УДК 004.94+519.2 DOI https://doi.org/10.35546/kntu2078-4481.2025.2.2.28

Д. О. МОСКАЛИК
аспірант кафедри інженерії програмного забезпечення
Державний університет «Житомирська політехніка»
ORCID: 0000-0002-4421-9325

Д. С. АНТОНЮК
кандидат педагогічних наук, доцент,
доцент кафедри інженерії програмного забезпечення
Державний університет «Житомирська політехніка»
ORCID: 0000-0001-7496-3553

МОДЕЛЮВАННЯ ЗВ'ЯЗНОСТІ ШЛЯХОМ ІМОВІРНІСНОГО РОЗПОДІЛУ ЧАСТОТ ВНЕСЕННЯ ЗМІН ДО ПРОГРАМНИХ МОДУЛІВ ПРИ СИМУЛЯЦІЇ ПРОЦЕСУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В сучасному світі продуктивність розробки програмних систем не враховується на етапі планування при виборі типу архітектури майбутнього програмного проєкту через недостатньо точний та занадто складний процес оцінки необхідних затрат часу та ресурсів на побудову такої програмної системи. Для забезпечення можливості дослідження впливу різних типів архітектури програмних систем на продуктивність їх розробки шляхом симуляції відповідних процесів розробки програмного забезпечення, в даній роботі досліджується підхід до моделювання зв'язності між модулями програмної системи на основі аналізу історії внесення змін до 8 модульних монолітних та мікросервісних проєктів з відкритим вихідним кодом. В результаті дослідження модульної структури та історії змін вихідного коду виявлено наявність логнормального імовірнісного розподілу як розміру програмних модулів, так і частоти їх модифікації, характер залежності між якими є лінійним, але параметри кореляції відрізняються між різними проєктами. Аналіз зв'язності модулів на основі видобування правил асоціації з історії їх незалежних та одночасних змін виявив степеневу функцію залежності середніх значень як вхідної, так і вихідної зв'язності від частоти зміни модуля, але параметри такої залежності також різні для різних проєктів та корелюються із кількістю модулів. З метою оптимізації обчислень при симуляції процесу розробки програмного забезпечення було запропоновано використання незалежних частот зміни модулів замість детального моделювання зв'язності кожної окремої їх пари, для чого було досліджено параметри логнормального розподілу незалежних імовірностей зміни модулів для кожного проєкту та за допомогою лінійної регресії виведено коефіцієнти залежності параметрів такого розподілу від кількості модулів в програмній системі, що моделюється. Для підвищення обчислювальної ефективності симуляцій було додатково запропоновано алгоритм коригування згенерованих частот зміни модулів для збільшення імовірності внесення змін хоча б в один модуль на кожній ітерації симуляції, оскільки всі імовірності зміни модулів незалежні та існує ненульова імовірність невнесення змін в жоден модуль.

Ключові слова: *моделювання, симуляція, модульність, еволюційна зв'язність, розробка програмного забезпечення, логнормальний розподіл, алгоритм.*

D. O. MOSKALYK
Postgraduate Student at the Department of Software Engineering
Zhytomyr Polytechnic State University
ORCID: 0000-0002-4421-9325

D. S. ANTONIUK
Candidate of Pedagogical Sciences, Associate Professor,
Associate Professor at the Department of Software Engineering
Zhytomyr Polytechnic State University
ORCID: 0000-0001-7496-3553

SOFTWARE MODULES COUPLING MODELLING IN SOFTWARE DEVELOPMENT PROCESS SIMULATIONS USING PROBABALISTIC DISTRIBUTION OF MODULE CHANGE FREQUENCIES

In today's world, the software development productivity is not considered at the planning stage when choosing the architecture type for a future software project due to an inaccurate and overly complex process of estimating the necessary time and resources needed to build such a system. To enable research on the impact of different types of software architectures on productivity through simulation of corresponding software development processes, this study examines

the approach to modeling the coupling between modules of a software system based on an analysis of the change history of 8 modular monolithic and microservice open-source projects. As a result of studying the modular structure and change history of the source code, it was found that there is a lognormal probability distribution both for the size of the software modules and their modification frequency, with linear dependence between them but different correlation parameters for different projects. The analysis of modules coupling based on the association rules mining from the history of their independent and simultaneous changes revealed a power function as the relationship between average values of both input and output coupling and the module change frequency, but the parameters of such a relationship also differed for different projects and were found to correlate with the number of modules in the software system being modeled. To optimize calculations during the simulation of the software development process, it was proposed to use independent module change frequencies instead of detailed modeling of each pair's coupling, which involved investigating the parameters of the lognormal distribution of independent module change probabilities for each project and using linear regression to derive the coefficients of the dependence of the parameters of such a distribution on the number of modules in the software system. To further enhance computational efficiency of simulations, an algorithm for adjusting the generated module change frequencies was additionally proposed to increase the probability of at least one module modification being made at each iteration of simulation, as all change probabilities are independent and there is a non-zero chance of not making any changes to any module.

