



UDC 004. [14.23+15.53]

В. В. Кухар¹, Ю. В. Шабатура^{1,2}

¹ Національний лісотехнічний університет України, м. Львів, Україна

² Національна академія сухопутних військ ім. гетьмана Петра Сагайдачного, м. Львів, Україна

МЕТОД ГРАФ-ОРІЄНТОВАНОЇ ДЕКОМПОЗИЦІЇ МОНОЛІТНИХ ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ ІЗОЛЯЦІЇ РЕЛІЗІВ І ЛОКАЛІЗАЦІЇ РЕГРЕСІЙНОГО ТЕСТУВАННЯ

Встановлено, що однією з ключових проблем супроводу великих інформаційних систем з монолітною архітектурою є необхідність виконання повного її регресійного тестування системи за будь-яких локальних змін коду, що істотно збільшує тривалість релізного циклу та навантаження на CI/CD-інфраструктуру. Наявні підходи до структурування монолітних систем переважно орієнтовані на повну міграцію до мікросервісної архітектури і не вирішують завдання ізоляції релізів та локалізації автотестів всередині єдиної кодової бази. Отже, є очевидною невирішена методологічна проблема, для вирішення якої розроблено новий метод декомпозиції монолітних систем на основі зваженого графа функціональної відповідальності. Для кожного ребра графа характерні три компоненти: структурна вагомість, яка визначається на основі статичного аналізу коду, доменна вагомість, що відображає спільне використання бізнес-сутностей, та темпоральна вагомість, що обчислюється з git-історії за модифікованим індексом Жаккара для множин комітів. Виявлено, що темпоральна компонента фіксує приховані залежності, які невидимі за статичного аналізу коду. Зокрема, під час аналізу модельної системи із 312 ребрами графа, 41 зв'язок мав ненульову темпоральну вагу за нульових значень структурної та доменної вагомостей. Це свідчить про існування неформальних архітектурних зв'язків, які сформувались упродовж релізної історії системи. У роботі розроблено алгоритм *affecteded(v,d)*, що реалізує обхід в ширину по графу з регульованою глибиною *d* і забезпечує формально верифіковану повноту відбору тестів для довільної зміни. Оцінено вплив параметрів порогу декомпозиції θ та глибини обходу *d* на ефективність роботи запропонованого методу граф-орієнтованої декомпозиції. Зокрема, виявлено стійкий діапазон $\theta \in [0,25, 0,35]$, у якому алгоритм стабільно виділяє 9-11 компонентів незалежно від точного значення порогу. Охарактеризовано закономірності зміни обсягу вибіркового тестування системи залежно від характеру змін. Сукупно це дало змогу скоротити тривалість регресійного тестування системи від 59,7 до 90,7 % за значення коефіцієнта виявлення дефектів 1,0 в усіх розглянутих сценаріях. З'ясовано, що запропонований метод є незалежним від технологічного стеку та придатним для веб-орієнтованих систем, реалізованих на основі будь-якої платформи.

Ключові слова: аналіз впливу змін; темпоральні залежності; функціональне розмежування; CI/CD-оптимізація; архітектурна модульність.

Вступ / Introduction

Історично розроблення і розвиток більшості інформаційних систем відбувалися на основі монолітної архітектури. На ранніх етапах розроблення такий підхід базувався на єдиній кодовій базі, що забезпечувало швидке нарощування функціоналу і спрощувало налагодження програмного коду. Однак із зростанням складності інформаційних систем ця перевага перетворюється на структурне обмеження: компоненти набувають прихованих залежностей, функціональні межі розмиваються, а будь-яке локальне втручання в код починає створювати непередбачувані відгуки по всій системі.

Проблему ускладнення монолітних систем широко досліджено у світовій науковій літературі. Значна частина робіт зосереджена на стратегіях міграції до мікросервісної архітектури, тоді як паралельно розвивається напрям вибіркового регресійного тестування системи

(англ. *Test Impact Analysis*), орієнтований на відображення змін коду на множині тестів без урахування архітектурних меж системи. У систематичному аналізі [1] зафіксовано понад 40 методів декомпозиції монолітних систем, переважна більшість яких спрямована саме на міграцію.

Попри широке охоплення суміжних задач актуальним залишається завдання розроблення формалізованого методу ізоляції релізів та локалізації регресійного тестування системи всередині монолітної системи без її повного переписування. Проведений аналіз доступних інформаційних джерел показав, що таке завдання не має належного методологічного забезпечення. Зазначене свідчить про актуальність цієї роботи, оскільки більшість практичних рішень у цьому напрямі мають описовий характер і спираються на специфіку конкретного фреймворку, що унеможливорює їх узагальнення і надійне відтворення.

© Copyright 2026 by the author(s)

Кухар Віктор Володимирович, магістр, аспірант, кафедра комп'ютерних наук.

Email: viktorkukharr@gmail.com; <https://orcid.org/0009-0009-4084-0710>

Шабатура Юрій Васильович, д-р техн. наук, професор, кафедра фундаментальних наук.

Email: shabaturayuriy@gmail.com; <https://orcid.org/0000-0002-9961-1244>

Цитування за ДСТУ: Кухар В. В., Шабатура Ю. В. Метод граф-орієнтованої декомпозиції монолітних інформаційних систем для ізоляції релізів і локалізації регресійного тестування. Науковий вісник НЛТУ України. 2026, т. 36, № 4. С. 79–87.

Citation APA: Kukhar, V. V., & Shabatura, Yu. V. Graph-based decomposition of monolithic information systems for release isolation and selective regression testing. *Scientific Bulletin of UNFU*, 36(4), 79–87. <https://doi.org/10.36930/40360409>

Постановка завдання. Необхідно розробити формалізований метод, який дає змогу виявляти природні межі між функціональними компонентами монолітної системи, формувати ці компоненти як релізо-ізольовані одиниці та визначати мінімально достатню множину автотестів для перевірки кожної локальної зміни. Метод має бути незалежним від технологічного стеку і придатним для застосування в інформаційних системах, що активно розвиваються.

Об'єкт дослідження – декомпозиція монолітних інформаційних систем за умови активного релізного циклу.

Предмет дослідження – методи виявлення функціональних залежностей між підсистемами моноліту та їх використання для ізоляції релізів і локалізації регресійного тестування інформаційної системи, що дасть змогу скоротити його обсяг без зниження повноти виявлення дефектів.

Мета роботи – розробити та верифікувати метод граф-орієнтованої декомпозиції монолітних інформаційних систем, який забезпечуватиме виявлення та формування релізо-ізольованих компонентів і вибіркове виконання регресійних тестів без зниження якості остаточного програмного продукту.

