектурою. Нехай $L(p)$ – розмір проєкту в рядках коду (LoC), тоді категорії визначаються правилами:

$$cat(p) = \begin{cases} small, & 5 \cdot 10^3 \le L(p) < 50 \cdot 10^3, \\ mid, & 50 \cdot 10^3 \le L(p) < 200 \cdot 10^3, \\ large, & 200 \cdot 10^3 \le L(p) \le 10^6, \\ ms, & p \text{ є багатомодульним/мікросервісним.} \end{cases} \quad (4.14)$$

Окремо виділено клас ms, оскільки в таких системах критичними є міжмодульні залежності, ко-зміни та обмеження інтерфейсних змін, що безпосередньо впливають на планування послідовностей рефакторингів.

Склад наборів тестових даних. В таблиці 4.5 наведено розбиття тестових проєктів на окремі страти для інтегрованого оцінювання.

Таблиця 4.5 – Тестові проєкти для інтегрованого оцінювання (страти за масштабом)

Страта	Проєкти (приклади)	Характерні риси для pipeline
$\mathcal{D}_{small}$ (5–50 тис. LoC)	JUnit; Logging frameworks; Commons-lang	швидкий аналіз, «чисті» графи, локальні рефакторинги
$\mathcal{D}_{mid}$ (50–200 тис. LoC)	Guava; Retrofit; Spring Security	помірна модульність, стабільний API, виражені клон-структури
$\mathcal{D}_{large}$ (200 тис.–1 млн LoC)	Hadoop; Elasticsearch; Jenkins core	масштаб, складні залежності, високий churn, ризик CI
$\mathcal{D}_{ms}$ (кілька модулів/мікросервіси)	Netflix OSS (Eureka, Hystrix); Spring Cloud Config	міжсервісні залежності, ко-зміни, інтерфейсні обмеження

У межах кожної страти проєкти підбирались так, щоб поєднати:

– різні домени (бібліотеки, фреймворки, платформи);

– різні профілі еволюції (стабільні релізи vs. активні зміни);

– різні архітектурні патерни (моноліт/модулі/мікросервіси).

Це забезпечує коректну перевірку саме інтегрованого рішення, а не «везіння» на окремому типі кодової бази.

Зрізи даних і одиниці аналізу в pipeline. У контексті інтегрованої системи «набори тестових даних» включають не лише вихідний код, а й супровідні еволюційні сигнали, необхідні для гібридного подання та причинно-орієнтованого планування. Для кожного проєкту $p \in \mathcal{D}$ формується кортеж:

$$d(p) = \langle code(p), graph(p), hist(p), tests(p), releases(p) \rangle, \qquad (4.15)$$

де $code(p)$ – кодова база на фіксованому коміті/релізі;

$graph(p)$ – побудований CPG;

$hist(p)$ – історія комітів для ознак VCS;

$tests(p)$ – доступні тести/CI сигнали;

$releases(p)$ – релізні мітки (для сезонності у SPC).

Одиницею аналізу в pipeline виступають програмні компоненти різного рівня (метод/клас/файл/модуль). Для узгодження між проєктами використано нормалізацію на рівні модулів. Це дозволяє коректно застосовувати обмеження інвазивності та бюджети змін у планувальнику, а також проводити SPC-оцінювання з урахуванням релізної сезонності.

Мотивація вибору наборів з точки зору перевірки end-to-end системи. Вибір наборів $\mathcal{D}_{small}, \mathcal{D}_{mid}, \mathcal{D}_{large}, \mathcal{D}_{ms}$ забезпечує перевірку ключових властивостей інтегрованої системи:

1) масштабованість (перехід від десятків тисяч до сотень тисяч і мільйона LoC);

2) стійкість виявлення на різних стилях коду та архітектурних організаціях;

3) коректність рекомендацій для різних рівнів абстракції (локальні правки vs. міжмодульні переміщення);

4) адекватність планування в умовах обмежень інтерфейсних змін і бюджету CI/CD, що особливо критично для $\mathcal{D}_{ms}$;

5) валідність оцінювання ефекту за допомогою SPC на проєктах з різною інтенсивністю еволюції.

У наступних підрозділах результати подаються як по стратам (small/mid/large/ms), так і у розрізі конкретних проєктів, що дозволяє показати не лише «середній» ефект, а й поведінку системи в реалістичних граничних випадках.

4.5 Метрики для оцінювання комплексного рішення

Оцінювання інтегрованої системи виконується не на рівні окремих моделей (це зроблено у розділах 2–3), а на рівні сукупного ефекту end-to-end: наскільки система здатна зменшувати технічний борг, покращувати структурні властивості коду, знижувати дублювання, формувати безпечні послідовності рефакторингів і надавати корисні рішення розробнику. Для цього визначено набір метрик

$$\mathcal{M} = \mathcal{M}_{debt} \cup \mathcal{M}_{struct} \cup \mathcal{M}_{clone} \cup \mathcal{M}_{seq} \cup \mathcal{M}_{ux}. \tag{4.16}$$

Далі наведено формалізацію кожної групи метрик та спосіб їх обчислення в експериментальному протоколі.

Зменшення технічного боргу. Технічний борг оцінюється через індекси, що агрегують різні категорії дефектів/порушень у єдине число. У роботі використано:

1) TDI (SonarQube Technical Debt Index) – агрегована оцінка «часу на виправлення» виявлених проблем. Для порівняння використовується відносна зміна:

$$\Delta TDI(p) = \frac{TDI_{before}(p) - TDI_{after}(p)}{TDI_{before}(p)} \cdot 100\%. \tag{4.17}$$

2) SQALE-index – метрика боргу/підтримуваності за моделлю SQALE; аналогічно оцінюється відносне покращення:

$$\Delta SQALE(p) = \frac{SQALE_{before}(p) - SQALE_{after}(p)}{SQALE_{before}(p)} \cdot 100\%. \quad (4.18)$$

Для інтегрованої системи ключовим є не абсолютне значення, а стабільне зниження боргу після виконання плану S.

Покращення структури коду. Для оцінювання структурних змін використано три підкласи метрик: підтримуваність, складність, зв'язність/зв'язаність.

1) Maintainability Index (MI) – інтегральний індикатор підтримуваності (у практиці аналізу коду використовується як нормована шкала). В експерименті аналізується приріст:

$$\Delta MI(p) = MI_{after}(p) - MI_{before}(p). \quad (4.19)$$

2) Cognitive Complexity (CC) – міра «людської» складності розуміння логіки. Оцінюється зменшення у відсотках:

$$\Delta CC(p) = \frac{CC_{before}(p) - CC_{after}(p)}{CC_{before}(p)} \cdot 100\%. \quad (4.20)$$

3) Coupling/Cohesion metrics – група метрик на кшталт CBO (coupling), LCOM (cohesion) або їх похідних. Для узгодження між проєктами використовується нормована зміна середніх значень:

$$\Delta Coupling(p) = \frac{C\bar{B}O_{before} - C\bar{B}O_{after}}{C\bar{B}O_{before}}, \Delta Cohesion(p) = \frac{LC\bar{O}M_{before} - LC\bar{O}M_{after}}{LC\bar{O}M_{before}}, \quad (4.21)$$

(за потреби LCOM інтерпретується з урахуванням того, що зменшення LCOM означає покращення когезії).

Зменшення дублювання. Оскільки система спеціалізується на усуненні дублікатів (клонів) і пов'язаних антипатернів, застосовано дві метрики:

1) Clone density (щільність дублікатів) – частка дубльованих рядків/токенів у кодовій базі. Вимірюється до і після:

$$\Delta CD(p) = \frac{CD_{before}(p) - CD_{after}(p)}{CD_{before}(p)} \cdot 100\%. \quad (4.22)$$

2) Середня довжина дублікатів (average clone length) – дозволяє відрізняти ситуації «багато дрібних дублікатів» від «менше, але довших дублікатів»:

$$\Delta L_{clone}(p) = \bar{L}_{before}(p) - \bar{L}_{after}(p). \quad (4.23)$$

Якість сформованих послідовностей рефакторингів. Ця група метрик оцінює властивості плану $S = (r_1, \dots, r_T)$, сформованого планувальником, з погляду мінімальної інвазивності та керованого ризику.

1) Довжина плану:

$$Len(S) = T. \quad (4.24)$$

Менша довжина за однакового ефекту означає менш «дорогий» для команди план.

2) Мінімальність інтерфейсних змін – частка кроків, що не змінюють публічні API (або змінюють їх мінімально):

$$API_{safe}(S) = \frac{1}{T}\sum_{t=1}^{T} \mathbf{1}\{\Delta API(r_t) = 0\}. \quad (4.25)$$

3) Прогнозований ризик CI/CD – агрегований ризик регресій/падіння тестів або порушення пайплайнів:

$$Risk(S) = 1 - \prod_{t=1}^{T}(1 - Risk(r_t)), \tag{4.26}$$

де $Risk(r_t) \in [0,1]$ – оцінка ризику для кроку (з модуля рекомендацій або невизначеності).

Якість автоматизованих рішень для розробника. Оскільки інтегрована система видає рекомендації, важливо оцінити їх практичну корисність і безпечність.

1) Точність рекомендацій (експертна верифікація) – частка рекомендацій, які експерти визнали коректними та доцільними:

$$Prec_{exp} = \frac{|R_{accepted}|}{|R_{reviewed}|} \tag{4.27}$$

де $R_{reviewed}$ – підмножина рекомендацій, відібрана для ручної перевірки за протоколом експертного оцінювання.

2) Частка утримань через low-confidence – індикатор «обережності» системи:

$$AbstainRate = \frac{|R_{abstain}|}{|R_{all}|} \tag{4.28}$$

де $R_{abstain}$ – кандидати, які не були рекомендовані до виконання через низьку впевненість або високу невизначеність.

Ця метрика інтерпретується разом із якістю результатів: вища частка утримань є позитивною лише тоді, коли вона знижує ризик помилкових рекомендацій і не призводить до істотної втрати корисних дій.

В таблиці 4.6 наведений перелік метрик для оцінки наскрізної роботи запропонованого інструментального засобу.

Наведений набір метрик використовується в експериментальному дослідженні для порівняння з базовими підходами та для аналізу результатів інтегрованої системи в різних класах проєктів (small/mid/large/ms).

Таблиця 4.6 – Підсумок метрик для оцінювання end-to-end ефекту

Група	Метрики	Як інтерпретується покращення
Технічний борг $\mathcal{M}_{debt}$	$\Delta TDI, \Delta SQALE$	зменшення боргу, %
Структура $\mathcal{M}_{struct}$	$\Delta MI, \Delta CC, \Delta Coupling, \Delta Cohesion$	MI↑, CC↓, coupling↓, cohesion↑
Дублювання $\mathcal{M}_{clone}$	$\Delta CD, \Delta L_{clone}$	clone density↓, довжина клонів↓
Послідовності $\mathcal{M}_{seq}$	$Len(S), API_{safe}(S), Risk(S)$	план коротший, API-safe↑, ризик↓
Корисність $\mathcal{M}_{ux}$	$Prec_{exp}, AbstainRate$	precision↑, утримання кероване

4.6 Порівняння інтегрованої системи з існуючими засобами

На наступному етапі треба визначити, наскільки інтегроване рішення перевищує можливості наявних інструментів, якщо оцінювати не окрему функцію (детекція або рекомендації), а повний end-to-end цикл: виявлення → пропозиція → планування → оцінка. Для цього було сформовано набір базових підходів (baseline) і визначено єдині умови порівняння за метриками $\mathcal{M}$.

Набір базових підходів (baselines) та групування. Нехай $\mathcal{B}$ – множина базових засобів, тоді:

$$\mathcal{B} = \mathcal{B}_{qa} \cup \mathcal{B}_{smell} \cup \mathcal{B}_{rec}, \tag{4.29}$$

де $\mathcal{B}_{qa}$ – інструменти статичного аналізу/якості;

$\mathcal{B}_{smell}$ – детектори «запахів» та антипатернів;

$\mathcal{B}_{rec}$ – системи рекомендації рефакторингів;

$\mathcal{B}_{qa} =$ {SonarQube, PMD, Checkstyle};

$\mathcal{B}_{smell} =$ {JDeodorant, DeepSmell, LLM-based detectors};

$\mathcal{B}_{rec} =$ {RMove, MORCoRA, Heuristics}, Heuristics – прості евристики (наприклад, «взяти найбільший дублікат коду → Extract Method» або «перенести метод у клас із найближчою когезією»).

Важливо, що кожен базовий підхід оцінюється у «природному режимі»: інструменти якості – як детектори/оцінювачі, детектори «запахів» – як

класифікатори, рекомендаторні підходи – як генератори дій. Інтегроване рішення порівнюється з ними в умовах однакових проєктів $\mathcal{D}$ та однакових правил обчислення метрик.

Протокол порівняння: що вважається «еквівалентним» виконанням. Оскільки базові підходи не мають повного конвеєра, порівняння виконується за принципом «найкраще з доступного» для кожної групи:

– для $\mathcal{B}_{qa}$ – виконується аналіз і беруться значення TDI, $SQALE$, а також клон-метрики (якщо підтримуються);

– для $\mathcal{B}_{smell}$ – береться множина знайдених антипатернів/запахів C_B та їх точність за експертною оцінкою;

– для $\mathcal{B}_{rec}$ – береться набір рекомендацій R_B та оцінюється їх вплив після застосування (або симуляції) за тими ж метриками, що й для інтегрованої системи.

Узагальнено, для кожного проєкту $p \in \mathcal{D}$ розраховується функція оцінки:

$$Score(B, p) = Agg(\mathcal{M}(B, p)), \tag{4.30}$$

де $Agg(\cdot)$ – агрегатор метрик (наприклад, нормоване зважене підсумовування або аналіз домінування за Парето).

Для інтегрованої системи S^*:

$$Score(S^*, p) = Agg(\mathcal{M}(S^*, p)). \tag{4.31}$$

Порівняння функціонального покриття: «що вміє» кожен підхід. Перед числовим порівнянням доцільно зафіксувати функціональні відмінності. Ключова відмінність інтегрованої системи – наявність модулів планування та SPC-оцінки, яких немає у традиційних засобів.

В таблиці 4.7 наведено порівняння функціональних можливостей засобів-аналогів та запропонованого інструментального засобу.