Key words: *modeling, simulation, modularity, evolutionary coupling, software development, lognormal distribution, algorithm.*

Постановка проблеми

В сучасному світі існують декілька найбільш поширених типів архітектури серверних програмних систем, що визначають правила організації вихідного коду, такі як багатошарова монолітна, модульна монолітна, а також сервісно-орієнтована та мікросервісна архітектури. При плануванні майбутньої програмної системи та виборі відповідної архітектурної моделі, на додачу до необхідності підтримки всіх обов'язкових функціональних вимог, мають також бути враховані багато різних нефункціональних факторів, таких як безпека, швидкодія, надійність, масштабованість та інші. В той же час, продуктивність процесу розробки такої системи не впливає на сам вибір архітектурної моделі, а враховується лише як наслідок такого вибору для розрахунку термінів та вартості створення даного програмного продукту. Причиною такої поведінки може бути недостатньо точний та занадто складний процес оцінки необхідних затрат часу та ресурсів на побудову запланованої програмної системи. Крім того, для врахування вартості розробки системи при виборі архітектурної моделі, оцінка необхідних затрат має бути розрахована для декількох різних архітектурних рішень, що робить сам процес такої оцінки ще більш складним та вартісним.

Можливим рішенням для врахування продуктивності процесу розробки на етапі вибору типу архітектури може стати наявність відносних порівняльних характеристик продуктивності розробки для різних архітектурних моделей.

Одним з шляхів дослідження таких порівняльних характеристик є історичний досвід побудови програмних систем одразу в декількох різних архітектурних моделях, проте такий підхід має ряд недоліків. Якщо одна і та ж програмна система буде побудована одними і тими ж розробниками спочатку в одній архітектурі, а потім в іншій, то продуктивність розробки останньої буде вища за об'єктивну через вже наявний досвід створення першої. Якщо ж та сама програмна система в різних архітектурних моделях буде створюватись паралельно різними командами розробки, то на результуючу продуктивність будуть мати вплив як професійні якості та досвід індивідуальних розробників, так і їх синергія в командній роботі. В обох випадках складно буде оцінити об'єктивність отриманих результатів через велику кількість невідомих змінних, а також варто враховувати високу вартість експериментів такого роду.

Іншим варіантом дослідження порівняльних характеристик продуктивності процесу розробки в контексті різних типів архітектури є визначення відмінностей між різними архітектурними моделями, що мають вплив на продуктивність розробки, та проведення низькорівневих симуляцій процесів розробки за допомогою дискретно-подійного та агентно-орієнтованого моделювання з врахуванням таких відмінностей.

Зв'язок модульності з різними метриками процесу розробки програмного забезпечення був досліджений ще декілька десятиліть тому, де було виявлено позитивний вплив модульної організації коду, слабкої зв'язності назовні та сильного зчеплення всередині модулів на продуктивність розробки, надійність програмної системи та зменшення вартості її подальшого обслуговування. Тому є доцільним моделювання модульної організації коду в низькорівневих симуляціях процесу розробки програмного забезпечення для врахування даного впливу на продуктивність розробки.

Аналіз останніх досліджень і публікацій

Моделювання як метод дослідження різноманітних процесів реального світу широко використовується як в науковій, так і в комерційній сферах. В працях Т. Шматковської, Т. Коробчук [1, с. 157], а також А. Сараванос (Saravanos, A.), М. Курінґа (Curinga, M. X.) [2, с. 15] досліджується важливість ролі моделювання бізнес-процесів підприємств, що дозволяє знаходити шляхи для покращення поточних процесів та впровадження інновацій.

Дослідженням процесу розробки програмного забезпечення шляхом його моделювання займались такі вітчизняні науковці, як Ю. С. Кордунова, М. Фелтіновскі, О. В. Придатко, О. О. Смотр [3, с. 29], С. Б. Приходько, Н. В. Приходько, К. О. Книрік [4, с. 50], а також іноземні дослідники, такі як А. Сараванос (Saravanos, A.), М. Курінга (Curinga, M. X.) [2, с. 15]. В даних працях процеси розробки програмних систем моделюються на рівні етапів життєвого циклу програмного проєкту та не враховують вплив низькорівневих архітектурних факторів на продуктивність розробки. При аналізі останніх наукових робіт щодо моделювання процесу розробки програмного забезпечення, Дж. Гарсія, Дж. Енрікес та інші виявили спад частоти застосування парадигми системної динаміки, високу популярність дискретно-подійного та агентно-орієнтованого модулювання, а також тенденцію до зростання їх популярності [5, с. 16].