Для досягнення зазначеної мети визначено такі основні завдання дослідження:

- з'ясувати природу залежностей між підсистемами моноліту та визначити причини, які унеможливають ізоляцію релізів без попередньої структурної підготовки системи, що дасть змогу обґрунтувати потребу у формальному описі залежностей, який враховує не тільки прямі зв'язки коду, а й приховані;
- побудувати формальну модель, яка даватиме змогу вимірювати і порівнювати виявлені залежності, що забезпечить єдину кількісну основу для встановлення меж між компонентами і відрізнятиме сильнозв'язані пари вузлів від слабкозв'язаних;
- розробити на основі отриманої моделі алгоритм, який визначатиме мінімально необхідну множину тестів для перевірки конкретної зміни, що дасть змогу замінити повний регресійний тест-ран вибірквим без зниження повноти виявлення дефектів;
- верифікувати розроблений алгоритм за умов, максимально наближених до реальних, що дасть змогу кількісно оцінити досяжне скорочення обсягу тестування системи і підтвердити збереження повноти виявлення дефектів.

Аналіз останніх досліджень та публікацій. Завдання структурування монолітних систем та оптимізації їх тестування перебуває на перетині двох дослідницьких напрямів, які розвиваються переважно ізольовано. Перший – декомпозиція монолітних додатків, другий – вибіркве регресійне тестування програмних систем.

У роботі [1] систематично проаналізовано понад 40 методів декомпозиції монолітних систем на основі 35 первинних публікацій. Автори виокремлюють чотири фази цього процесу та констатують принципову прогалину: жоден із розглянутих у цих роботах методів не поєднує одночасно статичні, динамічні та еволюційні характеристики коду під час визначення меж компонентів. Саме це спостереження стало відправною точкою для даної роботи.

У роботі [9] подано розширений систематичний огляд підходів до ідентифікації мікросервісів, що охоплює публікації до 2024 року. Автори підтверджують домінування міграційно-орієнтованих методів і окремо вказують на нестачу підходів, які враховують еволю-

ційну динаміку системи – тобто те, як залежності між компонентами змінюються з часом у процесі релізів.

У роботі [4] запропоновано автоматизований метод декомпозиції монолітних систем на основі графової кластеризації і комбінаторної оптимізації. Підхід технічно близький до розробленого методу граф-орієнтованої декомпозиції, однак принципово відрізняється за метою: автори вирішують завдання повної міграції до мікросервісної архітектури, тоді як збереження монолітної кодової бази залишається поза увагою.

У роботі [8] розроблено систему CARGO для міграції монолітних застосунків із застосуванням потоковочутливого графа залежностей, що враховує спільний доступ до таблиць бази даних. Автори фіксують, що статичний аналіз імпортів систематично пропускає залежності через спільні зовнішні ресурси – спостереження, суміжне з обґрунтуванням доменної вагомості w_d у запропонованому методі.

У роботі [7] запропоновано виявляти кандидатів у сервіси через аналіз трейсів виконання. Підхід забезпечує динамічну картину залежностей, однак вимагає повного покриття системою функціональними тестами – умова, яка для більшості реальних монолітів є недосяжною. Це обмеження є одним із практичних аргументів на користь використання git-історії як альтернативного джерела поведінкової інформації.

У роботі [13] вперше систематично досліджено концепцію модульного моноліту як повноцінної архітектурної стратегії. Автори фіксують зростання інтересу до цього підходу з боку промислових компаній і водночас відзначають відсутність формалізованих методів підтримки такої архітектури, зокрема, для визначення меж модулів і управління тестуванням системи програмних модулів.

У роботі [10] емпірично встановлено, що навіть за незначної кількості змінених файлів понад половина тестів потрапляє до зони впливу через щільність залежностей на рівні коду. Автори роблять висновок, що аналіз впливу змін без попередньої структурної підготовки системи дає обмежений практичний ефект.

У роботі [11] показано, що аналіз впливу змін на рівні інтеграційних тестів Java-програм дає можливість скоротити обсяг тестування системи до 80 % зі збереженням повноти виявлення дефектів. Автори при цьому спираються на динамічний аналіз в разі кожної зміни, що переносить основне обчислювальне навантаження на сам релізний процес, а не на попередню структурну підготовку.

Отже, аналіз останніх досліджень та публікацій засвідчує, що наявні методи декомпозиції монолітних систем вирішують завдання міграції, а не ізоляції релізів усередині монолітної системи, тоді як методи вибіркового тестування програмного забезпечення функціонують на рівні файлів і класів без урахування архітектурних меж між компонентами. Поєднання граф-орієнтованої декомпозиції монолітних систем з багатопаровою характеристикою залежностей – зокрема, з урахуванням еволюційної git-історії – у єдиному методі, орієнтованому на збереження монолітної кодової бази, у розглянутій літературі не наведено.

Матеріали та методи дослідження. Матеріалами дослідження є типові веборієнтовані інформаційні системи з монолітною архітектурою, для яких характерні активний релізний цикл та розгалужена структура фун-

кціональних залежностей між підсистемами. Методологічну основу роботи становлять методи статичного аналізу коду, теорія графів і кластеризації, аналіз історії системи контролю версій, а також методи вимірювання покриття автоматизованими тестами. Верифікацію розробленого методу граф-орієнтованої декомпозиції виконано на модельній системі, параметри якої відтворюють характеристики реальних промислових застосунків середнього масштабу.

Результати дослідження та їх обговорення / Research results and their discussion

Умови апробації та джерела похибки. Для верифікації методу граф-орієнтованої декомпозиції розроблено модельну веборієнтовану інформаційну систему електронної комерції – платформу для управління інтернет-магазином, що охоплює повний операційний цикл від публікації товарів до формування фінансової звітності. Обрано цей клас систем як представницький: подібні платформи середнього масштабу типово реалізуються на основі монолітної архітектури і характеризуються активним релізним циклом, що зумовлений змінами бізнес-правил, податкового законодавства та вимог до інтеграцій.

Систему реалізовано за архітектурним патерном MVC на платформі Ruby on Rails з механізмом Engine-модулів, що дає можливість природно відображати виділені компоненти у структуру коду, хоча сам метод від цього вибору не залежить. Система охоплює десять функціональних підсистем: управління каталогом товарів, оброблення замовлень, платіжні операції, управління користувачами, складський облік, логістику і доставку, маркетинг і промоції, систему сповіщень, аналітику і звітність, а також модуль відгуків. Сукупний набір автоматизованих тестів становить 1200 одиниць – одиничних та інтеграційних. Базовий час повного регресійного тест-рану до декомпозиції системи становить 47 хвилин. Структурну схему функціональних підсистем модельної системи наведено на рис. 3.

Підсистема платіжних операцій містить, зокрема, модулі InvoiceCalculator і TaxRuleEngine. Перший відповідає за формування рахунків і розрахунків підсумкових сум замовлень. Також цей модуль відповідає за логіку підрахунку фінансових показників у рахунку-фактурі (інвойсі). Його головне завдання – зібрати всі дані про товари або послуги та вивести фінальні суми. Водночас, модуль TaxRuleEngine відповідає за застосування податкових правил залежно від типу товару і географії покупця. Між ними відсутні прямі структурні виклики, однак обидва модулі регулярно змінювались спільно в разі оновлення податкового законодавства або введення нових категорій оподаткування, що і зафіксувала темпоральна вагомість.

Під час відтворення результатів варто враховувати три потенційних джерела похибки: варіацію часу виконання тестів залежно від апаратного середовища, суб'єктивний елемент – за визначення доменної вагомості w_d , а також неповноту статичного аналізу – за наявності динамічних викликів або метапрограмування. Кожне з цих обмежень детальніше розглянуто у розділі обговорення результатів дослідження.

