**УДК 004.415.5**
**DOI: https://doi.org/10.32782/IT/2026-2-21**

***Борис ПАНАСЮК***
*аспірант спеціальності «Інженерія програмного забезпечення», Вінницький національний технічний університет, Хмельницьке шосе, 95, м. Вінниця, Україна, 21021*
***ORCID:** 0009-0007-2064-9121*

***Наталя БАБЮК***
*кандидат технічних наук, доцент кафедри програмного забезпечення, Вінницький національний технічний університет, Хмельницьке шосе, 95, м. Вінниця, Україна, 21021*
***ORCID:** 0000-0003-0607-6340*

**Бібліографічний опис статті:** Панасюк, Б., Бабюк, Н. (2026). Порівняння класичного UML-моделювання та контрактно-орієнтованого UML-подібного моделювання для мікросервісних систем. *Information Technology: Computer Science, Software Engineering and Cyber Security*, doi: https://doi.org/10.32782/IT/2026-2-21

# ПОРІВНЯННЯ КЛАСИЧНОГО UML-МОДЕЛЮВАННЯ ТА КОНТРАКТНО-ОРІЄНТОВАНОГО UML-ПОДІБНОГО МОДЕЛЮВАННЯ ДЛЯ МІКРОСЕРВІСНИХ СИСТЕМ

***Мета** роботи. Мікросервісні архітектури підвищують автономність команд і прискорюють релізи, але збільшують кількість інтеграційних точок і частоту змін інтерфейсів. За класичного UML-моделювання контракти API (OpenAPI/AsyncAPI) часто підтримуються як окремі артефакти, що створює ризик розузгодження між діаграмами, документацією та реалізаціями. Мета роботи – розробити формальну основу контрактно-орієнтованого UML-подібного моделювання на рівні контрактів і виконати відтворюване порівняння з класичним UML-підходом за метриками часу узгодження, дефектів інтеграції, покриття документацією, дрейфу та синхронності змін.*

***Методологія.** Запропоновано метамодель контрактно-орієнтованого опису мікросервісної системи, де кожен Microservice має набір контрактів APIContract (з кратністю 1..*), які спеціалізуються на RESTContract та EventContract. RESTContract містить RESTOperation із параметрами й відповідями, а EventContract – канали Channel із повідомленнями Message; RequestParameter, Response та Message посилаються на DataSchema, сумісну з JSON Schema. Модель узгоджено з офіційними стандартами OpenAPI 3.1 і AsyncAPI 2.6 та використано як основу для автоматичної генерації специфікацій. Для трьох сценаріїв (Profile, Inventory, Search) описано два підходи: A (діаграми + окремі специфікації) та B (вбудовані схеми у зв'язки діаграми). Визначено метрики T_align, N_defects, DocCoverage, N_drift, ChangeSyncRate; сформовано протокол збору даних із репозиторію (timestamps PR/commit), трекера дефектів та інструментів перевірки дрейфу; підготовлено шаблони збору та CSV-схему.*

***Наукова новизна.** Новизна полягає у: (1) уніфікованій метамоделі контрактно-орієнтованого UML-подібного моделювання REST і event-взаємодій із виправленими кратностями, придатній до автоматичних трансформацій у OpenAPI/AsyncAPI; (2) операціоналізації «дрейфу» між моделлю та контрактами через N_drift і DocCoverage; (3) відтворюваному протоколі емпіричного порівняння процесів A і B на спільних сценаріях із парними статистичними тестами та оцінкою величини ефекту.*

***Висновки.** Синтетичні (демонстраційні) результати показують переваги підходу B: скорочення часу узгодження контрактів, зменшення дрейфу та інтеграційних дефектів, зростання покриття документацією і синхронності змін. Парний t-тест демонструє статистично значущі відмінності для всіх метрик; у випадках ненормальності різниць робастний тест Wilcoxon підтверджує висновки. Практично підхід B доцільно інтегрувати в CI/CD як джерело істини для контрактів, доповнивши quality gates на DocCoverage та N_drift і автоматизованими перевірками сумісності схем.*

***Ключові слова:** контракт-орієнтоване моделювання, інформаційна система, мікросервісна архітектура, UML-подібне середовище, високонавантажені системи, OpenAPI, AsyncAPI.*

***Borys PANASIUK***
*Postgraduate Student in Software Engineering, Vinnytsia National Technical University, 95, Khmelnytske Shose, Vinnytsia, Ukraine, 21021, boris.panasyuk@gmail.com*
***ORCID:** 0009-0007-2064-9121*

***Natalia BABIUK***
*Candidate of Technical Sciences, Associate Professor at the Department of Software Engineering, Vinnytsia National Technical University, 95, Khmelnytske Shose, Vinnytsia, Ukraine, 21021, babiuk@vntu.edu.ua*
***ORCID:** 0000-0003-0607-6340*