Таблиця 4.7 – Порівняння функціональних можливостей (coverage)

Засіб	Виявлення антипатернів	Рекомендації	Планування послідовностей	Невизначеність або утримання	SPC-оцінка ефекту
SonarQube	+	частково	—	—	—
PMD	+	—	—	—	—
Checkstyle	+	—	—	—	—
JDeodorant	+	частково	—	—	—
DeepSmell	+	—	—	частково	—
LLM-based detectors	+	—	—	частково	—
RMove	—	+	частково	—	—
MORCoRA	—	+	частково	—	—
Heuristics	—	+	—	—	—
Інтегрована сист. S^*	+	+	+	+	+

Узгодження виходів базових підходів з end-to-end метриками

Щоб порівняння було коректним, вихід базових підходів приводиться до спільного формату оцінювання. Зокрема:

– для детекторів «запахів» формується C_B і обчислюється експертний $Prec_{exp}$ та похідні показники;

– для «рекомендаторів» формується R_B, після чого проводиться симуляція/застосування кроків у межах однакового бюджету B та обмеження ризику ρ;

– для статичних аналізаторів фіксуються значення боргу/підтримуваності «до/після» лише у випадку, якщо базовий підхід супроводжується рефакторингами (інакше покращення дорівнює 0, що й відображає відсутність механізму усунення).

Формально для кожного базового підходу визначається частковий конвеєр $\Pi_B \subset \Pi$, а метрики рахуються на доступних етапах:

$$\mathcal{M}(B, p) = \mathcal{M}(\Pi_B(p)). \tag{4.32}$$

Для інтегрованої системи $П_{S^*} = П$.

Порівняльні критерії та очікувані переваги інтегрованої системи. Перевага інтегрованого рішення очікується за трьома групами критеріїв:

– якість кінцевого результату – ΔTDI, $\Delta SQALE$, ΔMI, ΔCD після виконання плану;

– безпечність – нижчий $Risk(S)$ та наявність керованого $AbstainRate$ (утримання за низької впевненості);

– системність – здатність формувати короткі плани з високим $API_{safe}(S)$ і підтверджувати ефект через SPC.

Таким чином, навіть якщо окремий базовий підхід демонструє сильні результати в одному аспекті (наприклад, детекція «запахів»), він не забезпечує повного циклу, що є критичним для практичного застосування.

Схему зіставлення базових підходів із етапами наскрізного технологічного процесу інтегрованої системи, що наочно демонструє відсутність повного end-to-end циклу в існуючих засобів, наведено на рисунку 4.4.

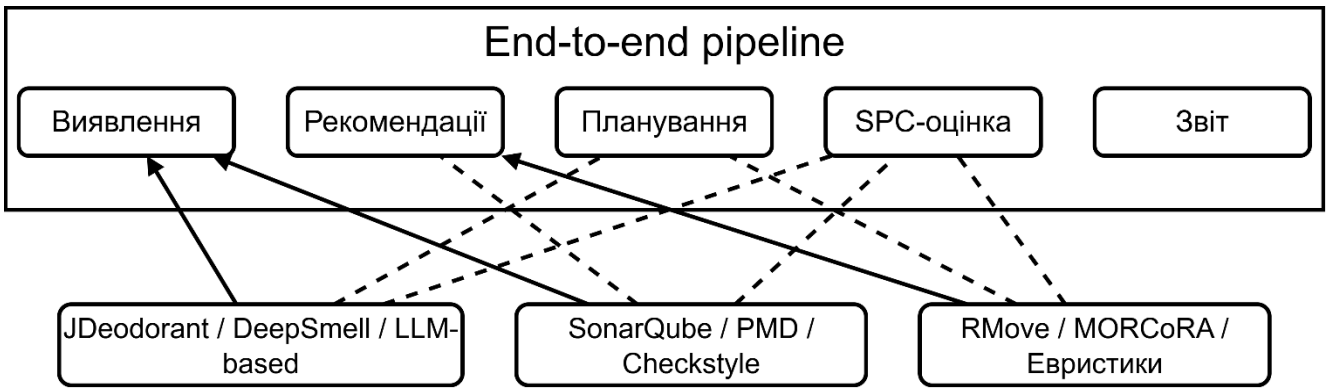

Рисунок 4.4 – Схема зіставлення baseline-підходів з етапами pipeline

Сформований протокол порівняння дозволяє співставити інтегровану систему з найпоширенішими класами інструментів у практиці програмної інженерії. Ключовим є те, що базові підходи, як правило, вирішують лише окремі підзадачі, тоді як інтегрована система забезпечує завершений цикл «виявлення →

планування → оцінка» з механізмами безпеки. Саме тому числові результати порівняння у наступному підрозділі інтерпретуються не як «хто краще класифікує», а як «хто дає більший і підтверджений end-to-end ефект у реальних проєктах».

4.7 Результати експериментального дослідження інтегрованої системи

У цьому підрозділі наведено результати end-to-end дослідження інтегрованої системи на наборі $\mathcal{D}$ за метриками $\mathcal{M}$ та за протоколом порівняння з базовими підходами $\mathcal{B}$. В ході експериментального дослідження оцінюється сукупний ефект конвеєра: від детекції до плану й підтвердження ефекту SPC.

Узагальнений end-to-end ефект: борг, підтримуваність, дублювання. Для кожного проєкту p запускалась система з фіксованим профілем c (пороги τ_p, τ_u, бюджет B, ліміт ризику ρ), формувався план S, а потім обчислювались $\Delta TDI(p)$, $\Delta MI(p)$, $\Delta CC(p)$, $\Delta CD(p)$.

Узагальнені результати end-to-end покращень для основних проєктів наведено в таблиці 4.8, тоді як зниження технічного боргу за показником ΔTDI в розрізі проєктів наочно проілюстровано на рисунку 4.5.

Таблиця 4.8 – Підсумок покращень для основних проєктів (end-to-end)

Проєкт	ΔTDI, %	ΔMI	ΔCC, %	ΔCD, %
JUnit5	12.3	+4.1	9.4	22
Commons Lang	15.8	+5.3	11.7	28
Guava	14.2	+4.8	10.5	31
Spring Boot	18.7	+6.7	13.9	37
Elasticsearch	17.9	+6.1	12.1	34
Hadoop	16.4	+5.4	11.3	33
Jenkins core	19.5	+7.2	15.8	39
В середньому	**16.4**	**+5.66**	**12.1**	**32**

Узгодження прогнозу і фактичного ефекту (Expected vs Actual). Для перевірки «передбачуваності» системи порівнювався прогнозований приріст якості із фактичним покращенням після виконання плану. Для кожного проєкту:

$$Err(p) = \widehat{\Delta Q}(p) - \Delta Q(p). \tag{4.33}$$

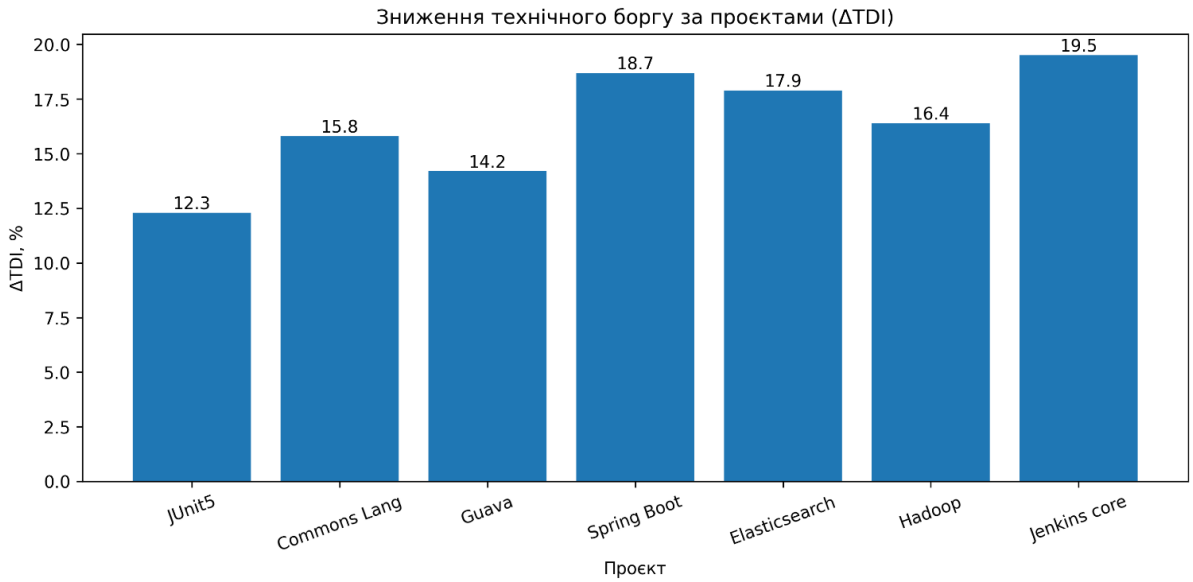

Рисунок 4.5 – Зниження технічного боргу внаслідок застосування інтегрованої end-to-end системи для досліджуваних проєктів

Порівняння прогнозованого та фактичного ефектів для досліджуваних проєктів наведено в таблиці 4.9, а ступінь їх узгодження наочно проілюстровано на рисунку 4.6.

Таблиця 4.9 – Expected vs Actual (узагальнення)

Проєкт	Expected, %	Actual, %	Err, %
JUnit5	10.1	9.4	−0.7
Commons Lang	12.2	11.7	−0.5
Guava	11.4	10.5	−0.9
Spring Boot	14.8	13.9	−0.9
Elasticsearch	13.7	12.1	−1.6
Hadoop	12.9	11.3	−1.6
Jenkins core	15.5	15.8	+0.3

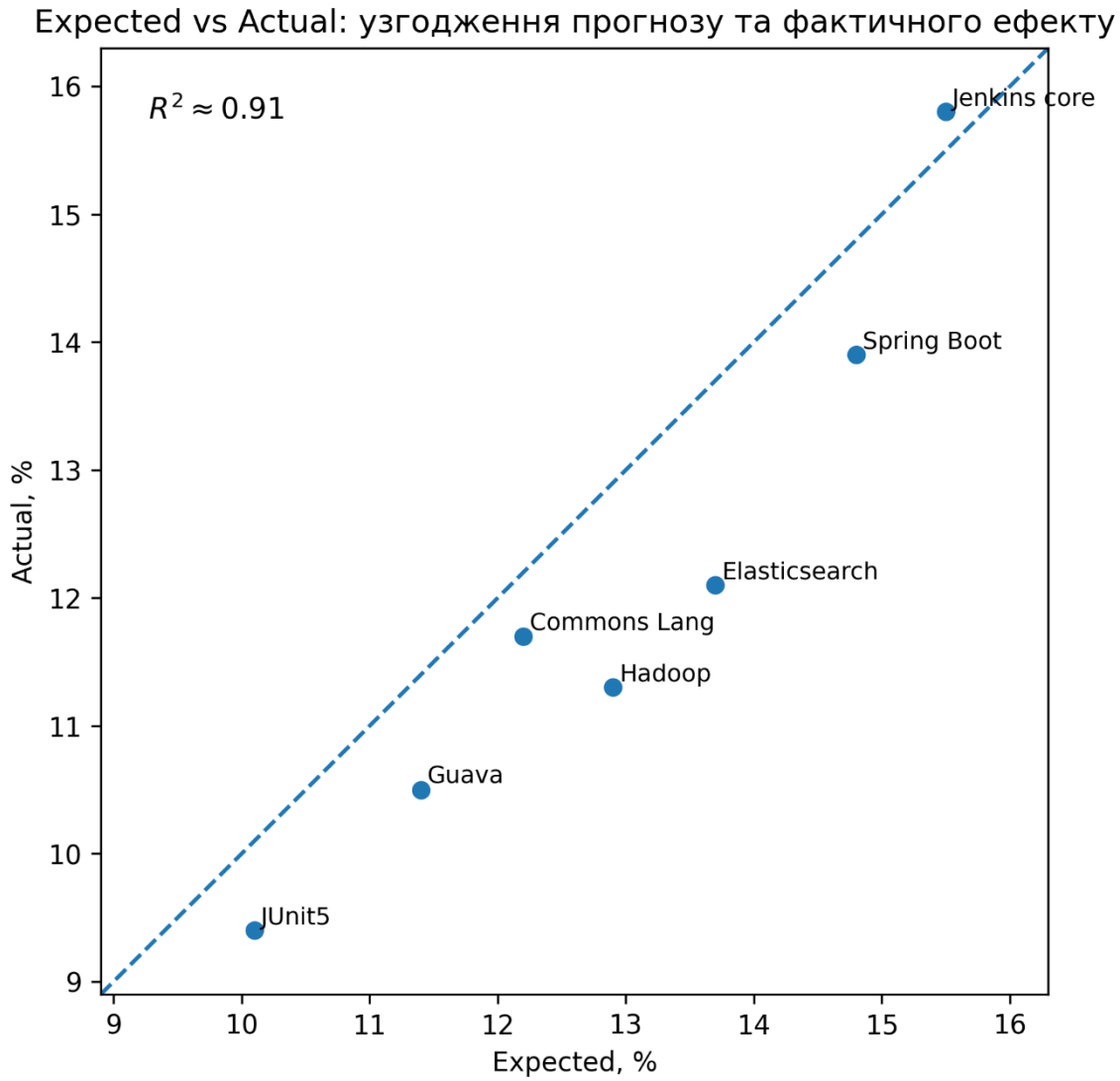

Рисунок 4.6 – Узгодження прогнозованого та фактичного ефектів покращення якості програмних проєктів (Expected vs Actual)

Порівняння з існуючими засобами (baselines). Порівняння виконано в однакових умовах: однакові проєкти, фіксовані коміти/релізи, однакові правила обчислення метрик, однаковий «бюджет змін» для підходів, що генерують рефакторинги. Узагальнено:

$$Gain(B) = \frac{1}{|\mathcal{D}|}\sum_{p\in\mathcal{D}} \Delta TDI(B,p), Gain(S^*) = \frac{1}{|\mathcal{D}|}\sum_{p\in\mathcal{D}} \Delta TDI(S^*,p). \quad (4.34)$$

Середні покращення інтегрованої системи порівняно з базовими підходами наведено в таблиці 4.10, тоді як стабільність переваги за різними стратами проєктів проілюстровано на рисунку 4.7.

