https://doi.org/10.31891/2307-5732-2025-357-39
УДК 004.8:004.41

МОРГУН АРТЕМ
Державний університет «Київський авіаційний інститут»
https://orcid.org/0009-0002-0027-1225
e-mail: 2175566@stud.kai.edu.ua
ГІЗУН АНДРІЙ
Державний університет «Київський авіаційний інститут»
https://orcid.org/0000-0002-2974-6987
e-mail: andriy.gizun@npp.kai.edu.ua

КОНЦЕПТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПРОЄКТУВАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

У статті представлено концепцію програмного забезпечення для комплексного проєктування мікросервісної архітектури (MSA) із використанням технологій штучного інтелекту (ШІ). Запропонований підхід охоплює ключові етапи життєвого циклу системи: реконструкцію чинної архітектури у формальну модель, обробку вимог у природній мові, трансформацію моделі та автоматичну генерацію DevOps-артефактів. Центральним елементом є модель, описана мовою архітектурного моделювання, яка забезпечує зв'язність компонентів і підтримує візуалізацію, аналіз і повторне використання архітектурних рішень. Використання ШІ (LLM, NLP) дозволяє автоматизувати процеси декомпозиції, рефакторингу та адаптації архітектури до нових вимог. Запропонований концепт сприяє підвищенню гнучкості, зменшенню витрат на підтримку та інтеграції архітектури в DevOps-процеси. У роботі також окреслено перспективи розробки спеціалізованої мови моделювання та створення інструменту для його практичного застосування.

***Ключові слова**: мікросервісна архітектура, штучний інтелект, архітектурне моделювання, реконструкція програмного забезпечення*

MORHUN ARTEM
GIZUN ANDRII
State University "Kyiv Aviation Institute"

CONCEPT OF SOFTWARE FOR MICROSERVICE ARCHITECTURE DESIGN WITH USAGE OF ARTIFICIAL INTELLIGENCE

This paper presents a comprehensive concept of software designed for the holistic development of microservice architectures (MSA) empowered by artificial intelligence (AI). The proposed solution addresses major challenges in MSA by automating key stages of its lifecycle: reconstructing of an existing deployed system into a formal model, interpretation of user-defined requirements written in natural language, architectural model transformation, and automated generation of infrastructure artifacts compatible with DevOps practices. At the core of the approach lies a formal architecture model described using a modeling, capable of capturing service semantics, communication protocols, persistence strategies, scalability mechanisms, and service ownership. The system leverages AI technologies - particularly large language models (LLMs) and natural language processing (NLP) - to analyze contextual information and derive architectural decisions such as service decomposition, refinement, or restructuring. Unlike traditional diagram-based tools, the proposed concept ensures traceability of changes, semantic consistency, support for iterative updates, and seamless integration with version control and deployment pipelines. The solution also features an interactive graphical user interface based on the C4 model, enabling engineers to visualize the system at different abstraction levels and to inspect architectural alternatives, including rejected ones, which are stored for historical traceability. Existing AI-based tools for MSA design often remain at the proof-of-concept level and do not provide formal outputs. This work addresses these limitations by introducing a complete conceptual framework that integrates AI reasoning, formal modeling, and DevOps automation. Future work will focus on the development of the modeling language suitable for MSA, implementation of a software prototype, and experimental validation in industrial case studies.

***Keywords**: Microservice Architecture, Artificial Intelligence, Architecture Modeling, Software Architecture Reconstruction*

Стаття надійшла до редакції / Received 11.07.2025
Прийнята до друку / Accepted 15.08.2025