Незважаючи на те, що факт позитивного впливу модульності на продуктивність розробки вже давно відомий, проблема вимірювання та моніторингу зв'язності мікросервісів [6, с. 6; 7, с. 15] та питання оптимальної модульності [8, с. 32728] все ще активно вивчається науковцями різних країн. Враховуючи комплексну природу взаємозалежностей різних компонентів програмної системи, що було детально висвітлено в праці «Demystifying dependence» Дж. Коппела та Д. Джексона [9, с. 49], є доцільним визначення параметрів модульності шляхом дослідження еволюційної зв'язності модулів на основі історії внесення змін до вихідного коду існуючих модульних програмних систем. Даний підхід також широко використовується для передбачення майбутніх змін коду в пов'язаних модулях, як, наприклад, в працях Т. Рольфснес, С. Ді Алезіо та інші [10, с. 202] і М. Мондал, Б. Рой на інші [11, с. 241].

Формулювання мети дослідження

Метою статті є дослідження алгоритму моделювання зв'язності між програмними модулями для низькорівневої симуляції процесу розробки програмного забезпечення на основі аналізу історії внесення змін до модульних монолітних та мікросервісних проєктів з відкритим вихідним кодом.

Викладення основного матеріалу дослідження

Для дослідження параметрів модульності були обрані 8 мікросервісних та модульних монолітних програмних проєктів з відкритим вихідним кодом на платформі GitHub. Загальна інформація про обрані проєкти відображена в таблиці 1. Кожен проєкт містить вихідний код на різних мовах програмування, включаючи клієнтський код для імплементації інтерфейсу користувача, проте для дослідження параметрів модульності саме серверних типів архітектури були проаналізовані лише ті частини вихідного коду, що організовані в модулі та представлені серверними мовами програмування. Довжина історії змін, кількість файлів та кількість рядків коду були відповідно відфільтровані та відображають лише характеристики модульних серверних частин коду, а також не включають в себе код автоматизованих тестів.

Таблиця 1

Модульні проєкти для аналізу

Назва проєкту	Код проєкту	Мова	Архітектура	К-ть модулів	К-ть змін	К-ть файлів	К-ть рядків коду
OrchardCore	https://bit.ly/45nFvFI	C#	монолітна	116	4457	4702	183 748
magento2	https://bit.ly/4k7pGaL	PHP	монолітна	220	79 454	15 543	1 120 353
ftgo-application	https://bit.ly/4dvKpml	Java, Groovy	мікросервісна	23	130	315	9004
SimplCommerce	https://bit.ly/3Sk10zB	C#	монолітна	43	509	730	27 560
Modular-monolith-by-example	https://bit.ly/4mAtyD2	C#	монолітна	11	185	754	14 534
dotnet-starter-kit	https://bit.ly/43tDMMt	C#	монолітна	25	23	288	6706
eShopOnContainers	https://bit.ly/45r4fNo	C#	мікросервісна	8	879	434	23 672
eShop	https://bit.ly/43etC3F	C#	мікросервісна	21	114	491	17 603

Так як компілятор мови C# не підтримує циклічних залежностей між окремими проєктами вихідного коду, загальновідомим шляхом вирішення таких технічних обмежень є розділення певного модуля на два окремих проєкта різного рівня абстракції. Так, більшість бібліотек фреймворку ASP.NET представлені двома пакетами: один містить лише абстракції, а інший код імплементації. Такий самий підхід використовується і в більшості кінцевих модульних проєктів, як, наприклад, в проєкті OrchardCore один окремий модуль може бути представлений двома проєктами вихідного коду, один з яких містить в своїй назві додатковий суфікс «.Core» після назви модуля. В інших проєктах також можуть використовуватись суфікси «.Abstractions» або «.Contracts». При аналізі вихідного коду в даному дослідженні такі проєкти сприймаються як складові частини відповідних модулів, а не як окремі модулі.

Як видно з таблиці 1, найбільша кількість модулів присутня в проєктах «OrchardCore» та «magento2», тому самі ці 2 проєкти доцільно проаналізувати на предмет функції розподілу розміру модулів. Гістограми розподілу розміру

модулів для обох проєктів з лінійною та логарифмічною шкалою наведені на рисунку 1. Для кращого сприйняття інформації, розмір модулів виражається у відсотках від загального об'єму модульного коду відповідного проєкту.