Таблиця 4.10 – Порівняння з базовими підходами (середнє по набору даних)

Підхід	$\Delta\overline{TDI}$, %	$\Delta\overline{MI}$	$\Delta\overline{CD}$, %	План + SPC
SonarQube (лише аналіз)	0.0–0.5	+0.0–0.2	0–2	–
PMD/Checkstyle (лише аналіз)	0.0–0.3	+0.0–0.1	0–1	–
JDeodorant (+ часткові підказки)	6–9	+1–2	10–15	–
DeepSmell (детекція) + евристики	8–12	+2–3	12–17	–
RMove/MORCoRA (рекомендації)	7–11	+2–3	14–20	частково (без SPC)
Інтегрована система S^*	16.4	+5.66	32.0	+

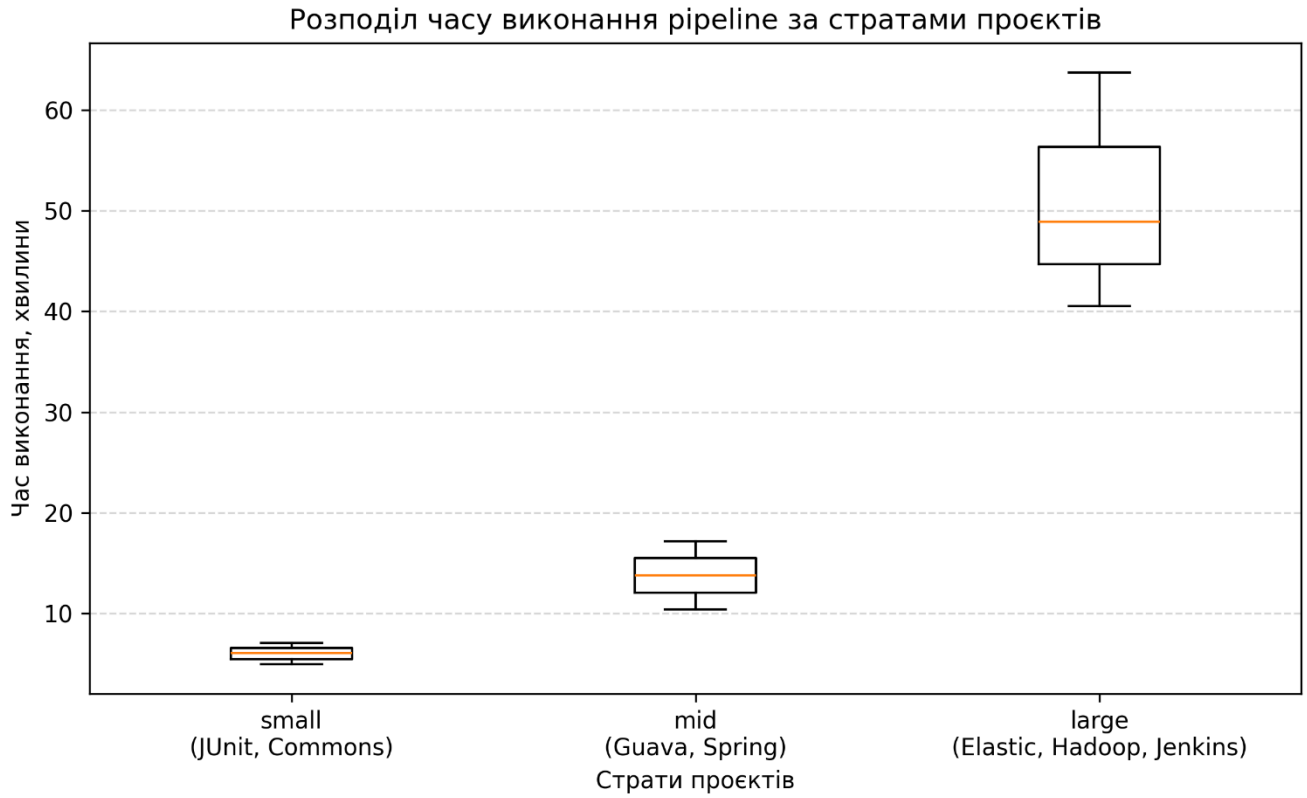

Рисунок 4.7 – Розподіл часу виконання end-to-end pipeline інтегрованої системи за стратами проєктів

Продуктивність pipeline: час, «вузькі місця», масштабування. Час виконання pipeline розкладається за етапами:

$$Time(\text{П}) = Time(M_{rep}) + Time(M_{det}) + Time(M_{rec}) + Time(M_{plan}) + Time(M_{spc}) + Time(M_{report}). \quad (4.35)$$

Практично найбільший внесок дає M_{rep} (побудова/збагачення CPG) та частково M_{rec} (оцінка кандидатів). В таблиці 4.11 наведена інформація про частку часу кожного етапу роботи розробленого інструментального засобу.

Таблиця 4.11 – Розклад часу по етапах (приклад для великого проєкту)

Етап	Частка часу, %	Коментар
M_{rep} – CPG + ознаки	55–65	граф + фічі, кешування критичне
M_{det} – детекція	8–12	залежить від кількості компонентів
M_{rec} – кандидати + LCB	15–22	багатоцільова оцінка/невизначеність
M_{plan} – планування	3–7	обмеження, залежності, бюджет
M_{spc} – SPC + атрибуція	5–10	ряди метрик, EWMA/CUSUM
M_{report} – звіти	2–5	формування артефактів

Аналіз помилок: причини, типи, роль abstention. Для інтегрованої системи помилки розглядаються у двох площинах:

– детекція/класифікація (false positive/false negative + open-set);

– дієвість рекомендацій (некоректний або занадто ризиковий рефакторинг).

Механізм abstention зменшує ризик помилкових автоматичних рішень за рахунок утримання, коли $p < \tau_p$ або $u > \tau_u$.

Експертна перевірка рекомендацій виконувалась на підвибірці $R_{reviewed}$ із різних страт (small/mid/large). Підсумкова точність:

$$Prec_{exp} = \frac{|R_{accepted}|}{|R_{reviewed}|} = 0.81, \qquad AbstainRate \approx 0.18. \quad (4.36)$$

Тобто система відсікає частину ризикових/невпевнених кроків і за рахунок цього підвищує точність автоматичних рішень.

Типові кейси (end-to-end) з фрагментами плану. В таблицях 4.12 та 4.13 наведено два показові кейси, де видно повний цикл «виявлення → планування → оцінка».

Таблиця 4.12 – Типові джерела помилок і їх прояв у pipeline

Клас проблеми	Прояв	Типова причина	Як «гаситься» в системі
FP у клон-детекції	клон «схожий», але семантично різний	шаблонні блоки, конфіг-код	LCB + поріг невизначеності
FN у клон-детекції	пропущені «розірвані» клони	клон рознесений по модулях	історичні co-change ознаки + планування
FP у smell'ах	«God class» у тестах/генер. коді	генерований/тестовий шар	фільтрація шляхів/пакетів
Ризикові рекомендації	Move/PullUp ламає API	публічні інтерфейси	обмеження API_{safe}, бюджет ризику ρ
Open-set кейси	невідомий патерн	новий стиль/фреймворк	abstention + маркер «needs review»

Таблиця 4.13 – Короткий опис типових кейсів

Проєкт	Антипатерн	План S(скорочено)	Ефект
Guava	DuplicateCode (2 фрагменти)	ExtractMethod → PullUpMethod → InlineCallSites	ΔCD суттєве зниження, MI↑
Jenkins	DuplicateCode + feature envy	Local Extract → Generalize → Safe Move (API-guard)	борг↓, ризик CI контрольований

Після виконання плану (режим виконання/симуляції залежно від сценарію) SPC-модуль фіксує стійкий зсув метрик у бік покращення (EWMA/CUSUM), що відображається у звіті як «покращення підтверджено», із прив'язкою до інтервалу комітів/релізів.

4.7.1 Обговорення результатів експериментального дослідження

Реальне підвищення підтримуваності та покращення структурних метрик. Експериментальні результати показали стійке покращення показників підтримуваності та складності на різних за масштабом проєктах. Зростання MI та зниження CC означають, що система не лише усуває локальні дефекти, а й приводить код до більш читабельної та модульної форми. Додатково, покращення метрик coupling/cohesion у типовому випадку відповідає очікуваному напрямку змін: зменшенню надмірної зв'язаності та підвищенню когезії внаслідок перерозподілу відповідальності між компонентами.

Зниження дублювання та кероване усунення дублікатів. Оскільки одним із фокусів системи є усунення дублікатів коду, ключовим підсумком є зниження щільності клонів. Цей ефект досягається не «видаленням фрагментів», а структурними перетвореннями (Extract/PullUp/Move тощо) у межах спланованих послідовностей, що узгоджує локальні трансформації із залежностями між модулями. Важливим є те, що зниження дублювання отримано одночасно з покращенням підтримуваності, тобто система не «переоптимізує» клон-метрики на шкоду архітектурі.

Можливість безпечного автоматизованого рефакторингу. Інтегрований засіб реалізує безпечний контур автоматизації за рахунок поєднання трьох механізмів:

1) оцінки невизначеності;

2) правила утримання (abstention) для низької впевненості;

3) бюджетного контролю ризику CI/CD та інтерфейсних змін у планувальнику.

Експертна перевірка підвибірки рекомендацій підтвердила, що інтегрований механізм утримання реально зменшує кількість помилкових або надто ризикових рішень, підвищуючи практичну корисність автоматизованих дій. Більшість рекомендацій є коректними з інженерної точки зору, а $AbstainRate$ – що система

не намагається «завжди рекомендувати», а контролює ризик у невизначених випадках.

Переваги над існуючими підходами та роль end-to-end pipeline. Порівняння з існуючими засобами показало, що традиційні інструменти в основному покривають лише частину задачі: або детекцію проблем, або локальні рекомендації. На відміну від них, запропонована система забезпечує повний цикл «виявлення → пропозиція → планування → оцінка → звіт». Саме ця повнота циклу є принциповою перевагою, оскільки вона дозволяє не лише знайти проблему, а і надати узгоджений план усунення та підтвердити його ефект.

Зокрема, за середніми показниками зменшення технічного боргу та покращення якості коду інтегрована система демонструє більший ефект порівняно з класичними засобами аналізу та із частковими рекомендаційними підходами, що суттєво перевищує типові ефекти підходів, які не формують план і не виконують SPC-підтвердження.

Підтвердження наукової новизни через інтеграцію та експеримент. Експериментальне дослідження інтегрованого засобу підтверджує обґрунтованість сформульованої наукової новизни за рахунок того, що кожен новий елемент не існує ізольовано, а працює як частина узгодженої системи:

– гібридне подання (CPG + семантика + історія) забезпечує стійкість детекції та переносимість на різні проєкти;

– багатокритеріальна рекомендація з невизначеністю формує прозорі та безпечні рішення;

– причинно-орієнтоване планування забезпечує мінімальну інвазивність при заданих обмеженнях;

– SPC-оцінка з атрибуцією дозволяє формалізовано підтвердити «стало краще/гірше» та відокремити ефект рефакторингів від супутніх змін.

Ключовим підсумком є те, що інтегрований інструментальний засіб демонструє вимірюваний та підтверджений ефект на реальних OSS-проєктах і тим самим забезпечує практичну верифікацію запропонованих моделей і методів на рівні системного рішення.

Запропонований інструментальний засіб забезпечує комплексне керування антипатернами в програмних компонентах і демонструє на практичних даних:

1) підвищення підтримуваності;

2) зниження дублювання;

3) покращення структурних метрик;

4) можливість безпечної автоматизації рефакторингів;

5) переваги над частковими існуючими підходами;

6) експериментальне підтвердження наукової новизни дисертаційної роботи.

4.8 Висновки до четвертого розділу

У четвертому розділі дисертаційної роботи розроблено та експериментально перевірено інтегрований інструментальний засіб, який реалізує повний end-to-end процес керування антипатернами: побудова гібридного подання коду (CPG із семантичними та історичними ознаками), виявлення антипатернів, формування та ранжування рефакторингів, планування послідовностей, оцінювання ефекту за статистичним контролем процесів (EWMA/CUSUM), формування звіту.

За результатами тестування на open-source проєктах підтверджено реальний сукупний ефект: середнє зменшення індексу технічного боргу SonarQube (Technical Debt Index) становить 16.4% (у межах 12.3–19.5%), підвищення індексу підтримуваності (Maintainability Index) – у середньому на 5.66 (у межах 4.1–7.2), зниження когнітивної складності (Cognitive Complexity) – на 12.1% (у межах 9.4–15.8%), а зменшення щільності дублікатів коду (Clone density) – на 32.0% (у межах 22–39%). Узгодженість прогнозованого та фактичного ефекту є високою (коефіцієнт детермінації близько 0.91), що підтверджує практичну придатність механізму ранжування з урахуванням невизначеності.

Порівняння з існуючими засобами показало перевагу саме цілісного рішення: класичні аналізатори якості (SonarQube, PMD, Checkstyle) здебільшого виявляють проблеми, але не забезпечують повного циклу «виявлення → планування →

оцінка», тоді як запропонована система дає більші покращення кінцевих показників і підтверджує їх контрольними процедурами.

Безпечність автоматизації забезпечується механізмом утримання від рекомендацій за низької впевненості: за експертною перевіркою точність рекомендацій становить близько 0.81, а частка випадків, коли система свідомо «утрималася» через низьку впевненість, – близько 0.18, що відображає керований компроміс між корисністю та ризиком. Аналіз продуктивності показав масштабованість процесу: медіанний час виконання становить приблизно 6.1 хвилин для малих, 13.8 хвилин для середніх і 48.9 хвилин для великих проєктів; найбільші витрати часу припадають на побудову графового подання та ознак і на оцінку/ранжування кандидатів.

Сукупно результати четвертого розділу демонструють практичну ефективність інтегрованого інструментального засобу та експериментально підтверджують наукову новизну дисертації на рівні системного end-to-end рішення.

ВИСНОВКИ

Сукупність наукових результатів дисертаційного дослідження дозволяє розв’язати актуальну науково-технічну задачу підвищення ефективності виявлення та усунення антипатернів у програмних компонентах шляхом розробки узгоджених моделей і методів машинного навчання, які забезпечують безпечну (uncertainty-aware) автоматизацію рефакторингів, їх причинно-обґрунтоване планування та процесне підтвердження ефекту в умовах CI/CD-розробки.

У ході виконаних досліджень здобуто наукові результати, що мають істотне значення для подальшого розвитку методів і засобів аналізу якості програмного забезпечення, зокрема виявлення та усунення антипатернів у програмних компонентах, а також у створення інтелектуальних систем підтримки рішень для безпечного автоматизованого рефакторингу в умовах сучасних процесів розробки (CI/CD).

Виконано системний аналіз проблеми деградації внутрішньої якості ПЗ та ролі антипатернів як стійких архітектурно-дизайнерських дисбалансів, що формуються у процесі еволюції проєкту. Показано багатовимірність внутрішньої якості та доведено, що ізольовані правила/метрики або одноканальні ML-підходи не забезпечують достатнього узагальнення, пояснюваності та надійності рішень у різних проєктах і мовах.

На підставі виявлених обмежень сформульовано вимоги до сучасних систем виявлення антипатернів: перехід до інтегрованих гібридних подань коду, багаторівневий аналіз (метод–компонент–проєкт), явне моделювання невизначеності та open-set/low-confidence режим, а також тісна інтеграція з CI/CD і формалізоване оцінювання ефекту рефакторингів у часовій динаміці.