**To cite this article:** Panasiuk, B., Babiuk, N. (2026). Porivniannia klasychnoho UML-modeliuvannia ta kontraktno-oriientovanoho UML-podibnoho modeliuvannia dlia mikroservisnykh system [Comparison of classic UML modeling vs contract-oriented UML-like modeling for microservice systems]. *Information Technology: Computer Science, Software Engineering and Cyber Security*, doi: https://doi.org/10.32782/IT/2026-2-21

## COMPARISON OF CLASSIC UML MODELING VS CONTRACT-ORIENTED UML-LIKE MODELING FOR MICROSERVICE SYSTEMS

***Purpose.*** *Microservice architectures accelerate delivery and support team autonomy, yet increase integration points and interface change frequency. In classic UML processes, API contracts (OpenAPI/AsyncAPI) are often maintained as separate artifacts, which leads to inconsistencies between diagrams, documentation, and implementations. The purpose of this paper is to provide a formal foundation for contract-oriented UML-like modeling at the contract level and to perform a reproducible comparison against classic UML modeling using metrics of contract alignment time, integration defects, documentation coverage, model–contract drift, and change synchronization.*

***Methodology.*** *We propose a contract-level metamodel where each Microservice owns a set of APIContract instances (multiplicity 1..*), specialized into RESTContract and EventContract. RESTContract contains RESTOperation objects with parameters and responses; EventContract contains Channel objects with Message payloads. RequestParameter, Response and Message reference DataSchema compatible with JSON Schema. The metamodel is aligned with OpenAPI 3.1 and AsyncAPI 2.6 and supports automated specification generation. Across three scenarios (Profile, Inventory, Search), we describe two approaches: A (UML diagrams plus separate specification files) and B (embedding JSON Schema fragments directly into diagram links). We define and operationalize T_align, N_defects, DocCoverage, N_drift, and ChangeSyncRate, and provide a reproducible protocol for data collection from repositories (PR/commit timestamps), issue tracking, and automated drift checks, including templates and a CSV schema.*

***Scientific novelty.*** *The novelty includes: (1) a unified contract-oriented UML-like metamodel for REST and event-driven interactions with corrected multiplicities and explicit mapping to OpenAPI/AsyncAPI; (2) an operational definition of model–contract drift via measurable N_drift and DocCoverage; (3) a reproducible empirical protocol with paired statistical testing and effect size estimation.*

***Conclusions.*** *Synthetic (demonstration) results show that contract-oriented modeling is expected to reduce alignment time and drift, decrease integration defects, and increase documentation coverage and change synchronization. Paired t-tests indicate statistically significant differences across metrics; when normality of differences is violated, Wilcoxon signed-rank tests confirm the same conclusions. In practice, Approach B should be integrated into CI/CD as the source of truth for contracts, supported by quality gates on DocCoverage and N_drift and automated schema compatibility checks.*

***Key words:*** *contract-oriented modeling, information system, microservice architecture, UML-like environment, high-load systems, OpenAPI, AsyncAPI.*

**Актуальність проблеми.** Мікросервісна архітектура передбачає декомпозицію інформаційної системи на множину незалежно розгортуваних сервісів, що взаємодіють через синхронні HTTP-інтерфейси та асинхронні канали повідомлень (Newman, 2021). Сам термін «мікросервіси» був закріплений у роботі, що окреслила ключові архітектурні характеристики цього стилю: організацію навколо бізнес-можливостей, децентралізоване управління даними та незалежність розгортання (Lewis, Fowler, 2014). Наявність великої кількості API-взаємодій і часті зміни інтерфейсів роблять критичною задачу підтримки узгодженості між архітектурною моделлю, контрактами API та реалізаціями. У класичних UML-процесах діаграми ефективно відображають структуру та взаємодії, однак контрактні деталі (схеми, коди відповіді, payload подій) нерідко зберігаються окремо як OpenAPI/AsyncAPI файли або «жива» документація, що з часом дрейфує відносно фактичного стану системи. Це збільшує час міжкомандного узгодження, підвищує ймовірність інтеграційних дефектів і ускладнює

контроль якості під час еволюції системи (OMG, 2017; OpenAPI Initiative, 2021; AsyncAPI Initiative, 2023).

**Аналіз останніх досліджень і публікацій.** OpenAPI визначає стандартний, незалежний від мови опис HTTP-API, який дозволяє і людям, і інструментам розуміти API без доступу до коду та є основою для автоматичної генерації артефактів (OpenAPI Initiative, 2021). AsyncAPI аналогічно визначає машинно-читаний опис асинхронних API та підтримує інструментальну екосистему для документування й інтеграції подій (AsyncAPI Initiative, 2023). Для формального опису структур даних запитів, відповідей і повідомлень у цій роботі використано JSON Schema – стандарт, що визначає словник для валідації, документування та керування структурою JSON-документів (Wright et al., 2022). Розвиток підходів до модельно-орієнтованої роботи з Web-API підтверджують роботи з побудови модельної інфраструктури для Web-API (Ed-Douibi, Cánovas Izquierdo, Bordeleau, Cabot, 2019) та підходи MDE для мікросервісних архітектур із трансформаціями між моделями, контрактами і кодом (Rademacher et al., 2024). Окремий практичний напрямок становить контракт-тестування, де контракти використовуються як виконувані специфікації взаємодій (Pact Foundation, 2022). Великомасштабне дослідження історії змін API показало, що неконтрольовані breaking changes становлять значну частку еволюції інтерфейсів і призводять до каскадних відмов у залежних системах (Xavier et al., 2017). Сучасні методи статичного аналізу дозволяють автоматично виявляти порушення семантичного версіонування через семантичне порівняння релізів, що є важливим інструментом контролю сумісності контрактів (Zhang et al., 2022). Водночас питання формального вимірювання «дрейфу» між UML-моделлю та контрактами і зменшення дрейфу через інтеграцію контрактів у модель залишається недостатньо операціоналізованим для емпіричного порівняння процесів проєктування.