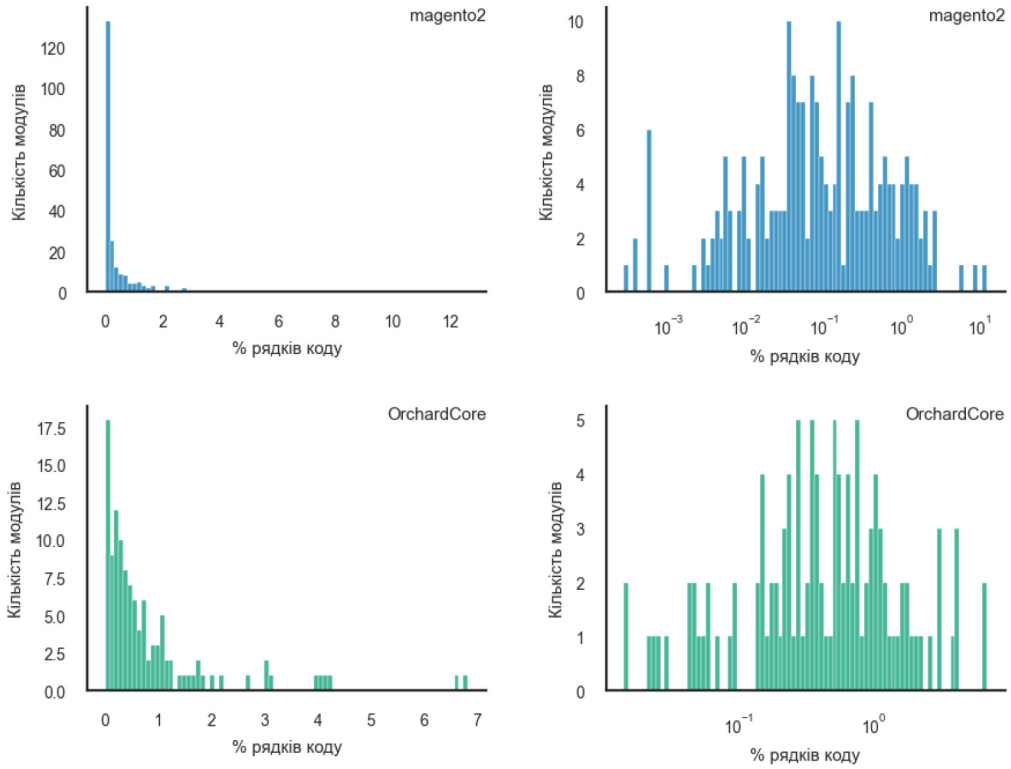

Рис. 1. Розподіл розміру модулів для проєктів «OrchardCore» та «magento2»

Так як гістограми з логарифмічною шкалою мають форму нормального розподілу, то реальний розподіл розмірів модулів даних програмних систем підпорядковується логнормальному розподілу, але з різними параметрами для різних проєктів.

Таким самим чином можна проаналізувати розподіл частот зміни різних модулів та дослідити зв'язок розміру модулів від частоти їх зміни. Частота зміни модуля може бути розрахована за допомогою формули 1, що також буде відповідати незалежній імовірності зміни модуля в кожній окремій транзакції модифікації вихідного коду системи.

$$change_frequency(A) = \frac{count(A)}{N}, \tag{1}$$

де $count(A)$ – кількість транзакцій модифікації коду, в яких вносяться зміни до модуля А, N – загальна кількість транзакцій.

Залежність частоти внесення змін в модуль від його розміру зображена на рисунку 2 у вигляді діаграми розсіювання. З даної діаграми можна зробити висновок, що залежність між розміром модуля та частотою його зміни є прямою та має лінійний характер, тобто більші за розміром модулі змінюються частіше. Також варто підкреслити, що дана кореляція притаманна як монолітній архітектурі, так і мікросервісній.

Наступним кроком є дослідження зв'язності модулів на основі історії транзакцій внесення змін до них. Це можливо зробити шляхом виявлення правил асоціації між модулями, що ґрунтується на аналізі частот зміни кожної пари модулів як окремо один від іншого, так і одночасно в рамках однієї транзакції. В методах передбачення майбутніх змін використовуються обидва параметри метрики: «support» (підтримка) та «confidence» (довіра), що розраховуються за формулами 2 та 3 [10, с. 202], проте для моделювання зв'язності достатньо буде розрахувати лише «confidence».

$$support(A \to B) = \frac{count(A \cup B)}{N}, \tag{2}$$

де count($A \cup B$) – кількість транзакцій зміни коду, в яких одночасно модифікуються модулі A і B, N – загальна кількість транзакцій.