Обґрунтовано та розроблено архітектуру багаторівневої моделі виявлення антипатернів на основі гібридного графового подання Code Property Graph, що узгоджує структурні, семантичні, метричні та еволюційні сигнали й дозволяє відображати комплексну природу антипатернів у промислових кодових базах.

Запропоновано механізми побудови локального, компонентного та проєктного рівнів моделі (гетерогенна GNN з урахуванням типів зв’язків, увага для агрегації локальних представлень, граф взаємодії компонентів з еволюційними атрибутами, ієрархічна увага для узгодження контекстів), що забезпечує більш стійке виявлення антипатернів у складних багатомодульних системах.

Розроблено класифікаційну підсистему з open-set головою та показниками невизначеності (енергетичні/ентропійні), яка реалізує селективне передбачення і зменшує ризик помилкових рішень у нетипових випадках; також запропоновано інкрементальний конвеєр аналізу для pull request-сценаріїв і механізми пояснюваності на основі ваг уваги та внесків каналів ознак.

Сформовано підхід до усунення антипатернів, що переходить від детекції до вибору та обґрунтування рефакторингів і подальшого контролю результату. Рекомендацію рефакторингів (для груп клонів) подано як вибір з дискретного набору альтернатив із можливістю обґрунтованого утримання від змін.

Запропоновано узгоджені моделі оцінювання трудомісткості та вигоди рефакторингу й обґрунтовано необхідність урахування CI-ризику. Рекомендації формалізовано як багатоцільову задачу у просторі «вигода–трудомісткість–ризик» з можливістю скаляризації під пріоритети команди та інтеграції в модуль планування послідовностей змін.

Розроблено та експериментально перевірено інтегрований інструментальний засіб, що реалізує end-to-end цикл «гібридне подання – виявлення – формування/ранжування – планування – процесне оцінювання (EWMA/CUSUM) – звіт». На open-source проєктах підтверджено реальний сукупний ефект: середнє зменшення Technical Debt Index на 16,4%, підвищення Maintainability Index у середньому на 5,66, зниження Cognitive Complexity на 12,1%, зменшення Clone density на 32,0%, а також високу узгодженість прогнозованого та фактичного ефекту ($R^2 \approx 0{,}91$).

Порівняння з наявними засобами показало перевагу саме цілісного рішення: класичні аналізатори здебільшого зосереджені на виявленні проблем, тоді як запропонована система забезпечує повний керований цикл «виявлення –

планування – оцінка» та підтверджує покращення контрольними процедурами, що підвищує практичну цінність у DevOps/CI/CD.

Безпечність автоматизації забезпечено механізмом утримання за низької впевненості: за експертною перевіркою точність рекомендацій становить близько 0,81, а частка випадків контрольованого «утримання» – близько 0,18, що відображає керований компроміс між корисністю та ризиком; оцінка продуктивності підтвердила масштабованість процесу для проєктів різних розмірів.

Дисертацію виконано згідно тематичних планів НДР Національного університету «Одеська політехніка» за період 2024 – 2025 рр. в рамках держбюджетної теми № 237-177 – «Програмні системи машинного навчання та їх застосування в галузі інтелектуального аналізу даних».

Розроблені в дисертаційній роботі моделі та методи отримали впровадження у діяльності науково-виробничого підприємства «КАРЕ», а також знайшли відображення у навчальному процесі та науково-дослідницький діяльності Національного університету «Одеська політехніка».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Bavota G., Russo B. A large-scale empirical study on the effect of code clones on software quality // Information and Software Technology. – 2015. – Vol. 58. – P. 206–222.

2) Palomba F., Bavota G., Di Penta M., Oliveto R., De Lucia A., Poshyvanyk D. On the diffuseness and the impact on maintainability of code smells // Proceedings of the 40th International Conference on Software Engineering. – Gothenburg, Sweden : ACM, 2018. – P. 482–493.

3) Tufano M., Palomba F., Bavota G., Di Penta M., Oliveto R., De Lucia A., Poshyvanyk D. When and why your code starts to smell bad // Proceedings of the 37th IEEE/ACM International Conference on Software Engineering. – Florence, Italy : IEEE, 2015. – P. 403–414.

4) Yamashita A., Moonen L. To what extent can maintenance problems be predicted by code smells? // Journal of Software: Evolution and Process. – 2013. – Vol. 25, No. 7. – P. 571–594.

5) Arcelli Fontana F., Zanoni M., Marino A., Mantyla M. Code smell detection: Towards a machine learning-based approach // Proceedings of the IEEE International Conference on Software Maintenance. – Eindhoven, Netherlands : IEEE, 2013. – P. 396–399.

6) Pecorelli F., Di Nucci D., De Lucia A., De Roover C. A large empirical assessment of the role of data balancing in machine-learning-based code smell detection // Journal of Systems and Software. – 2020. – Vol. 169. – Art. 110693.

7) Santos J., Ferreira T., Ribeiro M., Mendonça M. A systematic literature review on code smells detection and visualization // Archives of Computational Methods in Engineering. – 2021. – Vol. 28. – P. 3275–3317.

8) Sharma T., Spinellis D. A survey on software smells // Journal of Systems and Software. – 2018. – Vol. 138. – P. 158–173.

9) Mockus A., Votta L. G. Identifying reasons for software changes using historic databases // Proceedings of the International Conference on Software Maintenance. – San Jose, USA : IEEE, 2000. – P. 120–130.

10) Kagdi H., Collard M. L., Maletic J. I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution // Journal of Software Maintenance and Evolution: Research and Practice. – 2007. – Vol. 19, No. 2. – P. 77–131.

11) Hassan A. E. The road ahead for mining software repositories // Proceedings of the Frontiers of Software Maintenance. – Beijing, China : IEEE, 2008. – P. 48–57.

12) Zimmermann T., Nagappan N. Predicting defects using network analysis on dependency graphs // Proceedings of the 30th International Conference on Software Engineering. – Leipzig, Germany : ACM, 2008. – P. 531–540.

13) Nagappan N., Ball T. Using software dependencies and churn metrics to predict field failures: An empirical case study // Proceedings of the International Symposium on Software Testing and Analysis. – Portland, USA : ACM, 2005. – P. 364–374.

14) Kim M., Zimmermann T., Whitehead E., Zeller A. Predicting faults from cached history // Proceedings of the 29th International Conference on Software Engineering. – Minneapolis, USA : IEEE, 2007. – P. 489–498.

15) Moser R., Pedrycz W., Succi G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction // Proceedings of the 30th International Conference on Software Engineering. – Leipzig, Germany : ACM, 2008. – P. 181–190.

16) Hodovychenko M. A., Kurinko D. D. Analysis of Existing Approaches to Automated Refactoring of Object-Oriented Software Systems. Herald of Advanced Information Technology. 2025. Vol. 8, № 2. P. 179–196.

17) D’Ambros M., Lanza M., Robbes R. An extensive comparison of bug prediction approaches // Proceedings of the 7th IEEE Working Conference on Mining Software Repositories. – Cape Town, South Africa : IEEE, 2010. – P. 31–41.

18) Курінько Д. Д. Проблеми виявлення потреби у рефакторингу в об’єктно-орієнтованому коді. Innovative Research in Science and Economy: proceedings of the

2nd International Scientific and Practical Conference, Brussels, Belgium, December 3–5, 2025. Brussels: International Scientific Unity, 2025. С. 764–767.

19) Kamei Y., Shihab E., Adams B., Hassan A. E., Mockus A., Sinha A., Ubayashi N. A large-scale empirical study of just-in-time quality assurance // IEEE Transactions on Software Engineering. – 2013. – Vol. 39, No. 6. – P. 757–773.

20) Bird C., Nagappan N., Murphy B., Gall H., Devanbu P. Don't touch my code! Examining the effects of ownership on software quality // Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering. – Szeged, Hungary : ACM, 2011. – P. 4–14.

21) Meneely A., Williams L. Secure open source collaboration: An empirical study of linus' law // Proceedings of the 16th ACM Conference on Computer and Communications Security. – Chicago, USA : ACM, 2009. – P. 453–462.

22) Ostrand T. J., Weyuker E. J., Bell R. M. Predicting the location and number of faults in large software systems // IEEE Transactions on Software Engineering. – 2005. – Vol. 31, No. 4. – P. 340–355.

23) Hall T., Beecham S., Bowes D., Gray D., Counsell S. A systematic literature review on fault prediction performance in software engineering // IEEE Transactions on Software Engineering. – 2012. – Vol. 38, No. 6. – P. 1276–1304.

24) Tantithamthavorn C., McIntosh S., Hassan A. E., Matsumoto K. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models // IEEE Transactions on Software Engineering. – 2018. – Vol. 44, No. 12. – P. 1200–1219.

25) Herbold S., Trautsch A., Grabowski J. A comparative study to benchmark cross-project defect prediction approaches // IEEE Transactions on Software Engineering. – 2018. – Vol. 44, No. 9. – P. 811–833.

26) Shihab E., Mockus A., Kamei Y., Adams B., Hassan A. E. High-impact defects: A study of breakage and surprise defects // Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering. – Raleigh, USA : ACM, 2011. – P. 300–310.

27) Bavota G., De Lucia A., Di Penta M., Oliveto R., Palomba F. An experimental investigation on the innate relationship between quality and refactoring // Journal of Systems and Software. – 2015. – Vol. 107. – P. 1–14.

28) Mens T., Tourwé T. A survey of software refactoring // IEEE Transactions on Software Engineering. – 2004. – Vol. 30, No. 2. – P. 126–139.

29) Murphy-Hill E., Parnin C., Black A. P. How we refactor, and how we know it // IEEE Transactions on Software Engineering. – 2012. – Vol. 38, No. 1. – P. 5–18.

30) Kim M., Notkin D., Grossman D. Automatic inference of structural changes for matching across program versions // Proceedings of the 29th International Conference on Software Engineering. – Minneapolis, USA : IEEE, 2007. – P. 333–343.

31) Baxter I. D., Yahin A., Moura L., Sant'Anna M., Bier L. Clone detection using abstract syntax trees // Proceedings of the International Conference on Software Maintenance. – Bethesda, USA : IEEE, 1998. – P. 368–377.

32) Cordy J. R., Roy C. K. The NiCad clone detector // Proceedings of the IEEE International Conference on Program Comprehension. – Vancouver, Canada : IEEE, 2011. – P. 219–220.

33) Kamiya T., Kusumoto S., Inoue K. CCFinder: A multilinguistic token-based code clone detection system // IEEE Transactions on Software Engineering. – 2002. – Vol. 28, No. 7. – P. 654–670.

34) Krinke J. Identifying similar code with program dependence graphs // Proceedings of the Eighth Working Conference on Reverse Engineering. – Stuttgart, Germany : IEEE, 2001. – P. 301–309.

35) Ferrante J., Ottenstein K. J., Warren J. D. The program dependence graph and its use in optimization // ACM Transactions on Programming Languages and Systems. – 1987. – Vol. 9, No. 3. – P. 319–349.

36) Allen F. E. Control flow analysis // Proceedings of a Symposium on Compiler Optimization. – Urbana-Champaign, USA : ACM, 1970. – P. 1–19.

37) Aho A. V., Sethi R., Ullman J. D. Compilers: principles, techniques, and tools. – Reading : Addison-Wesley, 1986.

38) Muchnick S. S. Advanced compiler design and implementation. – San Francisco : Morgan Kaufmann, 1997.

39) Gabel M., Su Z. A study of the uniqueness of source code // Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. – Santa Fe, USA : ACM, 2010. – P. 147–156.

40) Hindle A., Barr E. T., Gabel M., Su Z., Devanbu P. On the naturalness of software // Proceedings of the 34th International Conference on Software Engineering. – Zurich, Switzerland : IEEE, 2012. – P. 837–847.

41) Allamanis M., Sutton C. Mining source code repositories at massive scale using language modeling // Proceedings of the 10th Working Conference on Mining Software Repositories. – San Francisco, USA : IEEE, 2013. – P. 207–216.

42) Raychev V., Vechev M., Yahav E. Code completion with statistical language models // Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. – Edinburgh, UK : ACM, 2014. – P. 419–428.

43) Raychev V., Bielik P., Vechev M. Probabilistic model for code with decision trees // Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications. – Portland, USA : ACM, 2016. – P. 731–747.

44) Nguyen T. T., Nguyen A. T., Nguyen H. A., Nguyen T. N. A statistical semantic language model for source code // Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering. – Cary, USA : ACM, 2013. – P. 532–542.

45) Mou L., Li G., Zhang L., Wang T., Jin Z. Convolutional neural networks over tree structures for programming language processing // Proceedings of the AAAI Conference on Artificial Intelligence. – Phoenix, USA : AAAI Press, 2016. – P. 1287–1293.

46) Alon U., Zilberstein M., Levy O., Yahav E. code2vec: Learning distributed representations of code // Proceedings of the ACM on Programming Languages. – 2019. – Vol. 3 (POPL). – Art. 40.

47) Alon U., Levy O., Yahav E. code2seq: Generating sequences from structured representations of code // Proceedings of the International Conference on Learning Representations. – New Orleans, USA : OpenReview, 2019.

48) Yamaguchi F., Golde N., Arp D., Rieck K. Modeling and discovering vulnerabilities with code property graphs // Proceedings of the IEEE Symposium on Security and Privacy. – San Jose, USA : IEEE, 2014. – P. 590–604.

49) Reps T., Horwitz S., Sagiv M. Precise interprocedural dataflow analysis via graph reachability // Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. – Paris, France : ACM, 1995. – P. 49–61.

50) Poshyvanyk D., Marcus A. Combining formal concept analysis with information retrieval for concept location in source code // Proceedings of the IEEE International Conference on Software Maintenance. – Amsterdam, Netherlands : IEEE, 2007. – P. 37–48.

51) Kipf T. N., Welling M. Semi-supervised classification with graph convolutional networks // Proceedings of the International Conference on Learning Representations. – Toulon, France : OpenReview, 2017.

52) Hamilton W. L., Ying Z., Leskovec J. Inductive representation learning on large graphs // Advances in Neural Information Processing Systems. – 2017. – Vol. 30. – P. 1024–1034.

53) Velickovic P., Cucurull G., Casanova A., Romero A., Lio P., Bengio Y. Graph attention networks // Proceedings of the International Conference on Learning Representations. – Vancouver, Canada : OpenReview, 2018.

54) Wu Z., Pan S., Chen F., Long G., Zhang C., Yu P. S. A comprehensive survey on graph neural networks // IEEE Transactions on Neural Networks and Learning Systems. – 2021. – Vol. 32, No. 1. – P. 4–24.

55) Zhou J., Cui G., Hu S., Zhang Z., Yang C., Liu Z., Wang L., Li C., Sun M. Graph neural networks: A review of methods and applications // AI Open. – 2020. – Vol. 1. – P. 57–81.