**Мета дослідження.** Розробити формальну метамодель контрактно-орієнтованого UML-подібного моделювання мікросервісних систем і виконати порівняння підходів A та B на трьох сценаріях (Profile, Inventory, Search) за метриками T_align, N_defects, DocCoverage, N_drift, ChangeSyncRate із відтворюваним протоколом збору та аналізу даних.

**Виклад основного матеріалу дослідження.**
**Формальна постановка.** Нехай MS – множина мікросервісів. Для кожного $ms \in MS$ визначено непорожню множину контрактів C(ms), тобто $|C(ms)| \geq 1$. Сукупність усіх контрактів C поділяється на дві підмножини: $C_{REST}$ – множину синхронних REST-контрактів і $C_{Event}$ – множину асинхронних контрактів подієвої взаємодії, причому $C = C_{REST} \cup C_{Event}$.

Нехай S – множина схем даних DataSchema, сумісних із JSON Schema. Для REST-контрактів вводиться множина операцій O, де кожна операція задається кортежем ⟨method,path,req,resp⟩, у якому method визначає HTTP-метод, path – шлях доступу, req – структуру запиту, а resp – множину можливих відповідей. Для контрактів подієвої взаємодії вводиться множина каналів Ch, де кожен канал задається кортежем ⟨channelName,role,message⟩, у якому channelName визначає ім'я каналу, role – роль учасника взаємодії (Publisher або Subscriber), а message – структуру повідомлення. У межах підходу B ці елементи є складовими моделі та можуть бути автоматично трансформовані у специфікації OpenAPI та AsyncAPI (OpenAPI Initiative, 2021; AsyncAPI Initiative, 2023).

Метамодель контрактно-орієнтованого UML-подібного моделювання визначає такі основні обмеження кратності: кожен об'єкт типу Microservice має щонайменше один контракт APIContract, тобто зв'язок Microservice–APIContract задається як 1..∗; залежність dependsOn між об'єктами Microservice має графовий характер і задається кратністю 0..∗↔0..∗; кожен RESTContract містить щонайменше одну операцію RESTOperation, тобто зв'язок RESTContract–RESTOperation має кратність 1..∗; для об'єкта Response зв'язок із bodySchema є необов'язковим і задається кратністю 0..1 (рис. 1).

**Сценарії та порівняння підходів A та B.**

**Сценарій Profile.** Сервіси: IdentityService, ProfileService, NotificationService.

Взаємодії: (1) реєстрація користувача (REST), (2) подія UserRegistered (Event) (рис. 2).

Підхід A (класичний UML). На діаграмі фіксується факт виклику та каналу, а схеми запиту/відповіді та payload події існують у зовнішніх файлах OpenAPI/AsyncAPI та/або у коді. Типовий ризик — несинхронне оновлення діаграми і контракту при зміні полів.

Підхід B (контрактно-орієнтований). У стрілку операції/каналу вбудовується схема даних (JSON Schema) або посилання на DataSchema. Модель стає джерелом істини, а OpenAPI/AsyncAPI генерується автоматично; це зменшує дрейф та прискорює узгодження.

**Сценарій Inventory.** Сервіси: OrderService, InventoryService, ShippingService. Взаємодії: (1) резервування складу (REST),