$$confidence(A \to B) = \frac{count(A \cup B)}{count(A)}, \tag{3}$$

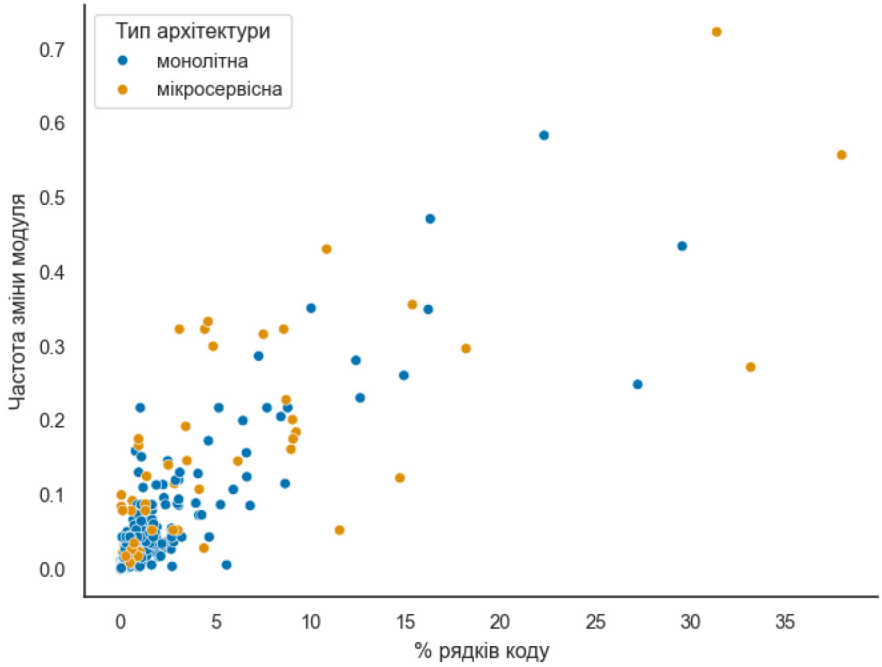

Рис. 2. Залежність частоти зміни модуля від його розміру

де $count(A \cup B)$ – кількість транзакцій зміни коду, в яких одночасно модифікуються модулі A і B, $count(A)$ – загальна кількість транзакцій, в яких змінюється модуль A.

Розраховані значення $confidence(A \rightarrow B)$ виражають вихідну зв'язність модуля A по відношенню до модуля B, або імовірність зміни модуля B при модифікації модуля A, та симетрично вхідну зв'язність модуля B по відношенню до модуля A. Для дослідження взаємозв'язку між зв'язністю модулів та їх незалежною імовірністю зміни доцільно розрахувати середні значення вихідної та вхідної зв'язності для кожного модуля та порівняти їх з частотами зміни модулів. На рисунку 3 зображені діаграми залежності середніх значень вхідної та вихідної зв'язності всіх модулів з частотою їх зміни. Дані різних проєктів зображені різними кольорами.

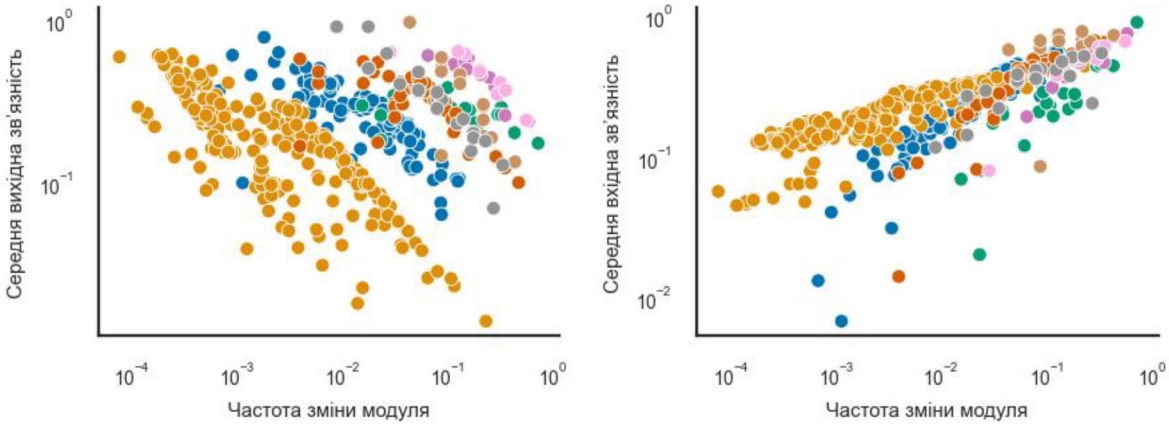

Рис. 3. Кореляція частоти зміни модулів та їх середніх значень вхідної та вихідної зв'язності

Логарифмічні шкали на діаграмах розкривають наявність степеневої функції залежності зв'язності модулів від частоти їх зміни, але параметри такої залежності різні для різних проєктів. Як видно з діаграм на рисунку 3, чим більша частота зміни коду модуля, а відповідно і його розмір, тим менша імовірність внесення змін в інші модулі та тим більша імовірність зміни такого модуля при модифікації інших модулів.

Теоретично, для моделювання зв'язності модулів в дискретно-подійних та агентно-орієнтованих симуляціях можна на кожній ітерації внесення змін до коду обирати перший модуль та, комбінуючи параметри вхідної та вихідної зв'язності, симулювати зміни в пов'язаних модулях, але у даного підходу є декілька проблем. Розраховані раніше незалежні імовірності зміни окремих модулів вже включають в себе як вхідну, так і вихідну зв'язаність з іншими модулями, впливаючи на розраховані імовірності зміни один одного, тому дані імовірності не можна використовувати

для вибору першого модуля для модифікації в певній транзакції внесення змін до вихідного коду системи. Крім того, моделювання як різної кількості модулів, так і зв'язності між кожною їх парою може бути обчислювально неефективним та сповільнювати виконання симуляцій. Також варто зауважити, що середня кількість зв'язаних модулів по всіх проєктах складає приблизно 96 % від загальної кількості модулів у відповідній системі.