56) Li Z., Zou D., Xu S., Ou X., Jin H., Wang S., Deng Z. VulDeePecker: A deep learning-based system for vulnerability detection // Proceedings of the Network and Distributed System Security Symposium. – San Diego, USA : Internet Society, 2018.

57) Lin G., Zhang J., Luo W., Pan L., Xiang Y., De Vel O. Cross-project transfer representation learning for vulnerable function discovery // IEEE Transactions on Industrial Informatics. – 2019. – Vol. 15, No. 6. – P. 3289–3297.

58) Wang S., Liu T., Tan L. Automatically learning semantic features for defect prediction // Proceedings of the 38th International Conference on Software Engineering. – Austin, USA : ACM, 2016. – P. 297–308.

59) Dam H. K., Tran T., Ghose A. Automatic feature learning for predicting vulnerable software components // IEEE Transactions on Software Engineering. – 2017. – Vol. 43, No. 10. – P. 913–932.

60) Li X., Liu X., Zhang H., Yang Q., Xie T. Deep learning-based vulnerability detection: A survey // ACM Computing Surveys. – 2022. – Vol. 54, No. 9. – Art. 187.

61) Zhou Y., Sharma A., Chen Y., Xu B., Zhang L. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks // Advances in Neural Information Processing Systems. – 2019. – Vol. 32. – P. 10197–10207.

62) Nguyen V. T., Nguyen T. D., Phan H. D., Nguyen T. N. Graph-based deep learning for vulnerability detection // Proceedings of the 26th ACM SIGSOFT International Symposium on Foundations of Software Engineering. – Lake Buena Vista, USA : ACM, 2018. – P. 785–796.

63) Allamanis M., Brockschmidt M., Khademi M. Learning to represent programs with graphs // Proceedings of the International Conference on Learning Representations. – Vancouver, Canada : OpenReview, 2018.

64) Hellendoorn V. J., Devanbu P. Are deep neural networks the best choice for modeling source code? // Proceedings of the 2017 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. – Paderborn, Germany : ACM, 2017. – P. 763–773.

65) White M., Tufano M., Vendome C., Poshyvanyk D. Deep learning code fragments for code clone detection // Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. – Singapore : IEEE, 2016. – P. 87–98.

66) Pradel M., Sen K. DeepBugs: A learning approach to name-based bug detection // Proceedings of the ACM on Programming Languages. – 2018. – Vol. 2 (OOPSLA). – Art. 147.

67) Chen Z., Kommrusch S., Monperrus M. A neural network-based approach for automatic detection of performance bugs // Proceedings of the 26th ACM SIGSOFT International Symposium on Foundations of Software Engineering. – Lake Buena Vista, USA : ACM, 2018. – P. 564–575.

68) Xu Z., Chen Z., Hu X., Wang J., Wang J., Li S. Pattern-based defect prediction using graph mining // Journal of Systems and Software. – 2019. – Vol. 153. – P. 121–136.

69) Fan Y., Yu L., Wang S., Chen J. A survey on software defect prediction with deep learning // ACM Computing Surveys. – 2023. – Vol. 55, No. 11. – Art. 234.

70) Zhang F., Kamei Y., Shihab E., Hassan A. E. Studying the noise in defect datasets: Its impact on classifier performance and how to handle it // Empirical Software Engineering. – 2022. – Vol. 27, No. 4. – Art. 89.

71) Hüllermeier E., Waegeman W. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods // Machine Learning. – 2021. – Vol. 110, No. 3. – P. 457–506.

72) Gal Y., Ghahramani Z. Dropout as a Bayesian approximation: representing model uncertainty in deep learning // Proceedings of the 33rd International Conference on Machine Learning. – New York, USA : PMLR, 2016. – P. 1050–1059.

73) Kendall A., Gal Y. What uncertainties do we need in Bayesian deep learning for computer vision? // Advances in Neural Information Processing Systems. – 2017. – Vol. 30. – P. 5574–5584.

74) Lakshminarayanan B., Pritzel A., Blundell C. Simple and scalable predictive uncertainty estimation using deep ensembles // Advances in Neural Information Processing Systems. – 2017. – Vol. 30. – P. 6402–6413.

75) El-Yaniv R., Wiener Y. On the foundations of selective classification // Journal of Machine Learning Research. – 2010. – Vol. 11. – P. 1605–1641.

76) Geifman Y., El-Yaniv R. Selective classification for deep neural networks // Advances in Neural Information Processing Systems. – 2017. – Vol. 30. – P. 4878–4887.

77) Chow C. K. On optimum recognition error and reject tradeoff // IEEE Transactions on Information Theory. – 1970. – Vol. 16, No. 1. – P. 41–46.

78) Cortes C., DeSalvo G., Mohri M. Boosting with abstention // Advances in Neural Information Processing Systems. – 2016. – Vol. 29. – P. 1660–1668.

79) Grandvalet Y., Bengio Y. Confidence estimation for neural networks // Advances in Neural Information Processing Systems. – 2004. – Vol. 17. – P. 421–428.

80) Kurinko D. D. Hybrid Graphs for Code Smells: A Multi-Level Model for Anti-Pattern Detection in Software Components. Applied Aspects of Information Technology. 2025. Vol. 8, № 3. P. 274–285.

81) Kuleshov V., Fenner N., Ermon S. Accurate uncertainties for deep learning using calibrated regression // Proceedings of the 35th International Conference on Machine Learning. – Stockholm, Sweden : PMLR, 2018. – P. 2796–2804.

82) Depeweg S., Hernández-Lobato J. M., Doshi-Velez F., Udluft S. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning // Proceedings of the 35th International Conference on Machine Learning. – Stockholm, Sweden : PMLR, 2018. – P. 1184–1193.

83) Курінько Д. Д. Інтегрований підхід до виявлення рефакторингу в ООП-системах. The Integration of Research, Innovation and Economy: proceedings of the 1st International Scientific and Practical Conference, Seville, Spain, October 8–10, 2025. Seville: International Scientific Unity, 2025. С. 26–30

84) Курінько Д. Д. Селективне виявлення антипатернів з open-set-головками та оцінкою невизначеності. Modern Science: Trends, Challenges, Solutions: proceedings of the 5th International Scientific and Practical Conference, Liverpool, United Kingdom, December 11–13, 2025. Liverpool: Cognum Publishing House, 2025. С. 249–252.

85) Saito T., Rehmsmeier M. The precision–recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets // PLOS ONE. – 2015. – Vol. 10, No. 3.

86) Kurinko D. D., Kryvda V. I. Multimodal Graph Representations for Reliable Anti-Pattern Detection in Evolving Codebases. Informatics. Culture. Technology. 2025. Vol. 2, № 2. P. 294–299.

87) Provost F., Fawcett T. Robust classification for imprecise environments // Machine Learning. – 2001. – Vol. 42, No. 3. – P. 203–231.

88) Domingos P. A unified bias-variance decomposition // Proceedings of the 17th International Conference on Machine Learning. – Stanford, USA : Morgan Kaufmann, 2000. – P. 231–238.

89) Dietterich T. Overfitting and undercomputing in machine learning // ACM Computing Surveys. – 1995. – Vol. 27, No. 3. – P. 326–327.

90) Vapnik V. N. Statistical learning theory. – New York : Wiley, 1998.

91) Box G. E. P., Jenkins G. M., Reinsel G. C., Ljung G. M. Time series analysis: forecasting and control. – 5th ed. – Hoboken : Wiley, 2015.

92) Hyndman R. J., Athanasopoulos G. Forecasting: principles and practice. – 3rd ed. – Melbourne : OTexts, 2021.

93) Bernal J. L., Cummins S., Gasparrini A. Interrupted time series regression for the evaluation of public health interventions: a tutorial // International Journal of Epidemiology. – 2017. – Vol. 46, No. 1. – P. 348–355.

94) Brodersen K. H., Gallusser F., Koehler J., Remy N., Scott S. L. Inferring causal impact using Bayesian structural time-series models // Annals of Applied Statistics. – 2015. – Vol. 9, No. 1. – P. 247–274.

95) Abadie A. Semiparametric difference-in-differences estimators // Review of Economic Studies. – 2005. – Vol. 72, No. 1. – P. 1–19.

96) Курінько Д. Д. Інкрементальне оновлення Code Property Graph для прискорення виявлення антипатернів у CI/CD-конвеєрах. Conceptual Framework and Dynamics of the Development of Science: abstracts of XVI International Scientific and Practical Conference, Munich, Germany, 2025. С. 283–287.

97) Angrist J. D., Pischke J.-S. Mostly harmless econometrics: an empiricist's companion. – Princeton : Princeton University Press, 2009.

98) Imbens G. W., Rubin D. B. Causal inference for statistics, social, and biomedical sciences. – Cambridge : Cambridge University Press, 2015.

99) Pearl J. Causality: models, reasoning, and inference. – 2nd ed. – Cambridge : Cambridge University Press, 2009.

100) Hernán M. A., Robins J. M. Causal inference: what if. – Boca Raton : Chapman & Hall/CRC, 2020.

101) Athey S., Imbens G. Machine learning methods for estimating heterogeneous causal effects // Statistical Science. – 2016. – Vol. 31, No. 4. – P. 1–26.

102) Montgomery D. C. Introduction to statistical quality control. – 8th ed. – Hoboken : Wiley, 2019.

103) Page E. S. Continuous inspection schemes // Biometrika. – 1954. – Vol. 41, No. 1–2. – P. 100–115.

104) Roberts S. W. Control chart tests based on geometric moving averages // Technometrics. – 1959. – Vol. 1, No. 3. – P. 239–250.

105) Lucas J. M., Saccucci M. S. Exponentially weighted moving average control schemes: properties and enhancements // Technometrics. – 1990. – Vol. 32, No. 1. – P. 1–12.

106) Hawkins D. M., Olwell D. H. Cumulative sum charts and charting for quality improvement. – New York : Springer, 1998.

107) Alwan L. C., Roberts H. V. Time series modeling for statistical process control // Journal of Business and Economic Statistics. – 1988. – Vol. 6, No. 1. – P. 87–95.

108) Montgomery D. C., Mastrangelo C. M. Some statistical process control methods for autocorrelated data // Journal of Quality Technology. – 1991. – Vol. 23. – P. 179–193.

109) Nelson L. S. The Shewhart control chart–tests for special causes // Journal of Quality Technology. – 1984. – Vol. 16. – P. 237–239.

110) Capizzi G., Masarotto G. Adaptive statistical process control: an overview and some remarks // Quality and Reliability Engineering International. – 2010. – Vol. 26, No. 7. – P. 697–713.

111) Coello Coello C. A. A comprehensive survey of evolutionary-based multiobjective optimization techniques // Knowledge and Information Systems. – 1999. – Vol. 1, No. 3. – P. 269–308.

112) Deb K. Multi-objective optimization using evolutionary algorithms. – Chichester : Wiley, 2001.

113) Deb K., Pratap A., Agarwal S., Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II // IEEE Transactions on Evolutionary Computation. – 2002. – Vol. 6, No. 2. – P. 182–197.

114) Zitzler E., Thiele L. Multiobjective optimization using evolutionary algorithms: a comparative case study // Parallel Problem Solving from Nature. – Berlin : Springer, 1998. – P. 292–301.

115) Zitzler E., Laumanns M., Thiele L. SPEA2: Improving the strength Pareto evolutionary algorithm // Technical Report. – Zurich : ETH Zurich, 2001.

116) Van Veldhuizen D. A., Lamont G. B. Multiobjective evolutionary algorithm research: A history and analysis // Technical Report. – Dayton : Air Force Institute of Technology, 1998.

117) Fonseca C. M., Fleming P. J. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization // Proceedings of the 5th International Conference on Genetic Algorithms. – Urbana-Champaign, USA : Morgan Kaufmann, 1993. – P. 416–423.

118) Knowles J., Corne D. On metrics for comparing nondominated sets // Proceedings of the Congress on Evolutionary Computation. – Honolulu, USA : IEEE, 2002. – P. 711–716.

119) Zitzler E., Thiele L., Laumanns M., Fonseca C. M., da Fonseca V. G. Performance assessment of multiobjective optimizers: an analysis and review // IEEE Transactions on Evolutionary Computation. – 2003. – Vol. 7, No. 2. – P. 117–132.

120) Ruiz C., Riquelme J. C., Aguilar-Ruiz J. S. Incremental wrapper-based gene selection from microarray data for cancer classification // Pattern Recognition. – 2006. – Vol. 39, No. 12. – P. 2383–2392.

121) Russell S., Norvig P. Artificial intelligence: a modern approach. – 4th ed. – Harlow : Pearson, 2020.

122) Geffner H., Bonet B. A concise introduction to models and methods for automated planning. – San Rafael : Morgan & Claypool, 2013.

123) Helmert M. The fast downward planning system // Journal of Artificial Intelligence Research. – 2006. – Vol. 26. – P. 191–246.

124) Ghallab M., Nau D., Traverso P. Automated planning and acting. – Cambridge : Cambridge University Press, 2016.

125) Bertsimas D., Kallus N. From predictive to prescriptive analytics // Management Science. – 2020. – Vol. 66, No. 3. – P. 1025–1044.

126) Amatriain X., Basilico J. Recommender systems in industry: a practical introduction // ACM Queue. – 2015. – Vol. 13, No. 6. – P. 30–48.

127) Kurinko D. D., Kryvda V. I. Uncertainty-Aware Multi-Objective Refactoring for Code Duplication. Herald of Advanced Information Technology. 2025. Vol. 8, № 3. P. 301–315.

128) Li Y., Wang H., Zhang Q. Uncertainty-aware refactoring recommendation under budget constraints // Proceedings of the ACM/SIGAPP Symposium on Applied Computing. – Marrakech, Morocco : ACM, 2024. – P. 1243–1252.

129) Ruhe G., Saliu M. O. The art and science of software release planning // IEEE Software. – 2005. – Vol. 22, No. 6. – P. 47–53.

130) Menzies T., Zimmermann T. Software analytics: so what? // IEEE Software. – 2013. – Vol. 30, No. 4. – P. 31–37.

131) Humble J., Farley D. Continuous delivery: reliable software releases through build, test, and deployment automation. – Boston : Addison-Wesley, 2011.

132) Курінько Д. Д., Кривда В. І. Від комітів до причинності: побудова маловпливових послідовностей рефакторингів через причинно-наслідковий аналіз

репозиторіїв. Наука і техніка сьогодні. Серія «Техніка». 2025. Т. 52, № 11. С. 1752–1773.

133) Forsgren N., Humble J., Kim G. Accelerate: the science of lean software and DevOps: building and scaling high performing technology organizations. – Portland : IT Revolution Press, 2018.

134) Bass L., Weber I., Zhu L. DevOps: a software architect’s perspective. – Boston : Addison-Wesley, 2015.