Зважений граф функціональної відповідальності. В основу методу граф-орієнтованої декомпозиції покладено використання моделі зваженого орієнтованого гра-

фа функціональної відповідальності (ЗГФВ), що формально визначається виразом:

$$G = (V, E, W), \quad (1)$$

де: V – множина функціональних одиниць системи; E – орієнтовані залежності між ними; $W: E \rightarrow \mathcal{R}$ – функція, що співставляє кожному ребру вектор вагомостей (w_s, w_d, w_i).

У складі вектора вагомостей виділено три компоненти. Структурна вагомість w_s відображає інтенсивність прямих технічних залежностей між вузлами – імпорти, виклики методів, успадкування. Вона обчислюється автоматично статичним аналізом. Доменна вагомість w_d відображає ступінь спільного використання w_i бізнес-сутностей; визначається напівавтоматично через аналіз залежностей на рівні моделей даних. Темпоральна вагомість w_t відображає частоту спільних змін у git-історії та обчислюється згідно з виразом:

$$w_t(v_i, v_j) = \frac{|\text{commits}(v_i) \cap \text{commits}(v_j)|}{|\text{commits}(v_i) \cup \text{commits}(v_j)|}, \forall i, j \in N, \quad (2)$$

де $\text{commits}(v)$ – множина комітів, що торкались вузла v .

Значення цієї вагомості знаходиться у діапазоні $[0, 1]$ і є модифікованим індексом Жаккара для множин комітів.

Запропоновані три шари ознак разом перекривають більшість сценаріїв прихованих залежностей. Структурна вагомість фіксує явні зв'язки коду. Доменна вагомість характеризує залежності через спільні таблиці і події, непомітні в імпортах. Темпоральна вагомість визначає неформальні зв'язки, які склались історично, причому іноді два модулі можуть не мати жодного спільного рядка коду, але стабільно змінюватись разом у кожному релізі. Водночас відзначимо, що будь-який з цих шарів окремо дає неповну картину.

Зведена вагомість ребра визначається як лінійна комбінація відповідно до виразу:

$$W(e_{ij}) = \alpha \cdot w_s + \beta \cdot w_d + \gamma \cdot w_t, \text{ де } \alpha + \beta + \gamma = 1. \quad (3)$$

Коефіцієнти α, β, γ є конфігурованими параметрами методу граф-орієнтованої декомпозиції. За замовчуванням застосовується рівномірний розподіл $\alpha = \beta = \gamma = 1/3$, що відповідає відсутності апіорних пріоритетів між типами залежностей.

Алгоритми декомпозиції монолітних систем та визначення області впливу. Декомпозицію виконано методом жадібної кластеризації за порогом θ . При цьому ребра сортуються за спаданням $W(e)$, а вузли об'єднуються в кластери поки $W(e) \geq \theta$. Ребра з $W(e) < \theta$ визначають межі між компонентами. Результатом є розбиття $C = \{C_1, \dots, C_k\}$, де кожен C_i є релізо-ізолюваним компонентом. У разі виявлення циклічних міжкомпонентних залежностей значення θ локально знижується або відповідні кластери об'єднуються. Рекомендований діапазон $\theta \in [0,25, 0,35]$ обґрунтовано у розділі результатів (рис. 1).

Після декомпозиції системи кожен автотест прив'язується до компонента через coverage mapping – статичний або динамічний аналіз покриття. У разі внесення зміни до вузла v запускається алгоритм $affecteded(v, d)$, що виконує обхід в ширину по ЗГФВ з максимальною глибиною d і повертає об'єднання тестів усіх вузлів у множині $affecteded(v, d) = \{u \in v \mid dist(v, u) \leq d\}$. Параметр d регулює компроміс між повнотою і швидкістю. При $d = 0$ запускаються тільки тести зміненого вузла,

при $d = \infty$ – повний тест-ран. Доведено, що за умови коректної декомпозиції системи і відсутності прихованих залежностей $Testsed(v, d) = 1$) є повною і мінімальною множиною для тестування результатів локальної зміни v .



Рис. 1. Загальна схема методу граф-орієнтованої декомпозиції монолітних систем / General scheme of the graph-based decomposition method: Етап I – одноразова побудова ЗГФВ і виділення компонентів / Stage I – one-time construction of the WFRG and component extraction; Етап II – вибіркове тестування програмного забезпечення в разі кожної зміни коду / Stage II – selective testing at each code change

Метрики оцінювання ефективності методу. Ефективність роботи методу граф-орієнтованої декомпозиції вимірюється трьома показниками. Коефіцієнт скорочення тестування системи (КСТ) – відношення тестів у вибірці $affecteded(v, d)$ до загального числа тестів системи. Чим нижче значення КСТ, тим вибірковішим є тестування системи. Тривалість виконання вибіркового тест-рану T_{sel} порівнюється з базовим, для модельної системи він становив: $T_{full} = 47$ хвилин. Коефіцієнт виявлення дефектів (КВД) є ключовим показником коректності методу граф-орієнтованої декомпозиції. Значення КВД = 1,0 є необхідною умовою, і будь-яке відхилення від нього сигналізує або про неповноту ЗГФВ, або про некоректно обраний параметр d .

Підходи до декомпозиції монолітних систем. Більшість наявних методів декомпозиції монолітних систем орієнтовані на повну міграцію до мікросервісної архітектури. У систематичному аналізі [1] 35 первинних досліджень фіксують чотири фази декомпозиції монолітних систем і водночас визначають недолік, суть якого полягає у тому, що відсутні методи, які одночасно враховують статичні, динамічні та еволюційні характеристики коду. Власне така констатація і стала відправною точкою нашої роботи. У роботі [4] автори пропонують автоматизований підхід на основі графової кластеризації і комбінаторної оптимізації. Він технічно близький до нашого, але принципово відмінний за метою, адже вони вирішують завдання повної міграції, тоді як наш метод зберігає моноліт.

Спільний доступ до зовнішніх ресурсів – баз даних, черг подій, є одним із найпоширеніших джерел прихо-

ваних залежностей, і саме тут статичний аналіз імпор-тів виявляється недостатнім. У роботі [8] систему CARGO розроблено з урахуванням цього фактора через потоково-чутливий граф, що містить спільний доступ до таблиць БД. Ця ідея суміжна з нашою доменною вагою w_d , хоча CARGO зосереджений на уточненні вже наявних розбиттів. У роботі [7] запропоновано виявляти кандидатів у сервіси через аналіз трейсів виконання. Це досить цінний динамічний підхід, але він потребує повного покриття функціональними тестами, що для великих монолітів рідко є досяжним. Запропонований метод уникає цієї залежності, спираючись на git-історію, як альтернативне джерело поведінкової інформації.

Методи вибіркового регресійного тестування програмного забезпечення. Вибіркове виконання тестів є найперспективнішим напрямом оптимізації регресійного тестування програмного забезпечення. До такої думки сходяться дослідники принаймні з часів фундаментального огляду роботи [15]. Розроблений алгоритм $affecteded(v, d)$ належить до цієї категорії, але відрізняється від класичних підходів тим, що спирається на архітектурні межі компонентів, а не на покриття рядків коду.