Враховуючи наведені обмеження та отримані дані, альтернативним підходом до моделювання зв'язності може бути використання незалежних імовірностей внесення змін для визначення всіх модулів програмної системи, що підлягають модифікації на кожній ітерації симуляції. Як видно з рисунку 3, зв'язність модулів має степеневу залежність від частоти їх модифікації, але відрізняється від проєкта до проєкта. Так як імовірнісний розподіл розміру модулів має логнормальну форму, а частоти зміни модулів лінійно залежать від їх розміру, то закон розподілу частот модифікації модулів також має логнормальний характер. Враховуючи те, що чим менша кількість модулів в програмній системі, тим частіше кожен з них змінюється, є доцільним визначення параметрів логнормального розподілу частот модифікації модулів для кожного проєкту та дослідження їх взаємозв'язку з загальною кількістю модулів в кожному з них.

Для розрахунку коефіцієнтів логнормального розподілу частот зміни модулів окремо для кожного проєкту можна використати метод максимальної правдоподібності. На рисунку 4 зображені співвідношення отриманих коефіцієнтів розподілу відносно логарифмів кількості модулів у відповідному проєкті та їх екстраполяції, отримані шляхом лінійної регресії.

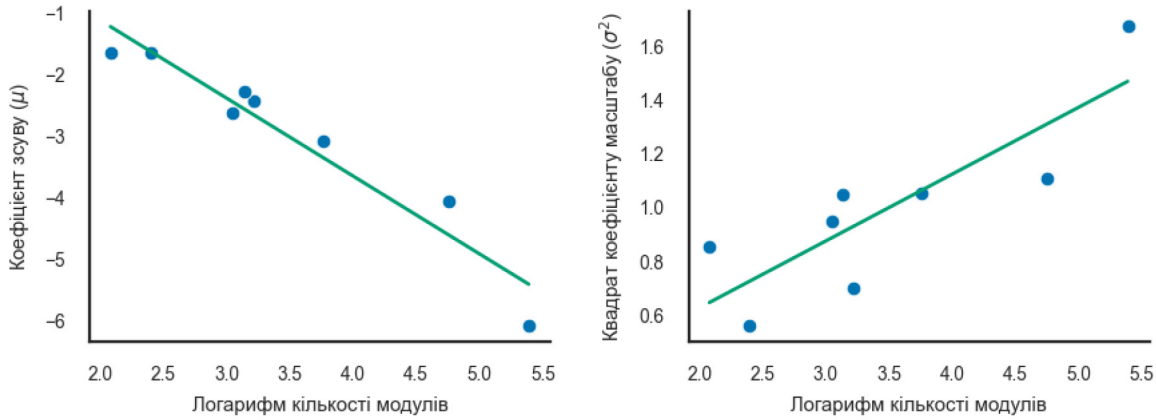

Рис. 4. Залежність параметрів логнормального розподілу частот зміни модулів від їх загальної кількості у відповідному проєкті

Згідно отриманих результатів, параметри логнормального розподілу частот зміни модулів від їх кількості при симуляції процесу розробки абстрактної програмної системи можуть бути розраховані за формулами 4 та 5.

$$\mu = -1.2624653134178174 \ln N + 1.407336741730882, \quad (4)$$

$$\sigma = \sqrt{0.24806030790958875 \ln N + 0.13184253926824063}, \quad (5)$$

де N – загальна кількість модулів в проєкті.

На етапі підготовки до запуску симуляції процесу розробки, виходячи з необхідної кількості модулів, мають бути розраховані параметри розподілу частот їх зміни відповідно до формул 4 і 5 та відповідно до отриманого розподілу кожен модуль отримає своє значення незалежної імовірності модифікації на кожній ітерації симуляції. Так як логнормальний розподіл – неперервний, то існує ненульова імовірність генерації частоти зміни модуля, що буде більше 100 %, тому під час генерації випадкових значень варто відсіювати всі згенеровані значення, більші за обрану верхню межу. Максимальне значення імовірності зміни модуля серед всіх проаналізованих проєктів складає 72.3 %, що також може бути використано в якості такої верхньої межі.

Так як згенеровані імовірності модифікації модулів незалежні, то існує ненульова імовірність, що на певній ітерації симуляції жоден модуль не буде змінений. В такому випадку процес відбору модулів, що підлягають модифікації на поточній ітерації симуляції має бути повторений знову, поки хоча б один модуль не увійде в поточну транзакцію внесення змін до коду системи. Спираючись на розраховані частоти зміни модулів проєктів, що аналізуються, середнє значення імовірності невнесення змін в жоден модуль серед всіх проєктів складає 5.6 %, а максимальне значення досягає 13.4 %. Для оптимізації обчислень в симуляціях процесу розробки програмного забезпечення пропонується підхід до коригування згенерованих імовірностей модифікації модулів таким чином, щоб максимально знизити імовірність невнесення змін в жоден модуль в процесі симуляції, але тим часом не спотворити розподіл частот змін між модулями. Для цього необхідно розрахувати поточну імовірність невнесення