Постановка проблеми

Мікросервісна архітектура (MSA) є сучасним підходом до проєктування програмного забезпечення (ПЗ) для розподілених систем, який широко застосовується технологічними компаніями. Популярність цього підходу зросла у 2014 році, а з 2015 року його впровадження стало масовим [1]. Водночас сам термін «мікросервісна архітектура» був уперше запропонований ще у 2011 році на конференції архітекторів ПЗ для окреслення спільної ідеї, що охоплювала поширені на той час шаблони проєктування [2]. MSA часто розглядають як еволюційний розвиток сервісно-орієнтованої архітектури (SOA) [3], оскільки вона пропонує вирішення низки проблем, характерних для монолітних систем: складність розгортання, жорстка залежність між модулями, обмеженість паралельної розробки та неефективне масштабування. Попри те, що SOA декларувала подолання цих викликів, її практичне застосування ускладнювалося відсутністю єдиної концептуальної основи. Це призводило до надмірно складних рішень, зокрема через використання корпоративних шин даних (ESB) і важковагових протоколів, таких як SOAP. Натомість MSA, спираючись на досвід застосування SOA, акцентує на незалежності сервісів і простих механізмах взаємодії, що знижує складність системи та підвищує її гнучкість [4].

Попри численні переваги, впровадження мікросервісної архітектури (MSA) супроводжується низкою серйозних викликів, які відзначають як дослідники, так і практики. Основні труднощі зосереджуються навколо таких аспектів: ідентифікація сервісів, тестування, міжсервісна комунікація, оцінка продуктивності, розгортання, моніторинг, безпека та декомпозиція [5, 6]. Ці фактори свідчать про складність ефективної реалізації MSA, що в окремих випадках змушує компанії відмовлятися від неї на користь монолітної архітектури. Причинами такого кроку часто виступають висока вартість розробки й підтримки, зниження продуктивності порівняно з монолітом, ускладнена координація між командами, а також невідповідність організаційної структури принципам архітектури - відповідно до закону Конвея [7].

Проєктування мікросервісної архітектури (MSA) з метою максимально ефективного використання її переваг є складним і багатоетапним процесом. У контексті сучасної розробки програмного забезпечення, яка здебільшого базується на методології Agile з її акцентом на часте розгортання функціональних модулів [1], архітектурне проєктування не обмежується початковим етапом, а триває впродовж усього життєвого циклу системи. Це зумовлює постійну потребу в аналізі поточного стану архітектури та прийнятті рішень щодо створення нових або вилучення неактуальних мікросервісів. Прийняття таких рішень вимагає від інженерів глибоких знань, технічної обізнаності та аналітичного мислення. Найбільш поширеним підходом до візуалізації MSA є використання блок-схем або діаграм [8], у яких компоненти системи подаються у вигляді фігур із описом функціональних властивостей. Одним із найвідоміших методів є модель C4, що включає чотири рівні [9]:

1. Context - взаємодія користувачів із системою;
2. Container - основні складові, як-от бази даних, веб- та мобільні застосунки;
3. Component - деталізація контейнерів: контролери, сервіси, запити;
4. Code - представлення класів та їх зв’язків.

Враховуючи специфіку проєктування MSA, рівні Component та Code відносяться до деталей реалізації відповідно до обраної мови програмування й, відповідно, не є основним фокусом у даному концепті.

Популярні інструменти для побудови таких діаграм — Miro, Lucidchart, Draw.io. Втім, ці засоби мають суттєвий недолік: відсутність логічної зв’язаності між елементами моделі. Будь-які зміни потребують ручного оновлення діаграм та додаткової перевірки на коректність і повноту, що ускладнює підтримку архітектурної документації та збільшує ризик її застарівання.

Одним із актуальних підходів до проєктування мікросервісної архітектури (MSA) є реконструкція наявної архітектури, її аналіз і подальше внесення змін. У роботі [10] було проаналізовано 37 інструментів для відновлення архітектури, що відрізняються методами, призначенням і ступенем автоматизації. Дослідники наголошують на тому, що ці інструменти потребують подальшої валідації в реальних промислових умовах, оскільки відсутній уніфікований підхід до їх застосування. Серед проблем також виокремлюються складність використання та обмежена підтримка окремих мов програмування.