У систематичному аналізі [5] 187 досліджень зафіксовано істотний розрив: тільки незначна частина академічних методів регресійного тестування програмного забезпечення демонструє готовність до промислового впровадження. Запропонований метод проєктувався саме з урахуванням цього розриву, як новий інструмент для реальних CI/CD-пайплайнів.

У роботі [10] емпірично встановлено, що навіть за малої кількості змінених файлів понад 50 % тестів залишаються в зоні впливу через щільність залежностей на рівні коду. Це пояснює обмежену ефективність аналізу впливу без попередньої декомпозиції системи й обґрунтовує доцільність підходу, який спочатку знижує щільність міжкомпонентних залежностей до 12,2 %, і тільки потім виконує відбір тестів. У роботі [11] підтверджується реалістичність такого підходу. Він показує, що аналіз впливу на рівні інтеграційних тестів Java-програм забезпечує скорочення до 80 %. Цей порядок величин збігається з результатами цієї роботи.

Модульний моноліт як архітектурна стратегія. У першому систематичному дослідженні концепції модульного моноліту у роботі [13] зафіксовано зростання інтересу до цього підходу. Shopify, Appsmith, Gusto і PlayTech обрали його замість мікросервісів як повноцінну архітектурну стратегію. Це підтверджує вихідну тезу роботи про те, що модульний моноліт потребує формальних методів підтримки, а не тільки концептуального визнання. У роботі [9] конкретизовано, де саме ця підтримка відсутня. Серед відкритих проблем галузі явно простежується нестача методів, що враховують еволюційну динаміку системи. Застосування темпоральної вагомості w_t в запропонованому методі є прямою відповіддю саме на цю прогалину.

Позиціонування роботи. Наявні методи декомпозиції монолітних систем [1, 4, 5, 7, 15] вирішують завдання міграції до мікросервісів, а не ізоляції релізів всередині монолітної системи. Методи вибіркового тестування програмного забезпечення [8, 9, 10, 11] функціонують на рівні файлів або класів без урахування архітектурних меж між компонентами. Концепція модульного моноліту у роботі [13] визнає потребу у структуруванні без повного переписування, однак не надає формалізо-

ваного інструменту для визначення меж компонентів. Запропонований метод на основі ЗГФВ заповнює цю нішу, поєднуючи граф-орієнтовану декомпозицію з алгоритмом відбору тестів у єдиний аналітичний конвеєр.

Усі результати отримано на модельній системі, параметри якої описано вище. Далі послідовно розглянуто: характеристики побудованого графа, результати аналізу темпоральних залежностей, структуру декомпозиції системи та показники ефективності вибіркового тестування системи.

Граф і темпоральні залежності. Статичним аналізом виявлено 312 орієнтованих залежностей між 78 функціональними одиницями. За рівномірних коефіцієнтів

скорочення тестування системи (КСТ) $\alpha = \beta = \gamma = 1/3$ відповідно до виразу (3) зведені вагомості ребер розподілились у діапазоні 0,04-0,81 з медіаною 0,29 (рис. 2), розподіл демонструє чітке розшарування між сильно- і слабкозв'язаними парами, що є необхідною передумовою для ефективної кластеризації. Нагадаємо, поняття розподілу зведених вагомостей $W(e)$ ребер ЗГФВ (загального графа фізичних/функціональних взаємозв'язків чи аналогічної структурної моделі) є специфічним терміном, який найчастіше застосовується в теорії складних мереж, системному аналізі та моделюванні структур складних систем (наприклад, у кібернетиці, транспортних або інформаційних мережах).

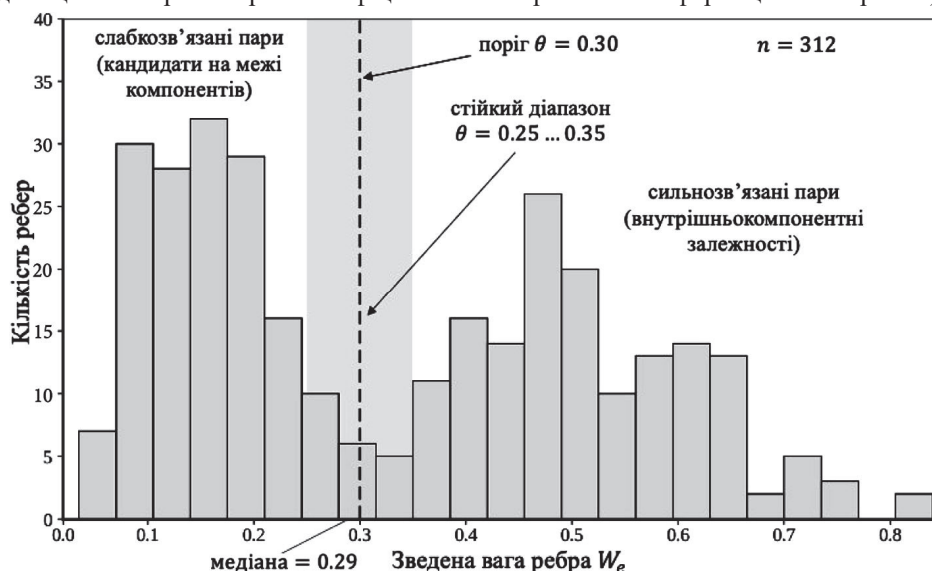


Рис. 2. Розподіл зведених вагомостей $W(e)$ ребер ЗГФВ модельної системи ($n = 312$): бімодальний характер розподілу обґрунтовує вибір порогу $\theta = 0,30$ та пояснює стійкість декомпозиції системи в діапазоні $\theta = 0,25 \dots 0,35$ / Distribution of aggregate edge weights $W(e)$ of the WFRG for the model system ($n = 312$): the bimodal pattern justifies the threshold selection $\theta = 0.30$ and explains decomposition stability in the range $\theta = 0.25 \dots 0.35$