135) Erich F. M. A., Amrit C., Daneva M. A qualitative study of DevOps usage in practice // Journal of Software: Evolution and Process. – 2017. – Vol. 29, No. 6.

136) Shahin M., Babar M. A., Zhu L. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices // IEEE Access. – 2017. – Vol. 5. – P. 3909–3943.

137) Chen L. Continuous delivery: huge benefits, but challenges too // IEEE Software. – 2015. – Vol. 32, No. 2. – P. 50–54.

138) Rahman F., Rigby P. C., Devanbu P. The impact of failing, flaky, and high-failure tests on the number of crashes // Proceedings of the 11th Working Conference on Mining Software Repositories. – Hyderabad, India : IEEE, 2014. – P. 62–71.

139) Курінько Д. Д. Маловпливові послідовності рефакторингів: причинно-орієнтований підхід до аналізу історій репозиторіїв. The impact of modern digital technologies on the future of professions: abstracts of XV International Scientific and Practical Conference, Sofia, Bulgaria, 2025. С. 264–267.

140) Vasilescu B., van Schuylenburg S., Wulms J., Serebrenik A., van den Brand M. Continuous integration in a social-coding world: empirical evidence from GitHub // Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution. – Victoria, Canada : IEEE, 2014. – P. 401–405.

141) Zampetti F., Scalabrino S., Oliveto R., Di Penta M., Canfora G. How open source projects use static code analysis tools in continuous integration pipelines // Proceedings of the 40th International Conference on Software Engineering. – Gothenburg, Sweden : ACM, 2018. – P. 302–312.

142) Lenarduzzi V., Taibi D., Huttunen H. Are SonarQube rules inducing bugs? // Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering. – London, UK : IEEE, 2020. – P. 203–212.

143) Beller M., Gousios G., Panichella A., Zaidman A. When, how, and why developers (do not) test in continuous integration // Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. – Bergamo, Italy : ACM, 2015. – P. 373–384.

144) Zhao Y., Serebrenik A., Vasilescu B., Filkov V. The impact of continuous integration on other software development practices // Proceedings of the 32nd IEEE International Conference on Software Maintenance and Evolution. – Raleigh, USA : IEEE, 2016. – P. 60–70.

145) Ståhl D., Bosch J. Modeling continuous integration practice differences in industry software development // Journal of Systems and Software. – 2014. – Vol. 87. – P. 48–59.

146) Rahman F., Khatri S., Devanbu P. Detecting and characterizing flaky tests: empirical study // Proceedings of the 9th Joint Meeting on Foundations of Software Engineering. – Saint Petersburg, Russia : ACM, 2017. – P. 559–570.

147) Vassallo C., Palomba F., Bacchelli A., Gall H. Continuous code quality: are we (really) doing it? // IEEE Software. – 2018. – Vol. 35, No. 4. – P. 82–88.

148) Leppänen V., Mäntylä M. V., Mäkinen S. Safe automation of code changes in continuous deployment pipelines // Journal of Systems and Software. – 2023. – Vol. 203.

149) Hilton M., Nelson N., Tunnell T., Marinov D., Dig D. Trade-offs in continuous integration: assurance, security, and flexibility // Proceedings of the 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering. – Lake Buena Vista, USA : ACM, 2017. – P. 197–207.

150) Olsson H. H., Alahyari H., Bosch J. Climbing the stairway to heaven: a multiple-case study exploring barriers in the transition from agile development towards continuous deployment // Information and Software Technology. – 2012. – Vol. 54, No. 9. – P. 887–905.

151) Lehman M. M. Laws of software evolution revisited // Proceedings of the European Workshop on Software Process Technology. – Berlin : Springer, 1996. – P. 108–124.

152) Lehman M. M., Ramil J. F. Software evolution: background, theory, practice // Information Processing Letters. – 2003. – Vol. 88, No. 1–2. – P. 33–44.

153) Mens T., Demeyer S. Software evolution. – Berlin : Springer, 2008.

154) Godfrey M. W., German D. M. The past, present, and future of software evolution // Proceedings of the IEEE International Conference on Software Maintenance. – Amsterdam, Netherlands : IEEE, 2008. – P. 129–138.

155) Lanza M., Marinescu R. Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. – Berlin : Springer, 2006.

156) Marinescu R. Detection strategies: metrics-based rules for detecting design flaws // Proceedings of the 20th IEEE International Conference on Software Maintenance. – Chicago, USA : IEEE, 2004. – P. 350–359.

157) Fowler M. Refactoring: improving the design of existing code. – 2nd ed. – Boston : Addison-Wesley, 2018.

158) Brown N., Cai Y., Guo Y., Kazman R., Kim M., Kruchten P., Lim E., MacCormack A., Nord R., Ozkaya I., Sangwan R., Seaman C., Sullivan K., Zazworka N. Managing technical debt in software-reliant systems // Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. – Santa Fe, USA : ACM, 2010. – P. 47–52.

159) Kruchten P., Nord R. L., Ozkaya I. Technical debt: from metaphor to theory and practice // IEEE Software. – 2012. – Vol. 29, No. 6. – P. 18–21.

160) Avgeriou P., Kruchten P., Ozkaya I., Seaman C. Managing technical debt in software engineering. – Dagstuhl : Dagstuhl Reports, 2016.

161) Alves N. S. R., Mendes T. S., de Mendonça M. G., Spínola R. O., Shull F., Zazworka N. Identification and management of technical debt: a systematic mapping study // Information and Software Technology. – 2016. – Vol. 70. – P. 100–121.

162) Rios N., Spínola R. O., Seaman C., Zazworka N. Managing technical debt: a systematic mapping study // Information and Software Technology. – 2018. – Vol. 103. – P. 73–92.

163) Arcelli Fontana F., Zanoni M. Code smell severity classification using machine learning techniques // Knowledge-Based Systems. – 2017. – Vol. 128. – P. 43–58.

164) Arcelli Fontana F., Braione P., Zanoni M. Automatic detection of bad smells in code: an experimental assessment // Journal of Object Technology. – 2012. – Vol. 11, No. 2. – P. 5:1–5:38.

165) Yamashita A., Moonen L. Do code smells reflect maintainability? // Proceedings of the IEEE International Conference on Software Maintenance. – Timisoara, Romania : IEEE, 2012. – P. 306–315.

166) Moha N., Guéhéneuc Y.-G., Duchien L., Le Meur A.-F. DECOR: A method for the specification and detection of code and design smells // IEEE Transactions on Software Engineering. – 2010. – Vol. 36, № 1. – P. 20–36.

167) Khomh F., Di Penta M., Guéhéneuc Y.-G. An exploratory study of the impact of code smells on software change-proneness // Proceedings of the 16th Working Conference on Reverse Engineering. – Lille, France : IEEE, 2009. – P. 75–84.

168) Mo R., Cai Y., Kazman R. Architecture anti-patterns in practice: a case study // Proceedings of the IEEE International Conference on Software Architecture. – Gothenburg, Sweden : IEEE, 2016. – P. 209–218.

169) Garcia J., Popescu D., Edwards G., Medvidovic N. Identifying architectural bad smells // Proceedings of the 13th European Conference on Software Maintenance and Reengineering. – Kaiserslautern, Germany : IEEE, 2009. – P. 255–258.

170) Suryanarayana G., Samarthyam G., Sharma T. Refactoring for software design smells: managing technical debt. – San Francisco : Morgan Kaufmann, 2014.

171) Allamanis M., Barr E. T., Devanbu P., Sutton C. A survey of machine learning for big code and naturalness // ACM Computing Surveys. – 2018. – Vol. 51, No. 4.

172) Monperrus M. A critical review of “Automatic Patch Generation Learned from Human-Written Patches” // IEEE Transactions on Software Engineering. – 2018. – Vol. 44, No. 11. – P. 1025–1044.

173) Ray B., Hellendoorn V., Godhane S., Tu Z. On the naturalness of buggy code // Proceedings of the IEEE/ACM International Conference on Software Engineering. – Gothenburg, Sweden : IEEE/ACM, 2018. – P. 428–439.

174) Allamanis M., Barr E. T., Bird C., Sutton C. Learning natural coding conventions // Proceedings of the 22nd ACM SIGSOFT Symposium on Foundations of Software Engineering. – Hong Kong : ACM, 2014. – P. 281–293.

175) Nguyen T. T., Nguyen A. T., Nguyen H. A., Nguyen T. N. A statistical semantic language model for source code // Proceedings of the 9th Joint Meeting on Foundations of Software Engineering. – Szeged, Hungary : ACM, 2013. – P. 532–542.

176) Ray B., Posnett D., Devanbu P., Filkov V. A large-scale study of programming languages and code quality in GitHub // Communications of the ACM. – 2017. – Vol. 60, No. 10. – P. 91–100.

177) Giger E., D’Ambros M., Pinzger M., Gall H. Predicting fault-prone Java classes // Proceedings of the 1st International Workshop on Mining Software Repositories. – Edinburgh, UK : IEEE, 2004. – P. 1–5.

178) Gyimóthy T., Ferenc R., Siket I. Empirical validation of object-oriented metrics on open source software for fault prediction // IEEE Transactions on Software Engineering. – 2005. – Vol. 31, № 10. – P. 897–910.

179) Menzies T., Greenwald J., Frank A. Data mining static code attributes to learn defect predictors // IEEE Transactions on Software Engineering. – 2007. – Vol. 33, No. 1. – P. 2–13.

180) Hassan A. E., Holt R. C. Predicting change propagation in software systems // Proceedings of the IEEE International Conference on Software Maintenance. – Montreal, Canada : IEEE, 2004. – P. 284–293.

181) Canfora G., Cerulo L., Di Penta M. Mining repositories to support software maintenance // Proceedings of the IEEE International Conference on Software Maintenance. – Beijing, China : IEEE, 2008. – P. 485–488.

182) Hassan A. E. Predicting faults using the complexity of code changes // Proceedings of the 31st International Conference on Software Engineering. – Vancouver, Canada : IEEE, 2009. – P. 78–88.

183) Bird C., Devanbu P., Gall H. Ownership and quality in software projects // Proceedings of the 16th ACM SIGSOFT Symposium on Foundations of Software Engineering. – Atlanta, USA : ACM, 2008. – P. 4–14.

184) Курінько Д. Д. Модель оцінювання ефективності рефакторингів на основі статистичного контролю. Наука і техніка сьогодні. Серія «Техніка». 2025. Т. 50, № 9. С. 1265–1280.

185) Kalliamvakou E., Gousios G., Blincoe K., Singer L., German D. M., Damian D. The promises and perils of mining GitHub // Proceedings of the 11th Working Conference on Mining Software Repositories. – Hyderabad, India : IEEE, 2014. – P. 92–101.

186) Bacchelli A., Bird C. Expectations, outcomes, and challenges of modern code review // Proceedings of the 35th International Conference on Software Engineering. – San Francisco, USA : IEEE/ACM, 2013. – P. 712–721.

187) Rigby P. C., Storey M.-A. The impact of modern code review practices on software quality // Empirical Software Engineering. – 2013. – Vol. 18, No. 6. – P. 1228–1265.

188) Zeller A. Why programs fail: a guide to systematic debugging. – 2nd ed. – San Francisco : Morgan Kaufmann, 2009.

189) Murphy-Hill E., Black A. P. An interactive ambient visualization for code smells // Proceedings of the 5th International Symposium on Software Visualization. – Salt Lake City, USA : ACM, 2010. – P. 5–14.

190) Kim M., Zimmermann T., DeLine R., Begel A. The emerging role of data scientists on software development teams // Proceedings of the 38th International Conference on Software Engineering. – Austin, USA : IEEE/ACM, 2016. – P. 96–107.

191) Rahman F., Khatri S., Devanbu P. Detecting and characterizing flaky tests // Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. – Bergamo, Italy : ACM, 2015. – P. 559–570.

192) Vassallo C., Palomba F., Gall H., Bacchelli A. Continuous code quality: are we (really) doing it? // IEEE Software. – 2018. – Vol. 35, No. 4. – P. 82–88.

193) Scalabrino S., Linares-Vásquez M., Poshyvanyk D., Oliveto R., Di Penta M., De Lucia A. Automatically assessing code understandability // IEEE Transactions on Software Engineering. – 2019. – Vol. 45, No. 9. – P. 899–924.

194) Bavota G., Russo B., Di Penta M., Oliveto R., Poshyvanyk D., De Lucia A. An empirical study on the developers' perception of software smells // Proceedings of the International Conference on Software Engineering. – Zurich, Switzerland : IEEE/ACM, 2013. – P. 482–493.

195) Palomba F., Zaidman A., Oliveto R., De Lucia A. An exploratory study on the relationship between changes and refactoring // Proceedings of the IEEE International Conference on Software Maintenance. – Trento, Italy : IEEE, 2014. – P. 56–65.

196) Tufano M., Palomba F., Bavota G., Oliveto R., Di Penta M., De Lucia A. An empirical investigation into the nature of test smells // Proceedings of the IEEE International Conference on Software Maintenance and Evolution. – Bremen, Germany : IEEE, 2016. – P. 1–10.

197) Sharma T., Fragkoulis M., Spinellis D. Does your configuration file smell? // Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering. – Montpellier, France : IEEE, 2018. – P. 189–200.

198) Spinellis D. Code quality: the open source perspective. – Boston : Addison-Wesley, 2006.

199) Opdyke W. F. Refactoring object-oriented frameworks : PhD thesis. – Urbana-Champaign : University of Illinois at Urbana-Champaign, 1992.

200) Dig D., Johnson R. E. How do APIs evolve? A story of refactoring // Journal of Software Maintenance and Evolution. – 2006. – Vol. 18, No. 2. – P. 83–107.

201) Tsantalis N., Chaikalis T., Chatzigeorgiou A. JDeodorant: identification and application of extract class refactorings // Proceedings of the IEEE International Conference on Software Maintenance. – Beijing, China : IEEE, 2008. – P. 365–374.

202) Ouni A., Kessentini M., Sahraoui H., Inoue K., Deb K. Multi-criteria code refactoring using search-based software engineering // IEEE Transactions on Software Engineering. – 2016. – Vol. 42, No. 4. – P. 373–389.

203) Harman M., Jones B. F. Search-based software engineering // Information and Software Technology. – 2001. – Vol. 43, No. 14. – P. 833–839.

204) Harman M., McMinn P., De Souza J. T., Yoo S. Search-based software engineering: techniques, taxonomy, tutorial // Empirical Software Engineering. – 2012. – Vol. 17, No. 1–2. – P. 43–95.

205) Feldt R., Magazinius A. Validity threats in empirical software engineering research: an initial survey // Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering. – Redwood City, USA : ACM, 2010. – P. 374–379.