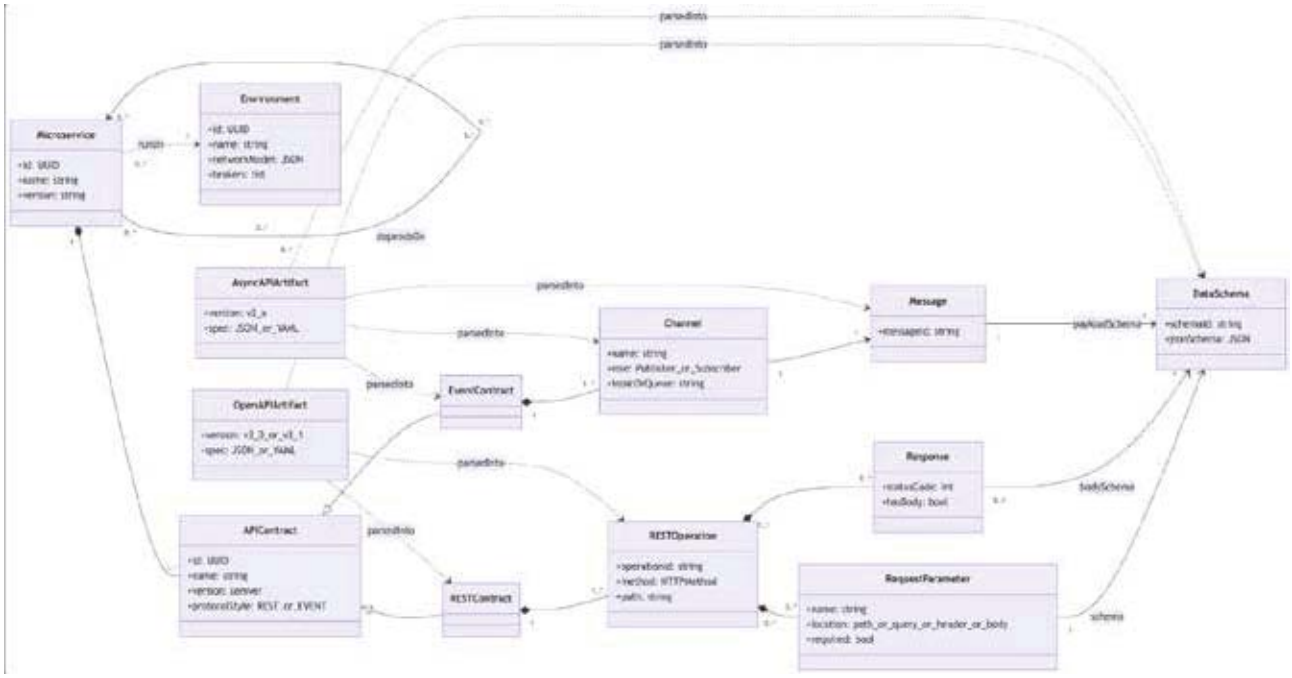

**Рис. 1. Метамодель контрактно-орієнтованого UML-подібного моделювання мікросервісних систем**


**Рис. 2. Сценарій Profile: взаємодії сервісів (REST + Event)**


**Рис. 3. Сценарій Inventory: взаємодії сервісів (REST + 2 Event)**

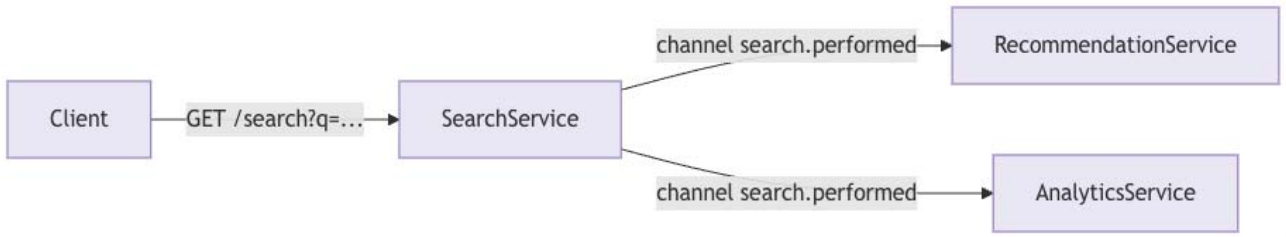


**Рис. 4. Сценарій Search: взаємодії сервісів (REST + Event fan-out)**

(2) StockReserved (Event), (3) StockRejected (Event) (рис. 3).

Підхід A. Контракти резервування та подій живуть у зовнішніх файлах; при зміні структури items/sku/qty можливі несумісності у споживачів та розузгодження з UML-діаграмою.

Підхід B. Схеми payload інтегровані в модель, використовуються для генерації специфікацій і можуть бути автоматично перевірені в CI.

**Сценарій Search.** Сервіси: SearchService, RecommendationService, AnalyticsService.

Взаємодії: (1) пошук (REST), (2) подія SearchPerformed з fan-out до двох споживачів (Event) (рис. 4).

Підхід A. Вищий ризик дрейфу через більшу кількість споживачів події: навіть невелика зміна payload може викликати ланцюгові несумісності.

Підхід B. Подія описується в моделі як контрактний артефакт і може автоматично перевірятися відносно схем і версіонування.

**Метрики, протокол експерименту та відтворюваний збір даних.** Для порівняння класичного UML-підходу та контрактно-орієнтованого UML-подібного підходу використано п'ять метрик: час узгодження контракту, кількість інтеграційних дефектів, рівень покриття документацією, кількість розсинхронів між моделлю та специфікаціями, а також частку синхронно відображених змін. Такі метрики дозволяють оцінити не лише швидкість проєктування, а й якість підтримки контрактів у процесі еволюції мікросервісної системи.

Метрика T_align характеризує час узгодження контракту між учасниками взаємодії. Вона обчислюється як різниця між моментом першої фіксації контракту або зміни контракту в репозиторії та моментом його погодження, наприклад через merge pull request, approval або релізну мітку. Менше значення T_align означає швидший процес узгодження API між командами.