Альтернативним підходом до проєктування є створення моделі MSA-системи з нуля. У дослідженні [11] автор запропонував мову моделювання Komponents, яка дає змогу формалізувати архітектуру у вигляді базових компонентів (наприклад, веб-серверів), файлових систем, з’єднувачів та композицій. Komponents включає низку інструментів для побудови моделей, їх трансформації у формат Infrastructure-as-Code (IaC) та подальшого розгортання за допомогою Docker Compose або Kubernetes. До обмежень Komponents належить відсутність можливості автоматичного відновлення вже існуючої архітектури, що вимагає створення моделей вручну. Крім того, мова має обмежену підтримку реактивних систем, які ґрунтуються на асинхронній взаємодії між сервісами.

Розробка нового, цілісного підходу до проєктування мікросервісної архітектури (MSA) є актуальним завданням, що дозволяє ефективніше вирішувати існуючі виклики які притаманні цьому типу архітектури. Перспективним напрямом є застосування штучного інтелекту (ШІ) для автоматизації таких процесів, як декомпозиція системи на мікросервіси, аналіз поточного стану, реконструкція архітектури у вигляді моделі та генерація рекомендацій щодо її рефакторингу. Хоча на сьогодні вже існують інструменти з використанням ШІ, зокрема MicroART [12] чи засоби для генерації компонентів MSA на основі неструктурованих вимог [13], більшість із них є лише прототипами. Як правило, результати їх роботи представлені у вигляді графічних або текстових описів, що не є формалізованими моделями та обмежують можливість подальшого аналізу. Запропонований підхід базується на формальній моделі, яка створюється на основі чинної архітектури, зберігає всі необхідні зв’язки між компонентами та може бути адаптована відповідно до нових вимог. Це дає змогу не лише виконувати гнучке моделювання та аналіз архітектури, а й автоматично генерувати артефакти для подальшого використання у DevOps-процесах. При цьому програмне забезпечення має забезпечувати зручний інтерфейс користувача та наочне графічне представлення системи.

Метою статті є представлення концепції програмного забезпечення для цілісного проєктування мікросервісної архітектури, що із використанням штучного інтелекту дозволяє відтворити поточну, вже розгорнуту архітектуру у вигляді формальної моделі та на її основі згенерувати новий стан системи відповідно до заданих вимог.

Виклад основного матеріалу

Проєктування мікросервісної архітектури є досить тривіальною, на перший погляд, задачею, але сучасні тенденції розвитку сфери інформаційних підходів породжують нові інтелектуалізовані, індивідуалізовані та інтерактивні методи та підходи реалізації такої задачі. В даній статті пропонується один з таких нових методів, а точніше його концепт, що в подальших роботах буде деталізований і описаний як цілісний метод, на основі якого можна буде реалізувати спеціалізоване програмне забезпечення.

У запропонованому підході центральну роль відіграє модель мікросервісної архітектури, описана мовою архітектурного моделювання (ADL). Ця модель повинна охоплювати ключові характеристики кожного мікросервісу, зокрема: доменне призначення, вхідні та вихідні інтерфейси, контракти взаємодії, тип і призначення бази даних, виділені ресурси, стратегії масштабування, а також команду-розробника, відповідальну за сервіс. Запропонована у [10] мова моделювання Komponents може бути розширена. Для забезпечення повноцінного опису архітектури необхідні подальші дослідження з уточнення вимог до мови моделювання, а також вибір або розробка формалізованої мови, здатної задовольнити ці вимоги. Концепція візуального представлення моделі мікросервісу наведена на рис. 1.