206) Kitchenham B., Charters S. Guidelines for performing systematic literature reviews in software engineering // Technical Report EBSE-2007-01. – Keele University, 2007.

207) Wohlin C., Runeson P., Höst M., Ohlsson M. C., Regnell B., Wesslén A. Experimentation in software engineering. – Berlin : Springer, 2012.

208) Petersen K., Feldt R., Systematic mapping studies in software engineering // Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. – Bari, Italy : British Computer Society, 2008. – P. 68–77.

209) Basili V. R., Caldiera G., Rombach H. D. The goal question metric approach // Encyclopedia of Software Engineering. – New York : Wiley, 1994. – P. 528–532.

210) Fenton N. E., Bieman J. M. Software metrics: a rigorous and practical approach. – 3rd ed. – Boca Raton : CRC Press, 2014.

211) Mockus A. Engineering big data solutions for software analytics // Proceedings of the IEEE International Conference on Software Maintenance and Evolution. – Bremen, Germany : IEEE, 2016. – P. 1–4.

212) Nagappan N., Shihab E., Menzies T. Future trends in software engineering analytics // Proceedings of the IEEE International Conference on Software Maintenance and Evolution. – Bremen, Germany : IEEE, 2016. – P. 1–10.

ДОДАТОК А СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

1) Годовиченко М.А.; Курінько Д.Д. "Аналіз існуючих підходів до автоматизації рефакторингу об’єктно-орієнтованих програмних систем" Publ. Nauka i Tekhnika. Odesa: Ukraine. Herald of Advanced Information Technology 8 (2), 179-196. https://doi.org/10.15276/hait.08.2025.11. (Index Scopus).

https://hait.op.edu.ua/index.php/journal/article/view/187

2) Курінько Д.Д. Модель оцінювання ефективності рефакторингів на основі статистичного контролю // Наука і техніка сьогодні. Серія «Техніка». – 2025. – Т. 50, № 9. – С. 1265–1280. – DOI: https://doi.org/10.52058/2786-6025-2025-9(50)-1265-1280. (Index Copernicus)

https://perspectives.pp.ua/index.php/nts/article/view/29453

3) Курінько Д.Д.; Кривда В.І. "Рекомендація рефакторингів із багатоцільовою оптимізацією та урахуванням невизначеності для дублювання коду" Publ. Nauka i Tekhnika. Odesa: Ukraine. Herald of Advanced Information Technology 8 (3), 301–315. https://doi.org/10.15276/hait.08.2025.19 (Index Scopus).

https://hait.op.edu.ua/index.php/journal/article/view/209

4) Курінько Д.Д.; "Гібридні графи для запахів коду: багаторівнева модель виявлення антипатернів у програмних компонентах" Publ. Nauka i Tekhnika. Odesa: Ukraine. AAIT 8 (3), 274–285. https://doi.org/10.15276/aait.08.2025.18 (Index Copernicus).

https://aait.op.edu.ua/index.php/journal/article/view/201

5) Курінько Д.Д.; Кривда В.І. «Від комітів до причинності: побудова маловпливових послідовностей рефакторингів черезпричинно-наслідковий аналіз репозиторіїв»// Наука і техніка сьогодні. Серія «Техніка». – 2025. – Т. 52, № 11. – С. 1752–1773. – DOI: https://doi.org/10.52058/2786-6025-2025-11(52)-1752-1773 . (Index Copernicus).

https://perspectives.pp.ua/index.php/nts/article/view/32491

6) Курінько Д.Д. Інтегрований підхід до виявлення рефакторингу в ООП-системах // Modern The Integration of Research, Innovation and Economy : Proceedings of the 1st International Scientific and Practical Conference. – Seville, Spain : International Scientific Unity, October 8–10, 2025. – P. 26–30.

https://isu-conference.com/en/archive/the-integration-of-research-innovation-and-economy-08-10-25/

7) Курінько Д.Д., Кривда В.І. Мультимодальні графові подання для надійного виявлення антипатернів в еволюційних кодових базах // Інформатика. Культура. Техніка. – 2025. – Т. 2, № 2. – С. 294–299. – DOI: https://doi.org/10.15276/ict.02.2025.45 .

https://ict.op.edu.ua/index.php/journal/article/view/56

8) Курінько Д.Д. Проблеми виявлення потреби у рефакторингу в об’єктно-орієнтованому коді // Innovative Research in Science and Economy : Proceedings of the 2nd International Scientific and Practical Conference. – Brussels, Belgium : International Scientific Unity, December 3–5, 2025. – P. 764–767.

https://isu-conference.com/en/archive/innovative-research-in-science-and-economy-03-12-25/

9) Курінько Д.Д. Маловпливові послідовності рефакторингів: причинно-орієнтований підхід до аналізу історій репозиторіїв // Abstracts of XV International Scientific and Practical Conference. – Sofia, Bulgaria. – P. 264–267.

https://eu-conf.com/en/events/the-impact-of-modern-digital-technologies-on-the-future-of-professions/

10) Курінько Д.Д. Селективне виявлення антипатернів з open-set-головками та оцінкою невизначеності // Proceedings of the 5th International Scientific and Practical Conference. – Liverpool, United Kingdom : Cognum Publishing House, 2025. – P. 249–252.

https://sci-conf.com.ua/v-mizhnarodna-naukovo-praktichna-konferentsiya-modern-science-trends-challenges-solutions-11-13-12-2025-liverpul-velikobritaniya-arhiv/

11) Курінько Д.Д. Інкрементальне оновлення Code Property Graph для прискорення виявлення антипатернів у CI/CD-конвеєрах // Abstracts of XVI International Scientific and Practical Conference. – Munich, Germany. – P. 283–287.

https://eu-conf.com/en/events/conceptual-framework-and-dynamics-of-the-development-of-science/

ДОДАТОК Б АКТ ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ У НВП «КАРЕ»

ПП НВП "КАРЕ"

НАУКОВО-ВИРОБНИЧО ПІДПРИЄМСТВО
«КАРЕ»
www.kareod.com

Україна, м. Одеса, вул. Ів. Франка, 55, тел. 380-765-20-14, **380-50-391-70-45**, kareod@i.ua
р/рах № UA 39 325365 0000002600001526800 в ПАТ «КРЕДОБАНК», м. Одеса, МФО 325365, ОКПО 37351868

«ЗАТВЕРДЖУЮ»

«19» 12 2025 року

АКТ

впровадження результатів дисертаційного дослідження

Курінька Дмитра Дмитровича на тему:

«Моделі та методи машинного навчання для виявлення та усунення антипатернів в програмних компонентах»

Розроблені в межах дисертаційного дослідження програмні інструменти, що поєднують методи машинного навчання, графового аналізу коду та автоматизованих стратегій підтримки рефакторингу, були адаптовані та впроваджені у практичну діяльність компанії.

Створений інструментарій забезпечує комплексне оцінювання внутрішньої якості програмних компонентів, виявлення антипатернів різних типів, формування рекомендацій щодо покращення структури коду та аналіз ефективності застосованих змін. Завдяки інтеграції гібридного графового подання коду, семантичних моделей та історичних даних із системи контролю версій, інструменти дозволяють точно і стабільно виявляти проблемні ділянки у багатомодульних, еволюційних програмних проєктах.

Розроблені програмні засоби були включені у внутрішні процеси статичного аналізу та технічного аудиту, що дало можливість частково автоматизувати виявлення структурних недоліків, дублювання логіки та інших антипатернів. Інструменти також підтримують формування узгоджених і мінімально інвазивних послідовностей рефакторингів, що враховують обмеження CI/CD, особливості архітектури та ризики для продуктивних систем. Використання методів оцінювання ефективності рефакторингів на основі статистичного контролю процесів надало компанії можливість відстежувати вплив внесених змін на метрики підтримуваності, стабільності тестів та загальної якості коду з використанням довірчих інтервалів та адаптивних контрольних меж.

Практичне впровадження результатів дисертаційного дослідження сприяло зменшенню обсягу ручної роботи під час рев'ю коду, покращенню керованості технічного боргу, підвищенню прозорості та обґрунтованості рішень щодо реструктуризації програмних компонентів. Компанія отримала ефективний інструмент для підтримки розробників і технічних лідерів у процесах модернізації коду, що позитивно вплинуло на якість продуктів та стабільність розробки.

Керівник підприємства Леонід Бурдика

ДОДАТОК В ДОВІДКА ПРО ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ У НАВЧАЛЬНОМУ ПРОЦЕСІ НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ОДЕСЬКА ПОЛІТЕХНІКА»

ДОВІДКА

про використання результатів дисертаційної роботи
Курінька Дмитра Дмитровича
«Моделі та методи машинного навчання для виявлення та усунення антипатернів в програмних компонентах»
у навчальному процесі
Національного університету «Одеська політехніка»

Довідка видана стосовно того, що в програмах, навчально-методичних матеріалах та курсах лекцій з дисциплін, що вивчаються на кафедрі Штучного інтелекту та аналізу даних Інституту штучного інтелекту та робототехніки бакалаврами спеціальності 122 «Комп'ютерні науки» – «Інтелектуальний аналіз даних», використовуються наукові результати, отримані в дисертації Курінька Дмитра Дмитровича:

– дисципліна «Об'єктно-орієнтоване програмування» – у межах тем, присвячених структурі та якості програмного коду, студенти знайомляться з концепціями антипатернів, технічного боргу та рефакторингу, розглядають реальні приклади порушення принципів ООП, а також вивчають сучасні підходи до автоматизованого аналізу коду. Матеріали дисертації застосовано для пояснення ролі графових подань AST/CFG/PDG, принципів виявлення дублікатів та структурних аномалій, а також для демонстрації можливостей машинного навчання у завданнях підтримки якості ПЗ. Використання практичних прикладів із запропонованих моделей покращує розуміння студентами проблематики підтримуваності та еволюції ООП-систем;

– дисципліна «Побудова систем штучного інтелекту» – результати дисертації застосовуються для ознайомлення студентів із задачами машинного навчання у сфері Software Engineering, зокрема у моделюванні програмного коду як графових структур, класифікації антипатернів, формуванні рекомендацій щодо рефакторингу та використанні оцінки невизначеності в інтелектуальних системах.

Перший проректор, проректор з науково-педагогічної та виховної роботи,
д.т.н., професор — Сергій НЕСТЕРЕНКО

Голова методичної ради ІШІР,
к.т.н., доцент — Павло СТУПЕНЬ

ДОДАТОК Г ДОВІДКА ПРО ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ У НАУКОВО-ДОСЛІДНИЦЬКІЙ ДІЯЛЬНОСТІ НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ОДЕСЬКА ПОЛІТЕХНІКА»

Україна
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

КОД ЄДРПОУ 43861328

пр.Шевченка, 1, м.Одеса, 65044
Україна
тел. +38 048 7223474
факс. +38 048 7221992
E-mail: mail@op.edu.ua

19.12.2025 № 2059/1991-07
на №

ДОВІДКА

про використання результатів дисертаційної роботи
Курінька Дмитра Дмитровича
«Моделі та методи машинного навчання для виявлення та усунення антипатернів в програмних компонентах»
у науково-дослідницькій діяльності
Національного університету «Одеська політехніка»

Довідка видана в тому, що у науково-дослідницькій діяльності Національного університету «Одеська Політехніка» використані наступні наукові результати, отримані у дисертаційній роботі Курінька Дмитра Дмитровича:

– модель виявлення антипатернів у програмних компонентах на основі гібридного графового подання коду;

– метод рекомендації рефакторингів через багатоцільову оптимізацію з оцінкою невизначеності, спеціалізований на усуненні дублікатів коду;

– метод композиції «мінімально інвазивних» послідовностей рефакторингів на основі причинно-наслідкового виведення з історії репозиторію;

– модель оцінювання ефективності рефакторингів на основі статистичного контролю процесів із атрибуцією впливу.

Дисертація є складовою частиною НДР № 237-177 «Програмні системи машинного навчання та їх застосування в галузі інтелектуального аналізу даних» (№ держреєстрації 0123U102594).

Проректор Дмитро ДМИТРИШИН

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ · НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА» · 43861328