змін в жоден модуль за формулою 6, вирахувати коригуючий коефіцієнт за формулою 7 та перерахувати поточні імовірності модифікації всіх модулів за формулою 8.

$$Q = \prod_{i}^{N} (1 - p_i), \tag{6}$$

де N – загальна кількість згенерованих модулів, p_i – імовірність внесення змін в i-тий модуль, Q – поточна імовірність невнесення змін в жоден модуль.

$$K = \left(\frac{Q'}{Q} \right)^{\frac{1}{N}}, \tag{7}$$

де N – загальна кількість згенерованих модулів, Q – поточна імовірність невнесення змін в жоден модуль, $Q' > 0$ – бажана імовірність невнесення змін в жоден модуль, K – коригуючий коефіцієнт.

$$p_i' = 1 - K \cdot (1 - p_i), \tag{6}$$

де p_i – згенерована імовірність внесення змін в i-тий модуль, K – коригуючий коефіцієнт, p_i' – скоригована імовірність зміни i-го модуля.

Висновки

У даній роботі були проаналізовані 8 модульних проєктів з відкритим вихідним кодом, як в монолітній архітектурі, так і у мікросервісному виконанні. Всі проєкти різного розміру, містять різну кількість модулів та різну довжину історії змін. В результаті дослідження модульної структури та історії змін вихідного коду виявлено наявність логнормального імовірнісного розподілу як розміру програмних модулів, так і частоти їх модифікації, характер залежності між якими є лінійним, але параметри кореляції відрізняються між різними проєктами, що може бути зумовлено як різними мовами програмування, так і різною внутрішньою складністю коду модулів. Аналіз зв'язності модулів на основі видобування правил асоціації з історії їх незалежних та одночасних змін виявив степеневу функцію залежності середніх значень як вхідної, так і вихідної зв'язності від частоти зміни модуля, але параметри такої залежності також різні для різних проєктів та корелюються із кількістю модулів. З метою оптимізації обчислень при симуляції процесу розробки програмного забезпечення було запропоновано використання незалежних частот зміни модулів замість детального моделювання зв'язності кожної окремої їх пари. Для цього було досліджено параметри логнормального розподілу незалежних імовірностей зміни модулів для кожного проєкту та за допомогою лінійної регресії виведено коефіцієнти залежності параметрів такого розподілу від кількості модулів в програмній системі, що моделюється. Так як всі імовірності зміни модулів незалежні, то існує ненульова імовірність невнесення змін в жоден модуль в процесі симуляції, що впливає на її обчислювальну ефективність. Тому було додатково запропоновано алгоритм коригування згенерованих частот зміни модулів для збільшення імовірності внесення змін хоча б в один модуль на кожній ітерації симуляції. Даний підхід до моделювання зв'язності модулів може бути застосований в дискретно-подійних та агентно-орієнтових симуляціях процесу розробки програмного забезпечення для подальшого дослідження впливу модульності на продуктивність розробки. Також є доцільним дослідження еволюції модульної структури програмних систем, оскільки кількість, розмір та зв'язність модулів не є сталими в часі і також змінюються в процесі розробки таких систем.

Список використаної літератури

1. Шматковська Т., Коробчук Т. Сучасні інформаційні та комунікаційні технології в моделюванні бізнес-процесів. *Економічний форум*. 2023. № 1(3). С. 156–161. DOI: https://doi.org/10.36910/6775-2308-8559-2023-3-20

2. Saravanos, A., Curinga, M. X. Simulating the Software Development Lifecycle: The Waterfall Model. *Applied System Innovation*. 2023. Vol. 6. P. 108. DOI: https://doi.org/10.3390/asi6060108

3. Кордунова Ю. С., Фелтіновскі М., Придатко О. В., Смотр О. О. Математичне моделювання процесу розробки спеціалізованих програмних систем безпеко-орієнтованого спрямування. *Вісник Львівського державного університету безпеки життєдіяльності*. 2023. № 27, С. 23–31. DOI: https://doi.org/10.32447/20784643.27.2023.03

4. Приходько, С. Б., Приходько, Н. В., & Книрік, К. О. Математичне моделювання трудомісткості розробки мобільних застосунків у фазі планування із врахуванням викидів. *Комп'ютерне моделювання та оптимізація складних систем*: матеріали V Міжнародної науково-технічної конференції. м. Дніпро, 6–8 листопада 2019 р. / Український державний хіміко-технологічний університет. Дніпро. 2019. С. 50. https://udhtu.edu.ua/wp-content/uploads/2023/08/cmoss_2019.pdf#page=50

5. J. A. García-García, J. G. Enríquez, M. Ruiz, C. Arévalo, A. Jiménez-Ramírez Software Process Simulation Modeling: Systematic literature review. *Computer Standards & Interfaces*. 2020. Vol. 70. P. 103425. DOI: https://doi.org/10.1016/j.csi.2020.103425