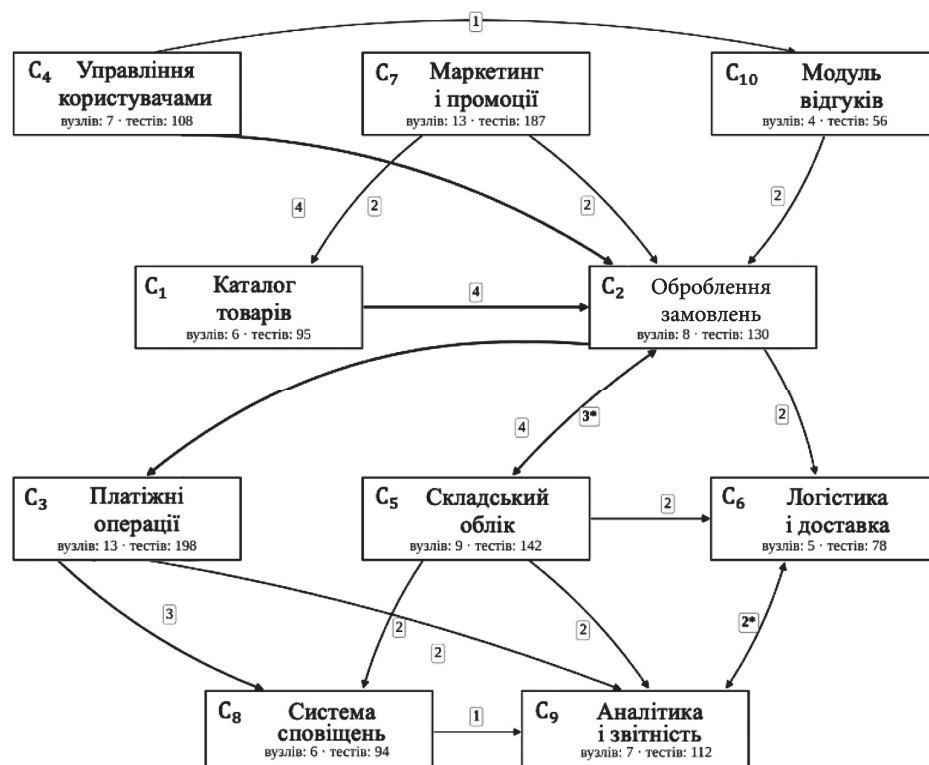


Рис. 3. Структура модельної e-commerce системи після декомпозиції системи при $\theta = 0,30$ / Structure of the model e-commerce system after decomposition at $\theta = 0.30$: компоненти з характеристиками та агреговані міжкомпонентні залежності (числа на ребрах – кількість залежностей між компонентами / components with characteristics and aggregated inter-component dependencies (edge labels indicate dependency counts; * – пари з виявленими циклічними залежностями) / * – pairs with detected cyclic dependencies)

Найінформативнішим виявився результат аналізу темпоральної компоненти, обчисленої згідно з виразом (2). Із 247 комітів за 14 місяців git-аналізом виявлено 41 ребро з ненульовою w_i за нульових w_s і w_d одночасно. Тобто відповідні пари модулів не мають ні спільних імпортів, ні спільних доменних сутностей – однак регулярно змінюються разом. Аналіз показав три типові причини таких залежностей: спільна зовнішня конфігурація, неявне узгодження форматів на межі підсистем, задубована логіка, що еволюціонує синхронно. Жодна з них не відображається у структурі коду – і саме тут статичний аналіз систематично допускає помилки.

Ілюстративним прикладом є пара модулів InvoiceCalculator і TaxRuleEngine підсистеми платіжних операцій (C_3). Між ними відсутній будь-який прямий виклик – статичний аналіз не виявляє залежності. Однак з 19 релізів за досліджуваний період обидва модулі змінювались спільно 13 разів ($w_i = 0,68$), переважно в разі введення змін до правил оподаткування. За відсутності темпорального шару ці модулі опинились би в різних компонентах і тест на їх взаємодію не потрапив би до вибірки affected в разі зміни одного з них.

Результати декомпозиції системи. При $\theta = 0,30$ алгоритмом виділено 10 компонентів (табл. 1). Відповідну структуру системи наведено на рис. 3. На цьому рисунку символом '*' позначено пари з виявленими циклічними залежностями (усунуто локальним q до 0,25), водночас як числа на ребрах – кількість міжкомпонентних залежностей: разом 38 із 312 (12,2 %).

Табл. 1. Характеристики компонентів модельної системи після декомпозиції системи при $\theta = 0,30$ / Characteristics of model system components after decomposition at $\theta = 0,30$

Компонент	Підсистема	Вузлів	Внутр. залежностей	Зовн.	Тестів
C_1	Каталог товарів	6	22	4	95
C_2	Оброблення замовлень	8	29	6	130
C_3	Платіжні операції	13	48	5	198
C_4	Управління користувачами	7	23	3	108
C_5	Складський облік	9	33	5	142
C_6	Логістика і доставка	5	16	4	78
C_7	Маркетинг і промоції	13	45	4	187
C_8	Система сповіщень	6	20	3	94
C_9	Аналітика і звітність	7	25	2	112
C_{10}	Модуль відгуків	4	13	2	56
Разом		78	274	38	1200

Розмір компонентів змінюється від 4 до 13 вузлів, що відповідає нерівномірному розподілу функціональності у вихідній системі. Кількість міжкомпонентних залежностей після декомпозиції системи становить 38 із 312, тобто 12,2 % залежностей розташовані на межах між компонентами, решта 87,8 % є внутрішніми. Під час перевірки функціональності вихідної системи виявлено два цикли: $C_2 \leftrightarrow C_5$ і $C_6 \leftrightarrow C_9$, обидва через ребра з домінантною темпоральною вагою, що підтверджує діагностичну цінність w_i . Цикли усунуто локальним зниженням θ до 0,25 у відповідних зонах графа.

Вибіркове тестування системи і чутливість до параметрів – це взаємопов'язані методології в інженерії ПЗ, системному аналізі та математичному моделюванні, що спрямовані на оптимізацію процесу перевірки складних систем у технічно чи часово обмежених умовах. Оскільки вичерпне тестування всіх можливих станів та комбінацій вводу є математично неможливим, ці підходи допомагають виявити найбільш критичні зони

інформаційної системи. Для оцінювання ефективності методу граф-орієнтованої декомпозиції сформовано чотири сценарії змін, що охоплюють практичний діапазон від мінімальної правки в ізолюваному компоненті до одночасної зміни у двох найзв'язаніших компонентах.

Результати вимірювання тривалості вибіркового тестування та перевірки повноти виявлення дефектів при $d = 1$ наведено у табл. 2. КВД = 1,0 у всіх чотирьох сценаріях підтверджує виконання формальної гарантії повноти. Особливої уваги заслуговує сценарій D , одночасна зміна у двох компонентах з найвищою зовнішньою зв'язністю – випадок, що зазвичай наводиться як контраргумент проти вибіркового тестування системи – забезпечує скорочення тривалості від 47 до 19 хв зі збереженням КВД = 1,0.

Табл. 2. Результати вибіркового тестування системи порівняно з повним тест-раном ($d = 1$, $T_{full} = 47$ хв) / Selective testing results compared to full regression run ($d = 1$, $T_{full} = 47$ min)

Сценарій	Тестів	КСТ	T_{sel} , хв	Скорочення, %
A – мінімальна зміна	112	0,093	4,4	90,7
B – типова зміна	238	0,198	9,3	80,2
C – велике ядро	427	0,356	16,8	64,4
D – два компоненти	484	0,403	19,0	59,7

Примітка: КВД = 1,0.

Аналіз чутливості виявив стійкість декомпозиції системи до варіації θ у діапазоні $[0,25, 0,35]$: кількість компонентів коливається між 9 і 11 незалежно від точного значення порогу, що свідчить про наявність природних архітектурних меж у системі. За межами зазначеного діапазону характер розбиття різко змінюється: при $\theta < 0,20$ система розділяється на 18-22 компоненти, при $\theta > 0,40$ залишається 4-5 укрупнених блоків.