Метрика N_defects відображає кількість інтеграційних дефектів, першопричиною яких є саме невідповідність контракту. До таких дефектів належать помилки у структурі схеми, невідповідність типів полів, відсутність обов'язкових параметрів, некоректні статус-коди відповіді, помилки в назвах подій, каналів або топіків. Менше значення N_defects свідчить про нижчий ризик помилок на стику сервісів.

Метрика DocCoverage показує, яка частка взаємодій, зафіксованих у моделі, має відповідний опис у специфікаціях OpenAPI або AsyncAPI. Вона обчислюється як відношення кількості взаємодій, що одночасно присутні в моделі та у специфікаціях, до загальної кількості взаємодій у моделі. Значення, близьке до 1, означає, що модель майже повністю покрита актуальною контрактною документацією.

Метрика N_drift використовується для кількісної оцінки розсинхрону між архітектурною моделлю та контрактними специфікаціями. Вона враховує три типи невідповідностей: наявність взаємодії в моделі без відповідної специфікації, наявність специфікації без відповідної взаємодії в моделі, а також ситуацію, коли взаємодія присутня в обох артефактах, але її схема, параметри або структура відповіді відрізняються. Менше значення N_drift означає кращу синхронізацію між моделлю та контрактами.

Метрика ChangeSyncRate характеризує синхронність внесення змін. Вона показує, яка частка змін контракту була своєчасно відображена в моделі протягом визначеного інтервалу, наприклад 24 годин. Високе значення ChangeSyncRate означає, що зміни в API не накопичуються окремо від архітектурної моделі, а оперативно потрапляють у спільний опис системи.

Для забезпечення відтворюваності експерименту та однозначного трактування показників доцільно зафіксувати структуру даних, на основі яких обчислюються метрики порівняння підходів A і B. У таблиці 1 наведено шаблон

Таблиця 1

**Шаблон збору даних для обчислення метрик**

| Поле | Тип | Опис |
|---|---|---|
| pair_id | int | Ідентифікатор парного прогону |
| scenario | enum | Назва сценарію |
| approach | enum | Підхід моделювання |
| contract_first_ts | datetime | Дата і час першої фіксації контракту |
| contract_approved_ts | datetime | Дата і час факту узгодження контракту |
| T_align | float | Період узгодження |
| N_defects | int | Кількість інтеграційних дефектів, спричинених саме контрактом |
| interactions_model | int | Кількість взаємодій, зафіксованих у моделі/діаграмі |
| interactions_in_spec | int | Кількість взаємодій із моделі, які реально присутні у специфікаціях |
| DocCoverage | float | Покриття документацією |
| drift_missing_spec | int | Кількість взаємодій, які є в моделі, але відсутні у специфікаціях |
| drift_missing_model | int | Кількість взаємодій, які є у специфікаціях, але відсутні в моделі |
| drift_schema_mismatch | int | Кількість взаємодій, які є і в моделі, і в специфікації, але не збігаються за контрактом |
| N_drift | int | Загальна кількість дефектів. |
| changes_total | int | Загальна кількість змін контракту |
| changes_synced | int | Кількість змін контракту, які були синхронно відображені в моделі в прийнятний термін |
| ChangeSyncRate | float | Частка синхронних змін |

збору даних, що визначає склад полів, їх типи та змістове призначення. Такий шаблон дає змогу уніфікувати фіксацію результатів для всіх сценаріїв, зменшити неоднозначність під час інтерпретації метрик і забезпечити коректність подальшого статистичного аналізу. Логіку експериментального порівняння доцільно подати у вигляді послідовності кроків, що забезпечують однакові умови для обох підходів і зменшують вплив випадкових чинників. На рис. 5 наведено відтворюваний протокол парного порівняння, у межах якого для кожного сценарію застосовуються обидва підходи моделювання, після чого обчислюються відповідні метрики та виконується їх статистичне зіставлення. Така організація експерименту дає змогу порівнювати не різні задачі, а два способи розв'язання однієї й тієї самої задачі, що підвищує коректність підсумкових висновків.