6. Panichella S., Rahman M. I., Taibi D. Structural Coupling for Microservices. *Proceedings of the 11th International Conference on Cloud Computing and Services Science*. 2021. Vol. 1. P. 280–287. DOI: https://doi.org/10.5220/0010481902800287

7. Apolinário D. R., de França B. B. A method for monitoring the coupling evolution of microservice-based architectures. *Journal of the Brazilian Computer Society*. 2021. Vol. 27(17). Article 17. DOI: https://doi.org/10.1186/s13173-021-00120-y

8. Vural H., Koyuncu M. Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice? *IEEE Access*. 2021. Vol. 9. P. 32721-32733. DOI: https://doi.org/10.1109/ACCESS.2021.3060895

9. Koppel J., Jackson D. Demystifying dependence. *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 2020. P. 48–64. DOI: https://doi.org/10.1145/3426428.3426916

10. Rolfsnes T. et al. Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2016. P. 201–212. DOI: https://doi.org/10.1109/SANER.2016.101

11. Mondal M. et al. Historank: History-based ranking of co-change candidates. *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2020. P. 240–250. DOI: https://doi.org/10.1109/SANER48275.2020.9054869

References

1. Shmatkovska, T., & Korobchuk, T. (2023). Suchasni informatsiini ta komunikatsiini tekhnolohii v modeliuvanni biznes-protsesiv [Modern information and communication technologies in business process modeling]. *Ekonomichnyi forum*, *1*(3), 156–161. https://doi.org/10.36910/6775-2308-8559-2023-3-20 [in Ukrainian]

2. Saravanos, A., & Curinga, M. X. (2023). Simulating the Software Development Lifecycle: The Waterfall Model. *Applied System Innovation*, *6*(6), 108. https://doi.org/10.3390/asi6060108

3. Kordunova Yu. S., Feltinovski M., Prydatko O. V., & Smotr O. O. (2023). Matematychne modeliuvannia protsesu rozrobky spetsializovanykh prohramnykh system bezpeko-oriientovanoho spriamuvannia [Mathematical modeling of the development process of specialized security-oriented software systems]. *Visnyk Lvivskoho derzhavnoho universytetu bezpeky zhyttiediialnosti*, *27*, 23–31. https://doi.org/10.32447/20784643.27.2023.03 [in Ukrainian]

4. Prykhodko, S. B., Prykhodko, N. V., & Knyrik, K. O. (2019). Matematychne modeliuvannia trudomistkosti rozrobky mobilnykh zastosunkiv u fazi planuvannia iz vrakhuvanniam vykydiv [Mathematical modeling of the labor intensity of mobile application development in the planning phase, taking into account emissions]. In *Kompiuterne modeliuvannia ta optymizatsiia skladnykh system: materialy V Mizhnarodnoi naukovo-tekhnichnoi konferentsii*, 6–8 lystopada 2019 roku (p. 50). Dnipro: Ukrainskyi derzhavnyi khimiko-tekhnolohichnyi universytet. https://udhtu.edu.ua/wp-content/uploads/2023/08/cmoss_2019.pdf#page=50 [in Ukrainian]

5. García-García, J. A., Enríquez, J. G., Ruiz, M., Arévalo, C., & Jiménez-Ramírez, A. (2020). Software Process Simulation Modeling: Systematic literature review. *Computer Standards & Interfaces*, *70*, 103425. https://doi.org/10.1016/j.csi.2020.103425

6. Panichella, S., Rahman, M., & Taibi, D. (2021). Structural Coupling for Microservices. In *Proceedings of the 11th International Conference on Cloud Computing and Services Science* (pp. 280–287). SCITEPRESS – Science and Technology Publications. https://doi.org/10.5220/0010481902800287

7. Apolinário, D. R. F., & de França, B. B. N. (2021). A method for monitoring the coupling evolution of microservice-based architectures. *Journal of the Brazilian Computer Society*, *27*(1), Article 17. https://doi.org/10.1186/s13173-021-00120-y

8. Vural, H., & Koyuncu, M. (2021). Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice? *IEEE Access*, *9*, 32721–32733. https://doi.org/10.1109/access.2021.3060895

9. Koppel, J., & Jackson, D. (2020). Demystifying dependence. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (pp. 48–64). SPLASH ’20: Conference on Systems, Programming, Languages, and Applications, Software for Humanity. ACM. https://doi.org/10.1145/3426428.3426916

10. Rolfsnes, T., Di Alesio, S., Behjati, R., Moonen, L., & Binkley, D. W. (2016). Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 201–212). 2016 IEEE 23rd International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE. https://doi.org/10.1109/saner.2016.101

11. Mondal, M., Roy, B., Roy, C. K., & Schneider, K. A. (2020). HistoRank: History-Based Ranking of Co-change Candidates. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 240–250). 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE. https://doi.org/10.1109/saner48275.2020.9054869