У табл. 3 наведено вплив глибини обходу d на обсяг тестування системи. Залежність КСТ і КВД від глибини d наведено на рис. 4. При $d = 0$ алгоритм не виявляє 9 % дефектів, що виникають на межах із суміжними компонентами; значення $d = 1$ усуває цю прогалину повністю, а подальше збільшення d тільки збільшує тривалість виконання без приросту КВД. Відповідно рекомендовано $d = 1$ як базове значення і $d = 2$ як консервативне за наявності сумнівів у повноті побудованого графа.

Табл. 3. Вплив глибини обходу d на обсяг тестування системи (сценарій B, типова локальна зміна) / Effect of traversal depth d on test volume (scenario B, typical local change)

d	Тестів	КСТ	T_{sel} , хв	КВД
0	108	0,090	4,2	0,91
1	238	0,198	9,3	1,0
2	346	0,288	13,6	1,0
3	512	0,427	20,1	1,0
∞	1200	1,0	47	1,0

Обговорення результатів. Отримані результати підтверджують вихідну гіпотезу, згідно з якою функціональна декомпозиція монолітної системи з чітким розмежуванням зон відповідальності дає можливість кратно скоротити обсяг регресійного тестування системи без зниження повноти виявлення дефектів. Поряд із цим ряд спостережень виявився менш очікуваним і потребує окремого обговорення.

Інтерпретація результатів. Очікуваним є загальний ефект скорочення тестування системи, у цьому полягає базова логіка застосування методу граф-орієнтованої декомпозиції. Менш очікуваним виявилась стійкість декомпозиції системи до варіації θ у діапазоні $[0,25, 0,35]$. Істотна залежність результату від точного

значення порогу зробила б метод ненадійним на практиці, адже кожна нова система вимагала б тривалого калібрування. Зафіксована стійкість свідчить про протилежне, усі природні архітектурні межі в системі існують незалежно від методу граф-орієнтованої декомпозиції, і ЗГФВ їх виявляє, а не конструює штучно.

Виявлення 41 прихованої темпоральної залежності становить найбільш практично значущий результат дослідження. Модулі, що виглядають незалежними з точки зору статичного аналізу, виявляють залежну пове-

дінку за умови реального релізного циклу, саме такі місця є джерелом найнесподіваніших регресій у виробничих системах. Здатність git-історії виявляти подібні зв'язки без динамічного трасування і без ручного аудиту є практично цінним результатом, який складно досягти іншими підходами. Статичний аналіз еволюції коду (репозиторію) дає унікальну можливість виявляти приховані зв'язки, які часто залишаються непомітними для інших інструментів.

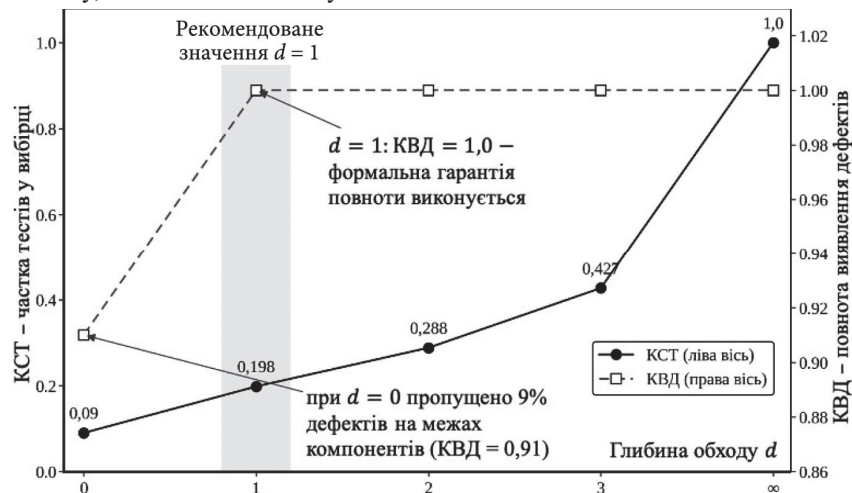


Рис. 4. Залежність коефіцієнтів скорочення тестування системи (КСТ) і виявлення дефектів (КВД) від глибини обходу d для сценарію В: при $d = 1$ досягається повне виявлення дефектів (КВД = 1,0), подальше збільшення d тільки нарощує обсяг тестування системи без приросту якості / Test reduction ratio (TRR) and defect detection rate (DDR) as functions of traversal depth d for scenario В: at $d = 1$ full defect detection is achieved (DDR = 1.0), while further increase of d only expands test volume without quality gain

Обмеження методу граф-орієнтованої декомпозиції та практичні рекомендації. Потрібно зазначити, що застосування розробленого методу граф-орієнтованої декомпозиції вимагає врахування принаймні трьох обмежень. Перше стосується якості git-історії, адже темпоральна вагомість w_i втрачає інформативність, якщо команда практикує укрупнені коміти з непов'язаними змінами. Оцінка дисципліни комітів перед застосуванням методу граф-орієнтованої декомпозиції дає можливість запобігти появі хибних залежностей у графі. Друге обмеження пов'язане з динамічними залежностями через рефлексію або черги подій, які статичний аналіз не виявляє. Для таких систем рекомендується використовувати $d = 2$ замість $d = 1$. Третє – вартість початкової декомпозиції системи. Для великих систем це може вимагати кілька днів роботи, однак точка окупності настає зазвичай після 3-5 релізів із вибірковою тестуванням ПЗ.

З практичної точки зору, рекомендується розпочинати із аналізу темпоральних залежностей, а не зі структурного аналізу, оскільки саме w_i найчастіше виявляє розбіжність між задокументованою і реальною архітектурою системи. Порогове значення θ доцільно узгоджувати з організаційною структурою команди, а параметр d розглядати як інструмент управління повнотою покриття з поступовим зниженням у міру підтвердження коректності побудованого графа.

Обговорення результатів дослідження. У роботі [14] виконано незалежне порівняння восьми інструментів автоматизованої декомпозиції монолітних систем на чотирьох еталонних застосунках. Автори встановили, що жоден із порівнюваних інструментів не забезпечує стабільно кращих результатів в усіх метриках і що відсутність єдиної методологічної бази унеможлиблює ко-

ректне порівняння між підходами. Запропонований метод не претендує на абсолютну перевагу в цьому відношенні, натомість заповнює конкретну нішу – декомпозицію без міграції з метою ізоляції релізів – яку жоден із порівнюваних у роботі [14] інструментів не охоплює.

У роботі [6] розроблено підхід до вибору регресійних тестів з урахуванням ін'єкції залежностей у Java-проектах. Автори показують, що наявні інструменти стають небезпечними за наявності DI-механізмів, оскільки не відстежують залежностей, що виникають через анотації фреймворків. Це обмеження є структурно спорідненим із проблемою динамічних залежностей, описаною у розділі обмежень запропонованого методу граф-орієнтованої декомпозиції: в обох випадках статичний аналіз пропускає залежності, що виникають поза явними імпортами. Відмінність у тому, що запропонований метод компенсує цю прогалину через темпоральний шар, а не через розширення статичного аналізатора.