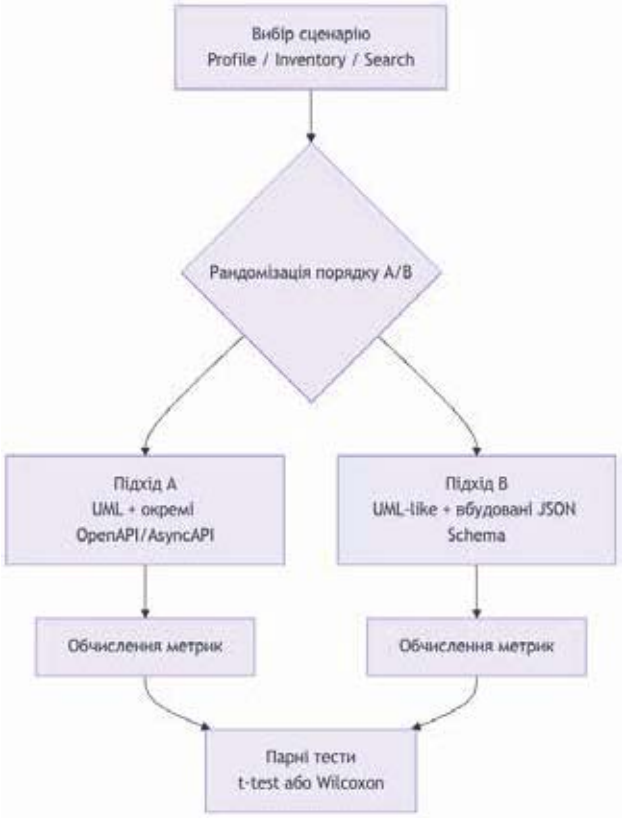


**Рис. 5. Протокол порівняння підходів A і B (парний дизайн)**

Симульовані результати, статистичний аналіз та інтерпретація

Для демонстрації методики оцінювання використано синтетичні дані 36 парних прогонів (по 12 пар на сценарій). Далі наведено зведені значення метрик у форматі середнє ± стандартне відхилення для підходів A (класичне UML моделювання з окремими специфікаціями) та B (контрактно-орієнтоване UML-подібне моделювання з інтегрованими схемами і автоматичною генерацією артефактів).

Таблиця 2

**Зведені показники для сценарію Profile (синтетичні дані)**

| Метрика | A | B |
|---|---|---|
| T_align, год | 8.08 ± 1.15 | 5.59 ± 0.64 |
| N_defects, од. | 4.08 ± 2.61 | 1.67 ± 1.37 |
| DocCoverage | 0.68 ± 0.05 | 0.86 ± 0.06 |
| N_drift, од. | 4.58 ± 0.97 | 2.42 ± 0.95 |
| ChangeSyncRate | 0.45 ± 0.10 | 0.84 ± 0.10 |

Таблиця 3

**Зведені показники для сценарію Inventory (синтетичні дані)**

| Метрика | A | B |
|---|---|---|
| T_align, год | 11.32 ± 1.47 | 7.59 ± 1.71 |
| N_defects, од. | 5.33 ± 2.37 | 3.42 ± 1.17 |
| DocCoverage | 0.70 ± 0.07 | 0.87 ± 0.07 |
| N_drift, од. | 7.08 ± 1.44 | 2.83 ± 0.67 |
| ChangeSyncRate | 0.47 ± 0.17 | 0.87 ± 0.17 |

Таблиця 4

**Зведені показники для сценарію Search (синтетичні дані)**

| Метрика | A | B |
|---|---|---|
| T_align, год | 12.91 ± 1.41 | 8.70 ± 1.64 |
| N_defects, од. | 5.42 ± 1.78 | 3.67 ± 1.61 |
| DocCoverage | 0.69 ± 0.08 | 0.86 ± 0.06 |
| N_drift, од. | 9.67 ± 1.70 | 3.17 ± 0.83 |
| ChangeSyncRate | 0.49 ± 0.12 | 0.78 ± 0.11 |

**Статистичний аналіз.** Оскільки порівняння виконано у парному дизайні (кожен pair_id має два запуски, A і B), для кожної метрики спочатку обчислюється різниця між підходами як d = B – A. Далі для d наводяться описові характеристики (середнє і стандартне відхилення) та оцінюється, чи відрізняється середня різниця від нуля. Якщо розподіл d є приблизно симетричним і не має виражених відхилень від нормальності, використовується парний t тест. Якщо метрика має дискретний або обмежений характер, або розподіл d є явно ненормальним, застосовується непараметричний критерій Вілкоксона для парних вибірок. Для інтерпретації практичної значущості додатково оцінюється розмір ефекту для парного дизайну як d_z, який визначається відношенням середнього значення d до стандартного відхилення d (табл. 5).

**Інтерпретація результатів.** Синтетичні дані показують, що підхід B має статистично значущу перевагу над підходом A для всіх розглянутих

Таблиця 5

**Статистичне порівняння підходів A та B (парний дизайн)**

| Метрика | Δ (B−A) | t | p_t | d_z | p_SW | p_W |
|---|---|---|---|---|---|---|
| T_align, год | −3.476 | −9.935 | $1.0\cdot10^{-11}$ | −1.656 | 0.281 | $7.28\cdot10^{-10}$ |
| N_defects, од. | −2.028 | −3.993 | $3.19\cdot10^{-4}$ | −0.665 | 0.122 | $6.89\cdot10^{-4}$ |
| DocCoverage | +0.176 | +12.008 | $5.7\cdot10^{-14}$ | +2.001 | $3.88\cdot10^{-3}$ | $4.72\cdot10^{-7}$ |
| N_drift, од. | −4.083 | −13.190 | $3.83\cdot10^{-15}$ | −2.198 | 0.092 | $1.53\cdot10^{-7}$ |
| ChangeSync Rate | +0.363 | +13.932 | $7.61\cdot10^{-16}$ | +2.322 | 0.447 | $1.81\cdot10^{-7}$ |

метрик. Найбільші ефекти спостерігаються для DocCoverage, N_drift та ChangeSyncRate, що узгоджується з припущенням про зменшення розсинхрону між моделлю і специфікаціями при вбудовуванні контрактів у модель та автоматичній генерації артефактів. Для метрик, де розподіл парних різниць може відхилятися від нормального або є природні обмеження (наприклад, DocCoverage як частка в інтервалі [0; 1]), висновки додатково підтверджуються критерієм Вілкоксона, що підсилює надійність інтерпретації.