ДОДАТОК Д ВИХІДНИЙ КОД ПРОГРАМНОГО ІНСТРУМЕНТАЛЬНОГО ЗАСОБУ

```
"""
Core domain entities for the refactoring advisory system.

This module defines the fundamental data structures used across the entire
instrumental tool, including code entities, refactoring candidates, feature
representations, uncertainty estimates, and recommendation objects.

The entities defined here are intentionally framework-agnostic and are
designed
to support experimental reproducibility, explainability, and extensibility.
"""

from __future__ import annotations

from dataclasses import dataclass, field
from enum import Enum
from typing import Any, Dict, List, Optional


class CodeEntityType(Enum):
    """
    Enumeration of supported code entity types.
    """
    MODULE = "module"
    CLASS = "class"
    METHOD = "method"
    FUNCTION = "function"
    FILE = "file"


@dataclass(frozen=True)
class CodeLocation:
    """
    Represents a location of a code entity in the source code.
    """
    file_path: str
    start_line: Optional[int] = None
    end_line: Optional[int] = None


@dataclass
class CodeEntity:
    """
    Represents a structural element of the source code
    (e.g., class, method, or module).

    This entity serves as a unifying abstraction for feature extraction,
    modeling, and refactoring analysis.
    """
    entity_id: str
    name: str
    entity_type: CodeEntityType
```

```
    location: CodeLocation

    language: str
    parent_id: Optional[str] = None

    metadata: Dict[str, Any] = field(default_factory=dict)

    def is_local(self) -> bool:
        """
        Returns True if the entity is a local-level construct
        (method or function).
        """
        return self.entity_type in {
            CodeEntityType.METHOD,
            CodeEntityType.FUNCTION,
        }

    def is_global(self) -> bool:
        """
        Returns True if the entity represents a higher-level
        architectural element.
        """
        return self.entity_type in {
            CodeEntityType.CLASS,
            CodeEntityType.MODULE,
            CodeEntityType.FILE,
        }


@dataclass
class FeatureVector:
    """
    Container for heterogeneous features describing a code entity.

    Features may include structural metrics, smell indicators,
    historical signals, and contextual attributes.
    """
    entity_id: str
    features: Dict[str, float]

    def get(self, name: str, default: float = 0.0) -> float:
        """
        Safely retrieve a feature value by name.
        """
        return self.features.get(name, default)

    def as_dict(self) -> Dict[str, float]:
        """
        Returns the raw feature dictionary.
        """
        return dict(self.features)


@dataclass
class UncertaintyEstimate:
    """
    Represents uncertainty information associated with a model prediction.
```

```
    This abstraction allows the system to reason about confidence,
    abstention, and risk-aware decision-making.
    """
    confidence: float
    variance: Optional[float] = None
    entropy: Optional[float] = None
    method: Optional[str] = None

    def is_high_uncertainty(self, threshold: float) -> bool:
        """
        Returns True if the uncertainty exceeds a given threshold.
        """
        return self.confidence < threshold


class RefactoringType(Enum):
    """
    Enumeration of supported refactoring operation types.
    """
    EXTRACT_METHOD = "extract_method"
    MOVE_METHOD = "move_method"
    EXTRACT_CLASS = "extract_class"
    RENAME = "rename"
    INLINE = "inline"
    NONE = "none"


@dataclass
class RefactoringCandidate:
    """
    Represents a potential refactoring opportunity detected by the system.

    This object aggregates structural evidence, model predictions,
    and uncertainty estimates for a single refactoring action.
    """
    candidate_id: str
    entity: CodeEntity
    refactoring_type: RefactoringType

    feature_vector: FeatureVector
    predicted_benefit: Optional[float] = None
    estimated_effort: Optional[float] = None
    estimated_risk: Optional[float] = None

    uncertainty: Optional[UncertaintyEstimate] = None
    rationale: Optional[str] = None

    def is_actionable(self, risk_threshold: float) -> bool:
        """
        Determines whether the refactoring candidate is considered
        actionable under a given risk threshold.
        """
        if self.estimated_risk is None:
            return False
        return self.estimated_risk <= risk_threshold
```

```
@dataclass
class Recommendation:
    """
    Represents a final refactoring recommendation produced by the system.

    Recommendations are intended to be explainable, risk-aware,
    and optionally subject to developer feedback.
    """
    recommendation_id: str
    candidates: List[RefactoringCandidate]

    accepted: Optional[bool] = None
    developer_feedback: Optional[str] = None

    explanation: Optional[str] = None
    created_at: Optional[str] = None

    def has_high_uncertainty(self, threshold: float) -> bool:
        """
        Returns True if any candidate in the recommendation
        exhibits high uncertainty.
        """
        for candidate in self.candidates:
            if candidate.uncertainty is not None:
                if candidate.uncertainty.is_high_uncertainty(threshold):
                    return True
        return False

"""
Feature vector abstraction for refactoring analysis.

This module defines a structured and extensible representation of
heterogeneous features extracted from source code entities.

Feature vectors are designed to support:
- noisy and incomplete data,
- heterogeneous feature sources,
- normalization and filtering,
- explainable downstream decision-making.
"""

from __future__ import annotations

from dataclasses import dataclass, field
from typing import Dict, Iterable, Optional


@dataclass
class FeatureVector:
    """
    Represents a set of numerical features associated with a code entity.

    Features may originate from different sources such as:
    - structural metrics,
    - code smell detectors,
    - historical analysis,
```

```
    - contextual project metadata.
    """
    entity_id: str
    values: Dict[str, float] = field(default_factory=dict)
    source: Optional[str] = None

    def get(self, name: str, default: float = 0.0) -> float:
        """
        Retrieve a feature value by name with a safe default.
        """
        return self.values.get(name, default)

    def set(self, name: str, value: float) -> None:
        """
        Set or update a feature value.
        """
        self.values[name] = float(value)

    def remove(self, name: str) -> None:
        """
        Remove a feature from the vector if it exists.
        """
        self.values.pop(name, None)

    def keys(self) -> Iterable[str]:
        """
        Return an iterable over feature names.
        """
        return self.values.keys()

    def as_dict(self) -> Dict[str, float]:
        """
        Return a shallow copy of the feature dictionary.
        """
        return dict(self.values)

    def merge(self, other: FeatureVector, overwrite: bool = True) ->
FeatureVector:
        """
        Merge another feature vector into this one.

        Parameters
        ----------
        other : FeatureVector
            Feature vector to be merged.
        overwrite : bool
            If True, existing values will be overwritten.

        Returns
        -------
        FeatureVector
            New merged feature vector.
        """
        merged = FeatureVector(entity_id=self.entity_id)
        merged.values.update(self.values)

        for key, value in other.values.items():
```

```
            if overwrite or key not in merged.values:
                merged.values[key] = value

        return merged

    def normalize_min_max(
        self,
        min_values: Dict[str, float],
        max_values: Dict[str, float],
        epsilon: float = 1e-8,
    ) -> FeatureVector:
        """
        Apply min-max normalization to feature values.

        Normalization is performed per feature using provided
        global or project-level statistics.

        Features without corresponding statistics are left unchanged.
        """
        normalized = FeatureVector(entity_id=self.entity_id)

        for name, value in self.values.items():
            if name in min_values and name in max_values:
                min_v = min_values[name]
                max_v = max_values[name]
                if abs(max_v - min_v) > epsilon:
                    normalized.values[name] = (value - min_v) / (max_v -
min_v)
                else:
                    normalized.values[name] = 0.0
            else:
                normalized.values[name] = value

        return normalized

    def filter_by_prefix(self, prefix: str) -> FeatureVector:
        """
        Return a new feature vector containing only features
        whose names start with the given prefix.
        """
        filtered = FeatureVector(entity_id=self.entity_id)
        for name, value in self.values.items():
            if name.startswith(prefix):
                filtered.values[name] = value
        return filtered

    def filter_by_whitelist(self, allowed: Iterable[str]) -> FeatureVector:
        """
        Return a new feature vector containing only whitelisted features.
        """
        allowed_set = set(allowed)
        filtered = FeatureVector(entity_id=self.entity_id)

        for name, value in self.values.items():
            if name in allowed_set:
                filtered.values[name] = value
```

```
        return filtered

    def sparsity(self) -> float:
        """
        Compute sparsity of the feature vector.

        Sparsity is defined as the ratio of zero-valued features
        to the total number of features.
        """
        if not self.values:
            return 1.0

        zero_count = sum(1 for v in self.values.values() if v == 0.0)
        return zero_count / len(self.values)

    def is_empty(self) -> bool:
        """
        Return True if the feature vector contains no features.
        """
        return not bool(self.values)

    def summary(self) -> str:
        """
        Return a concise textual summary of the feature vector.

        Intended for logging, debugging, or explainability.
        """
        return (
            f"FeatureVector(entity_id={self.entity_id}, "
            f"features={len(self.values)}, "
            f"sparsity={self.sparsity():.2f})"
        )

"""
Main analysis pipeline for the refactoring advisory system.

This module defines a high-level orchestration logic that connects
code analysis, feature extraction, model inference, decision-making,
and recommendation generation into a single, reproducible workflow.

The pipeline is intentionally designed to be modular, incremental,
and suitable for experimental evaluation.
"""

from __future__ import annotations

from typing import Iterable, List, Optional

from refactoring_advisor.core.entities import (
    CodeEntity,
    FeatureVector,
    RefactoringCandidate,
    Recommendation,
    RefactoringType,
    UncertaintyEstimate,
)
```

```
class AnalysisPipeline:
    """
    High-level orchestration of the refactoring analysis workflow.

    Each pipeline step is implemented as a replaceable component,
    allowing alternative models, feature sets, and policies to be
    evaluated in isolation.
    """

    def __init__(
        self,
        feature_extractor,
        prediction_model,
        scoring_model,
        risk_policy,
        uncertainty_threshold: float = 0.6,
    ) -> None:
        """
        Initialize the analysis pipeline.

        Parameters
        ----------
        feature_extractor
            Component responsible for extracting FeatureVector objects.
        prediction_model
            Model that predicts refactoring benefits or probabilities.
        scoring_model
            Component estimating benefit, effort, and risk.
        risk_policy
            Policy object defining acceptance and abstention logic.
        uncertainty_threshold : float
            Threshold below which predictions are considered uncertain.
        """
        self.feature_extractor = feature_extractor
        self.prediction_model = prediction_model
        self.scoring_model = scoring_model
        self.risk_policy = risk_policy
        self.uncertainty_threshold = uncertainty_threshold

    def analyze_entities(
        self, entities: Iterable[CodeEntity]
    ) -> List[Recommendation]:
        """
        Run the full refactoring analysis pipeline on a collection of
        code entities.

        Parameters
        ----------
        entities : Iterable[CodeEntity]
            Code entities to be analyzed.

        Returns
        -------
        List[Recommendation]
            Final recommendations produced by the pipeline.
        """
```

```
    recommendations: List[Recommendation] = []

    for entity in entities:
        feature_vector = self._extract_features(entity)
        candidate = self._build_candidate(entity, feature_vector)

        if candidate is None:
            continue

        recommendation = self._build_recommendation(candidate)
        if recommendation is not None:
            recommendations.append(recommendation)

    return recommendations

def _extract_features(self, entity: CodeEntity) -> FeatureVector:
    """
    Extract a feature vector for a given code entity.
    """
    return self.feature_extractor.extract(entity)

def _build_candidate(
    self,
    entity: CodeEntity,
    feature_vector: FeatureVector,
) -> Optional[RefactoringCandidate]:
    """
    Construct a refactoring candidate using model predictions
    and uncertainty estimation.
    """
    prediction = self.prediction_model.predict(feature_vector)

    if prediction is None:
        return None

    uncertainty = self._estimate_uncertainty(prediction)

    candidate = RefactoringCandidate(
        candidate_id=f"cand:{entity.entity_id}",
        entity=entity,
        refactoring_type=prediction.get(
            "refactoring_type", RefactoringType.NONE
        ),
        feature_vector=feature_vector,
        predicted_benefit=prediction.get("benefit"),
        uncertainty=uncertainty,
    )

    benefit, effort, risk = self.scoring_model.score(candidate)
    candidate.predicted_benefit = benefit
    candidate.estimated_effort = effort
    candidate.estimated_risk = risk

    return candidate

def _estimate_uncertainty(
    self, prediction: dict
```

```
    ) -> Optional[UncertaintyEstimate]:
        """
        Estimate uncertainty associated with a prediction.
        """
        confidence = prediction.get("confidence")

        if confidence is None:
            return None

        return UncertaintyEstimate(
            confidence=confidence,
            variance=prediction.get("variance"),
            entropy=prediction.get("entropy"),
            method=prediction.get("uncertainty_method"),
        )

    def _build_recommendation(
        self, candidate: RefactoringCandidate
    ) -> Optional[Recommendation]:
        """
        Decide whether to emit a recommendation or abstain
        based on uncertainty and risk policies.
        """
        # Abstain if uncertainty is too high
        if (
            candidate.uncertainty is not None
            and             candidate.uncertainty.confidence             <
self.uncertainty_threshold
        ):
            return None

        # Apply risk policy
        if not self.risk_policy.accept(candidate):
            return None

        explanation = self._generate_explanation(candidate)

        return Recommendation(
            recommendation_id=f"rec:{candidate.candidate_id}",
            candidates=[candidate],
            explanation=explanation,
        )

    def _generate_explanation(
        self, candidate: RefactoringCandidate
    ) -> str:
        """
        Generate a minimal human-readable explanation for a recommendation.
        """
        parts = [
            f"Entity '{candidate.entity.name}'",
            f"refactoring='{candidate.refactoring_type.value}'",
        ]

        if candidate.predicted_benefit is not None:
            parts.append(f"benefit={candidate.predicted_benefit:.2f}")
```

```
        if candidate.estimated_risk is not None:
            parts.append(f"risk={candidate.estimated_risk:.2f}")

        return ", ".join(parts)

"""
Uncertainty modeling utilities for refactoring analysis.

This module provides abstractions and helper methods for representing,
estimating, and reasoning about uncertainty associated with model
predictions in the refactoring advisory system.

The goal is not to enforce a specific probabilistic framework, but to
support multiple uncertainty estimation strategies in a unified manner.
"""

from __future__ import annotations

from dataclasses import dataclass
from enum import Enum
from typing import Optional


class UncertaintyMethod(Enum):
    """
    Enumeration of supported uncertainty estimation methods.
    """
    NONE = "none"
    ENSEMBLE_VARIANCE = "ensemble_variance"
    BAYESIAN_POSTERIOR = "bayesian_posterior"
    MC_DROPOUT = "mc_dropout"
    ENTROPY = "entropy"
    HEURISTIC = "heuristic"


@dataclass
class UncertaintyEstimate:
    """
    Represents an uncertainty estimate associated with a model prediction.

    This object is intentionally lightweight and model-agnostic.
    """
    confidence: float
    method: UncertaintyMethod = UncertaintyMethod.NONE

    variance: Optional[float] = None
    entropy: Optional[float] = None
    sample_count: Optional[int] = None

    def is_uncertain(self, threshold: float) -> bool:
        """
        Return True if the confidence is below the given threshold.
        """
        return self.confidence < threshold

    def is_reliable(self, threshold: float) -> bool:
        """
```

```
        Return True if the confidence is above or equal to the threshold.
        """
        return self.confidence >= threshold

    def summary(self) -> str:
        """
        Return a short textual summary of the uncertainty estimate.
        """
        return (
            f"Uncertainty(confidence={self.confidence:.2f}, "
            f"method={self.method.value})"
        )


class UncertaintyAggregator:
    """
    Utility class for aggregating uncertainty estimates obtained
    from multiple sources or models.
    """

    @staticmethod
    def aggregate_confidence(
        *estimates: UncertaintyEstimate,
        strategy: str = "min",
    ) -> Optional[float]:
        """
        Aggregate confidence values using a specified strategy.

        Supported strategies:
        - 'min' : pessimistic aggregation
        - 'max' : optimistic aggregation
        - 'mean': average confidence
        """
        if not estimates:
            return None

        confidences = [e.confidence for e in estimates if e is not None]
        if not confidences:
            return None

        if strategy == "min":
            return min(confidences)
        if strategy == "max":
            return max(confidences)
        if strategy == "mean":
            return sum(confidences) / len(confidences)

        raise ValueError(f"Unknown aggregation strategy: {strategy}")

    @staticmethod
    def aggregate_entropy(
        *estimates: UncertaintyEstimate,
    ) -> Optional[float]:
        """
        Aggregate entropy values by computing their mean.

        Estimates without entropy values are ignored.
```

```
        """
        entropies = [
            e.entropy for e in estimates
            if e is not None and e.entropy is not None
        ]

        if not entropies:
            return None

        return sum(entropies) / len(entropies)


class AbstentionPolicy:
    """
    Implements abstention logic based on uncertainty estimates.

    This policy determines whether the system should refrain from
    producing a refactoring recommendation.
    """

    def __init__(self, confidence_threshold: float) -> None:
        """
        Initialize the abstention policy.

        Parameters
        ----------
        confidence_threshold : float
            Minimum acceptable confidence level.
        """
        self.confidence_threshold = confidence_threshold

    def  should_abstain(self,  estimate:  Optional[UncertaintyEstimate])  ->
bool:
        """
        Return True if the system should abstain from making a decision.
        """
        if estimate is None:
            return True

        return estimate.is_uncertain(self.confidence_threshold)
```