У роботі [3] запропоновано багатомовний відбір регресійних тестів з урахуванням системи збирання в середовищі СІ. Автори демонструють практичну застосовність вибіркового тестування системи на кодовій базі понад 13 млн рядків коду та фіксують скорочення тривалості тестування системи без втрат у безпечності відбору. Порівняно з цим підходом, запропонований метод функціонує на вищому рівні абстракції – архітектурних компонентів, а не файлів і модулів системи збирання, – що робить його менш залежним від специфіки конкретного технологічного середовища.

У роботі [11] продемонстровано скорочення обсягу тестування системи на 50 % під час застосування аналізу впливу змін на рівні інтеграційних тестів Java-програм. Запропонований метод забезпечує скорочення від 59,7 до 90,7 %, однак пряме порівняння цих показників

некоректне через відмінність у природі даних – реальна система проти модельної. Принципова відмінність полягає в іншому: у підході [11] аналіз впливу виконується динамічно в разі кожної зміни, тоді як у запропонованому методі межі між компонентами встановлюються апіорно, що переносить основні витрати з кожного релізу на одноразову процедуру декомпозиції системи.

У роботі [10] встановлено, що навіть за незначної кількості змінених файлів понад 50 % тестів залишаються в зоні впливу через щільність залежностей на рівні коду. Отриманий у цій роботі показник 12,2 % міжкомпонентних залежностей пояснює принципову різницю результатів: проблема полягає не в алгоритмі відбору тестів, а в тому, що без попередньої структурної підготовки будь-який алгоритм відбору працює з надмірно щільним графом залежностей.

У роботі [12] емпірично досліджено залежності між тестами і модулями коду через аналіз еволюційного зв'язування. Автори показують, що тести та відповідні модулі коду демонструють стійкий патерн спільних змін у git-історії, і цей патерн може бути використаний для локалізації дефектів. Це підтверджує теоретичну основу темпоральної вагомості w_t у запропонованому методі: git-історія несе інформацію про залежності, яка недоступна за статичного аналізу коду.

У роботі [16] запропоновано підхід до вимірювання зв'язності між мікросервісами через аналіз спільного доступу до ресурсів – баз даних, черг подій, API. Автори зазначають, що класичні метрики зв'язності, розраховані на основі викликів між сервісами, систематично недооцінюють реальну щільність залежностей. Цей висновок є незалежним підтвердженням доцільності багатоваріаційної характеристики залежностей у ЗГФВ: структурна вагомість фіксує виклики, тоді як доменна і темпоральна вагомості охоплюють саме ті типи зв'язності, які робота [16] ідентифікує як систематично пропущені.

У роботі [2] зафіксовано принципові труднощі регресійного тестування програмного забезпечення за наскрізного тестування мікросервісних систем: зміна в одному сервісі може спричинити регресію в інших, не пов'язаних з ним прямими викликами. Цей результат є прямим аргументом на користь підходу, за якого межі між компонентами встановлюються апіорно на основі всіх типів залежностей, а не тільки структурних – саме так, як реалізовано в запропонованому методі.

Отже, обговорення результатів дослідження підтверджує доцільність виконаної роботи. Наявні підходи до вибіркового тестування системи або розширюють статичний аналізатор під конкретну мовну платформу [3, 6], або потребують повної мікросервісної архітектури як передумови [2, 16], або функціонують без попередньої структурної підготовки системи і через це стикаються з надмірною щільністю графа залежностей [10, 11]. Запропонований метод відрізняється тим, що компенсує обмеження статичного аналізу через git-історію – джерело, доступне для будь-якої системи незалежно від мови і архітектури – і робить це в рамках незміненої монолітної кодової бази, що у розглянутій літературі аналогів не має.

Отже, внаслідок виконаної роботи можна сформулювати таку наукову новизну та практичну значущість результатів дослідження.

Наукова новизна отриманих результатів дослідження – отримав подальшого розвитку метод граф-орієнто-

ваної декомпозиції монолітних інформаційних систем, який, на відміну від наявних методів, що враховують тільки статичні структурні залежності, ґрунтується на зваженому графі функціональної відповідальності з тришаровою характеристикою залежностей – структурною, доменною та темпоральною вагомостями, обчисленою за модифікованим індексом Жаккара на множинах git-комітів, – що забезпечує виявлення прихованих архітектурних зв'язків, формування релізо-ізованих компонентів і локалізацію регресійного тестування системи всередині єдиної монолітної кодової бази без переходу до мікросервісної архітектури.

Практична значущість результатів дослідження – можна застосувати у процесі супроводу веборієнтованих інформаційних систем з монолітною архітектурою та активним релізним циклом для скорочення обсягу регресійного тестування системи від 59,7 до 90,7 % без зниження повноти виявлення дефектів шляхом впровадження граф-орієнтованої декомпозиції системи на релізо-ізовані компоненти та вибіркового запуску тестів на основі алгоритму *affecteded(v,d)*, до того ж незалежність методу граф-орієнтованої декомпозиції від технологічного стеку уможливорює його використання на системах, реалізованих на будь-якій платформі, без модифікації наявної кодової бази.

Висновки / Conclusions

Розроблено та верифіковано метод граф-орієнтованої декомпозиції монолітних інформаційних систем, який забезпечує виявлення та формування релізо-ізованих компонентів, а також вибіркове виконання регресійних тестів без зниження якості остаточного програмного продукту. За результатами проведеного дослідження можна зробити такі основні висновки.

1. З'ясовано, що основною причиною неможливості ізоляції релізів у монолітних системах є наявність прихованих залежностей між компонентами, які не відображаються у структурі імпорту, але регулярно проявляються у релізному циклі – зокрема, в модельній системі виявлено 41 таку залежність з-поміж 312 ребер графа.
2. Розроблено формальну модель зваженого графа функціональної відповідальності (ЗГФВ), в якій для кожного ребра характерні три компоненти вагомості – структурна w_s , доменна w_d і темпоральна w_t , – що разом забезпечують повне охоплення типів залежностей між функціональними одиницями системи.
3. Встановлено, що темпоральна вагомість, обчислена за модифікованим індексом Жаккара на множинах git-комітів, є найбільш інформативною компонентою методу граф-орієнтованої декомпозиції, оскільки фіксує неформальні архітектурні зв'язки, невидимі за статичного аналізу коду.
4. Розроблено алгоритм *affecteded(v,d)*, що реалізує обхід в ширину по ЗГФВ з регульованою глибиною d і забезпечує формально верифіковану повноту відбору регресійних тестів для довільної зміни; за значення $d = 1$ досягається коефіцієнт виявлення дефектів 1,0 в усіх розглянутих сценаріях змін.
5. Проведено верифікацію методу граф-орієнтованої декомпозиції на модельній системі електронної комерції з 78 функціональними одиницями, 312 залежностями та 1200 автоматизованими тестами, за результатами якої скорочення тривалості регресійного тестування системи становить від 59,7 до 90,7 % залежно від характеру змін зі збереженням повноти виявлення дефектів.
6. Встановлено стійкість декомпозиції системи до варіації порогу θ у діапазоні $[0,25, 0,35]$, що свідчить про наявність природних архітектурних меж у досліджуваній

системі і забезпечує практичну надійність методу граф-орієнтованої декомпозиції без тривалого калібрування.