**Загрози валідності та практичні рекомендації.** Підхід B зменшує ефект «подвійного ведення» артефактів, коли діаграма підтримується окремо від контрактів, а специфікації оновлюються із запізненням. Інтеграція схем у модель знижує імовірність розузгодження, спрощує верифікацію змін і скорочує час узгодження між учасниками. У сценаріях із подієвим розгалуженням споживачів (Search) очікуваний виграш є більшим, оскільки навіть невеликі зміни payload впливають на ширше коло залежних компонентів і швидше накопичують дрейф. У сценаріях із критичними транзакційними кроками (Inventory) помилки у структурі даних (наприклад, items, qty, sku) здатні багаторазово відтворюватися в похідних інтеграціях і призводити до дефектів на стику сервісів, тому контроль узгодженості схеми через контрактно-орієнтовану модель дає найбільшу інженерну користь.

Внутрішня валідність може порушуватися через різний рівень досвіду виконавців і ефект навчання, що впливає на T_align та N_defects. Зменшити цей вплив дозволяють парний дизайн і рандомізація порядку виконання A та B. Конструктна валідність пов’язана з можливими помилками класифікації дефектів, зокрема віднесенням проблем до «контрактних» або «неконтрактних». Для зниження ризику доцільно застосовувати формалізований чек-лист ознак контрактного дефекту та виконувати аудит частини випадків незалежним рецензуванням. Зовнішня валідність обмежується тим, що три сценарії не покривають усіх доменних особливостей, зокрема специфіки потокової обробки або регуляторних вимог. Зменшення цього обмеження можливе шляхом розширення набору кейсів сценарієм з версіонуванням, backward compatibility та політиками обмеження запитів. Висновкова валідність може страждати через «важкі хвости» у часових метриках і обмеженість часток інтервалом [0; 1]. Тому поряд із параметричними тестами доцільно застосовувати робастні альтернативи (критерій Вілкоксона), а також перевірку чутливості висновків при різних припущеннях.

Для систем із частими релізами та великою кількістю інтеграційних контрактів доцільно застосовувати підхід B як основне джерело істини для інтерфейсів, забезпечивши автоматичну генерацію OpenAPI та AsyncAPI й регулярний контроль розсинхрону між моделлю та специфікаціями. У CI/CD доцільно вводити порогові критерії якості, зокрема мінімальне значення DocCoverage та максимальне значення N_drift на реліз або на сервіс, доповнивши їх автоматичною валідацією JSON Schema для request, response та payload. Для посилення гарантій сумісності між споживачами та провайдерами інтерфейсу підхід B доцільно поєднувати з практиками контрактного тестування, реалізованими, зокрема, інструментами Pact (Pact Foundation, 2022) та Spring Cloud Contract (Dudczak та ін., б. д.).

**Висновки та перспективи подальших досліджень.** У роботі запропоновано метамодель контрактно-орієнтованого UML-подібного моделювання мікросервісних систем, що охоплює REST-інтерфейси та подієві взаємодії, а також визначено формальні метрики для оцінювання узгодженості моделі зі специфікаціями та якості процесу внесення змін. Описано три сценарії (Profile, Inventory, Search) і сформульовано протокол порівняння підходу A, де UML-діаграми та контракти підтримуються окремо, і підходу B, де схеми інтегровані у зв’язки моделі та автоматично трансформуються у специфікації. Синтетичні результати ілюструють очікувані переваги підходу B за часом узгодження, зменшенням дрейфу та дефектів інтеграції, а також підвищенням покриття документацією і синхронності змін,

причому висновки підтверджуються парним статистичним аналізом і оцінками розміру ефекту.

Подальші дослідження доцільно спрямувати на перевірку протоколу на реальних репозиторіях і командах, розширення аналізу на еволюцію контрактів із класифікацією breaking changes та оцінюванням сумісності версій, а також інтеграцію метрик DocCoverage і N_drift у практики управління архітектурою та DevOps процеси як регулярних показників якості контрактної взаємодії.

**ЛІТЕРАТУРА:**

1. AsyncAPI Initiative. AsyncAPI Specification. Version 2.6.0. 2023. URL: https://github.com/asyncapi/spec/blob/v2.6.0/spec/asyncapi.md (дата звернення: 16.04.2026).

2. OpenAPI Initiative. OpenAPI Specification v3.1.0. 2021. URL: https://spec.openapis.org/oas/v3.1.0.html (дата звернення: 16.04.2026).

3. Wright A., Andrews H., Hutton B., Dennis G. JSON Schema: A Media Type for Describing JSON Documents (Draft 2020-12). 2022. URL: https://json-schema.org/draft/2020-12/json-schema-core.html (дата звернення: 16.04.2026).

4. Object Management Group. Unified Modeling Language (UML). Version 2.5.1. 2017. URL: https://www.omg.org/spec/UML/2.5.1 (дата звернення: 16.04.2026).

5. Rademacher F., Wizenty P., Sorgalla J., Sachweh S., Zündorf A. Model-Driven Engineering of Microservice Architectures – The LEMMA Approach. Ernst Denert Award for Software Engineering 2022: Practice Meets Foundations. Cham: Springer Nature Switzerland, 2024. P. 105–147. DOI: 10.1007/978-3-031-44412-8_5.

6. Newman S. Building Microservices. 2nd ed. Sebastopol, CA: O'Reilly Media, 2021.

7. Lewis J., Fowler M. Microservices: a definition of this new architectural term. 2014. URL: https://www.martinfowler.com/articles/microservices.html (дата звернення: 16.04.2026).

8. Ed-Douibi H., Cánovas Izquierdo J. L., Bordeleau F., Cabot J. WAPIml: Towards a Modeling Infrastructure for Web APIs. Proceedings of the 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C 2019). 2019. P. 748–752. DOI: 10.1109/MODELS-C.2019.00116.

9. Pact Foundation. Pact Docs (Consumer-Driven Contract Testing). 2022. URL: https://docs.pact.io/ (дата звернення: 16.04.2026).

10. Dudczak A., Düsterhöft M., Grzejszczak M. та ін. Spring Cloud Contract Reference Documentation [б. д.]. URL: https://docs.spring.io/spring-cloud-contract/docs/current/reference/htmlsingle/ (дата звернення: 16.04.2026).

11. Xavier L., Brito A., Hora A., Valente M. T. Historical and impact analysis of API breaking changes: A large-scale study // 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). 2017. P. 138–147. DOI: 10.1109/SANER.2017.7884616.

12. Zhang L., Liu C., Xu Z., Chen S., Fan L., Chen B., Liu Y. Has My Release Disobeyed Semantic Versioning? Static Detection Based on Semantic Differencing. 37th IEEE/ACM International Conference on Automated Software Engineering (ASE 2022). 2022. DOI: 10.1145/3551349.3556956.

**REFERENCES:**

1. AsyncAPI Initiative. (2023). AsyncAPI specification (Version 2.6.0). Retrieved from: https://github.com/asyncapi/spec/blob/v2.6.0/spec/asyncapi.md

2. OpenAPI Initiative. (2021). OpenAPI specification (Version 3.1.0). Retrieved from: https://spec.openapis.org/oas/v3.1.0.html

3. Wright, A., Andrews, H., Hutton, B., & Dennis, G. (2022). JSON Schema: A media type for describing JSON documents (Draft 2020-12). Retrieved from: https://json-schema.org/draft/2020-12/json-schema-core.html

4. Object Management Group. (2017). Unified Modeling Language (UML) (Version 2.5.1). Retrieved from: https://www.omg.org/spec/UML/2.5.1

5. Rademacher, F., Wizenty, P., Sorgalla, J., Sachweh, S., & Zündorf, A. (2024). Model-driven engineering of microservice architectures: The LEMMA approach. In Ernst Denert Award for Software Engineering 2022: Practice Meets Foundations (pp. 105–147). Cham: Springer Nature Switzerland. DOI: 10.1007/978-3-031-44412-8_5.

6. Newman, S. (2021). Building microservices (2nd ed.). Sebastopol, CA: O'Reilly Media.

7. Lewis, J., & Fowler, M. (2014). Microservices: A definition of this new architectural term. Retrieved from: https://www.martinfowler.com/articles/microservices.html

8. Ed-Douibi, H., Cánovas Izquierdo, J. L., Bordeleau, F., & Cabot, J. (2019). WAPIml: Towards a modeling infrastructure for Web APIs. Proceedings of the 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C 2019) (pp. 748–752). DOI: 10.1109/MODELS-C.2019.00116.

9. Pact Foundation. (2022). Pact docs: Consumer-driven contract testing. Retrieved from: https://docs.pact.io/

10. Dudczak, A., Düsterhöft, M., Grzejszczak, M. et al. (n.d.). Spring Cloud Contract reference documentation. Retrieved from: https://docs.spring.io/spring-cloud-contract/docs/current/reference/htmlsingle/

11. Xavier, L., Brito, A., Hora, A., & Valente, M. T. (2017). Historical and impact analysis of API breaking changes: A large-scale study. 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 138–147). DOI: 10.1109/SANER.2017.7884616.

12. Zhang, L., Liu, C., Xu, Z., Chen, S., Fan, L., Chen, B., et al. (2022). Has my release disobeyed semantic versioning? Static detection based on semantic differencing. 37th IEEE/ACM International Conference on Automated Software Engineering (ASE 2022). DOI: 10.1145/3551349.3556956.



Дата першого надходження статті до видання: 17.04.2026
Дата прийняття статті до друку після рецензування: 11.05.2026
Дата публікації (оприлюднення) статті: 29.05.2026