7. З'ясовано, що метод є незалежним від технологічного стеку і придатним для застосування на веборієнтованих інформаційних системах, реалізованих на будь-якій платформі, без модифікації наявної кодової бази.

References

1. Abgaz, Y., McCarren, A., Elger, P., Solan, D., Lapuz, N., Bivol, M., Jackson, G., Yilmaz, M., Buckley, J., & Clarke, P. (2023). Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. *IEEE Transactions on Software Engineering*, 49(8), 4213–4242. <https://doi.org/10.1109/TSE.2023.3287297>
2. Elsner, D., Bertagnolli, D., Pretschner, A., & Klaus, R. (2022). Challenges in Regression Test Selection for End-to-End Testing of Microservice-Based Software Systems. In *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test (AST 22)* (pp. 93–103). ACM. <https://doi.org/10.1145/3524481.3527217>
3. Elsner, D., Wuersching, R., Schnappinger, M., Pretschner, A., Graber, M., Dammer, R., & Reimer, S. (2022). Build System Aware Multi-language Regression Test Selection in Continuous Integration. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 22)* (pp. 87–96). ACM. <https://doi.org/10.1145/3510457.3513078>
4. Filippone, G., Mehmood, N. Q., Autili, M., Rossi, F., & Tivoli, M. (2023). From monolithic to microservice architecture: An automated approach based on graph clustering and combinatorial optimization. In *Proceedings of the 2023 IEEE 20th International Conference on Software Architecture (ICSA)* (pp. 47–57). IEEE. <https://doi.org/10.1109/ICSA56044.2023.00013>
5. Greca, R., Miranda, B., & Bertolino, A. (2023). State of Practical Applicability of Regression Testing Research: A Live Systematic Literature Review. *ACM Computing Surveys*, 55(13s), article ID 274. <https://doi.org/10.1145/3579851>
6. Hundsdorfer, S., Elsner, D., & Pretschner, A. (2023). DIRTS: Dependency Injection Aware Regression Test Selection. In *Proceedings of the 2023 IEEE 16th International Conference on Software Testing, Verification and Validation (ICST 2023)* (pp. 422–432). IEEE. <https://doi.org/10.1109/ICST57152.2023.00046>
7. Jin, W., Liu, T., Cai, Y., Kazman, R., Mo, R., & Zheng, Q. (2021). Service Candidate Identification from Monolithic Systems Based on Execution Traces. *IEEE Transactions on Software Engineering*, 47(5), 987–1007. <https://doi.org/10.1109/TSE.2019.2910531>
8. Nitin, V., Asthana, S., Ray, B., & Krishna, R. (2022). CARGO: AI-Guided Dependency Analysis for Migrating Monolithic Applications to Microservices Architecture. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE 22)* (pp. 1–12). ACM. <https://doi.org/10.1145/3551349.3556960>
9. Ourmoussa, I., Saidi, R., & Abumandour, W. (2024). Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review. *IEEE Access*, 12, 21871–21902. <https://doi.org/10.1109/ACCESS.2024.3365079>
10. Peng, Z., Chen, T.-H., & Yang, J. (2022). Revisiting Test Impact Analysis in Continuous Testing From the Perspective of Code Dependencies. *IEEE Transactions on Software Engineering*, 48(6), 1979–1993. <https://doi.org/10.1109/TSE.2020.3045914>
11. Shahbaz, M. (2023). An Approach for Test Impact Analysis on the Integration Level in Java Programs. In: Yang, X.-S., Sherratt, R. S., Dey, N., & Joshi, A. (Eds.), *Proceedings of Eighth International Congress on Information and Communication Technology (ICICT 2023)*. Lecture Notes in Networks and Systems, vol. 694 (pp. 171–188). Springer. https://doi.org/10.1007/978-981-99-3091-3_14
12. Sohn, J., & Papadakis, M. (2022). CEMENT: On the Use of Evolutionary Coupling Between Tests and Code Units. In *Proceedings of the 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE 2022)* (pp. 133–144). IEEE. <https://doi.org/10.1109/ISSRE55969.2022.00023>
13. Su, R., & Li, X. (2024). Modular Monolith: Is This the Trend in Software Architecture? In *Proceedings of the 1st International Workshop on New Trends in Software Architecture (SATrends 24)* (article ID 4, 6 pages). ACM. <https://doi.org/10.1145/3643657.3643911>
14. Wang, Y., Bornais, S., & Rubin, J. (2024). Microservice Decomposition Techniques: An Independent Tool Comparison. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE 24)* (pp. 1295–1307). ACM. <https://doi.org/10.1145/3691620.3695504>
15. Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability*, 22(2), 67–120. <https://doi.org/10.1002/stv.430>
16. Zhong, C., Zhang, H., Li, C., Huang, H., & Feitosa, D. (2023). On measuring coupling between microservices. *Journal of Systems and Software*, 200, article ID 111670. <https://doi.org/10.1016/j.jss.2023.111670>

V. V. Kukhar¹, Yu. V. Shabatura²

¹ Ukrainian National Forestry University, Lviv, Ukraine

² Hetman Petro Sahaidachny National Army Academy, Lviv, Ukraine

GRAPH-BASED DECOMPOSITION OF MONOLITHIC INFORMATION SYSTEMS FOR RELEASE ISOLATION AND SELECTIVE REGRESSION TESTING

Monolithic architectures present a recurring challenge for teams working under continuous delivery: even a small, well-scoped code change forces a full regression suite to run, stretching pipeline wait times and slowing release cadence in ways that compound over time. This study develops and validates a graph-based decomposition method for monolithic information systems that supports isolated releases and selective regression test execution – without requiring an architectural shift to microservices. The method builds a Weighted Functional Responsibility Graph in which each edge carries a three-component weight vector: a structural weight from static code analysis, a domain weight capturing shared business entities, and a temporal weight derived from version control co-change history using a Jaccard similarity metric over commit sets. The method was tested on a model system of 78 functional units, 312 dependency edges, and 1,200 automated tests, against a baseline full regression run of 47 minutes. The temporal weight proved the most diagnostically informative component: 41 of 312 edges carried a non-zero temporal weight while having zero structural and domain weights, uncovering implicit coupling between modules that static analysis alone would miss – a pattern commonly linked to hard-to-predict regression failures. A threshold-based greedy clustering algorithm decomposed the system into ten release-isolated components, bringing inter-component dependencies down to 12.2 % of total edges. Moreover, the *affecteded(v,d)* breadth-first traversal algorithm computed a minimal sufficient test set for each change scenario. Across all four evaluated scenarios – from a minimal isolated fix to concurrent changes in two heavily-coupled components – the defect detection rate held at 1.0, while test execution time fell by 59.7 % to 90.7 % relative to the full run. The results indicate that decomposition-aware test selection provides stronger reduction guarantees than file-level impact analysis while maintaining complete fault coverage. Future work should address automated weight coefficient calibration through machine learning on release history data and adaptation of the temporal analysis layer to event-driven system architectures.

Keywords: change impact analysis; temporal coupling; functional boundary detection; CI/CD pipeline optimization; modular monolith.