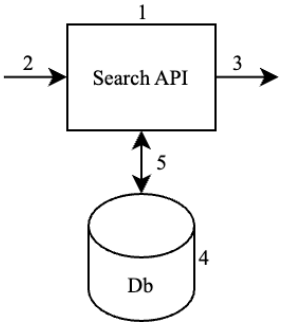

Рис. 1. Концепт візуалізації моделі мікросервісу. 1 – мікросервіс із загальною назвою; 2 – вхідний інтерфейс для взаємодії з мікросервісом; 3 – вихідний виклик іншого мікросервісу або продукування подій; 4 – база даних, що використовується мікросервісом; 5 – напрямок взаємодії мікросервісу з базою даних: зчитування чи запис інформації

Цілісний підхід до проєктування мікросервісної архітектури (MSA) включає п’ять ключових етапів:

1. Реконструкція чинної архітектури у вигляді формальної моделі. Побудована модель може бути візуалізована згідно з обраним рівнем деталізації: Context чи Container.
2. Формування плану змін на основі вимог, заданих користувачем у формі природної мови. Текстові вимоги обробляються системою ШІ для генерації відповідних модифікацій у моделі.
3. Внесення змін до архітектурної моделі згідно з отриманим планом, включаючи оновлення, додавання або вилучення окремих компонентів.
4. Генерація артефактів, які можуть бути використані в межах DevOps-процесів (наприклад, скрипти розгортання, конфігураційні файли тощо).
5. Ітеративне повернення до етапу реконструкції, що дозволяє повторно адаптувати архітектуру після реалізації змін.

Для реалізації зазначеного процесу запропоновано концепцію модульної структури програмного забезпечення, зображену на рис. 2.

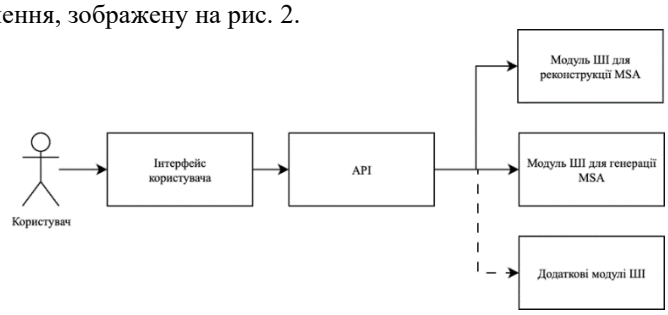

Рис. 2. Основні модулі концепту ПЗ

Таким чином концепт передбачає розробку методу, що має 4 етапи:

- Етап 1. Реконструкція та моделювання архітектури мікросервісної системи.
- Етап 2. Візуалізація моделі та задання параметрів модифікації.
- Етап 3. Модифікація базової архітектури та її моделі.
- Етап 4. Генерація інфраструктурних артефактів архітектурної моделі.

Реконструкція архітектури є першим і критично важливим етапом, що передбачає побудову моделі чинної MSA-системи. Для цього необхідно дослідити наявні інструменти для реконструкції

архітектури з метою вибору оптимального підходу, який можна адаптувати до генерації артефактів мовою моделювання із залученням штучного інтелекту (ШІ) для автоматизації процесу.

Діаграма послідовності цього етапу представлена на рис. 3. Процес ініціюється інженером через інтерфейс користувача, де обирається метод реконструкції. Далі користувач вводить необхідні дані: адреси репозиторіїв із кодом та (або) документацією, параметри серверів логів і метрик, бази даних подій, що публікують мікросервіси, а також іншу контекстну інформацію у довільній формі. Введені дані надсилаються до API, який перевіряє їх на коректність і доступність зазначених мережевих адрес. У разі виявлення помилок система повідомляє про це користувача із зазначенням деталей. Якщо перевірку пройдено, API формує запит до ШІ-модуля - ним може бути як наявна велика мовна модель (LLM), так і спеціально навчена модель, орієнтована на мову опису MSA. Результатом роботи ШІ є архітектурний опис системи у мові моделювання. Цей опис проходить валідацію, зберігається у базі даних та відображається у користувацькому інтерфейсі за обраним рівнем деталізації (наприклад, Container). Інженер має змогу переглянути результат, за потреби внести уточнення або зміни. Затверджена версія моделі передається назад від інтерфейсу користувача до API, який виконує збереження в базу даних та надсилає зворотний зв'язок до ШІ-компонента для навчання й покращення майбутніх генерацій. Після цього інженер переходить до наступного етапу проєктування.

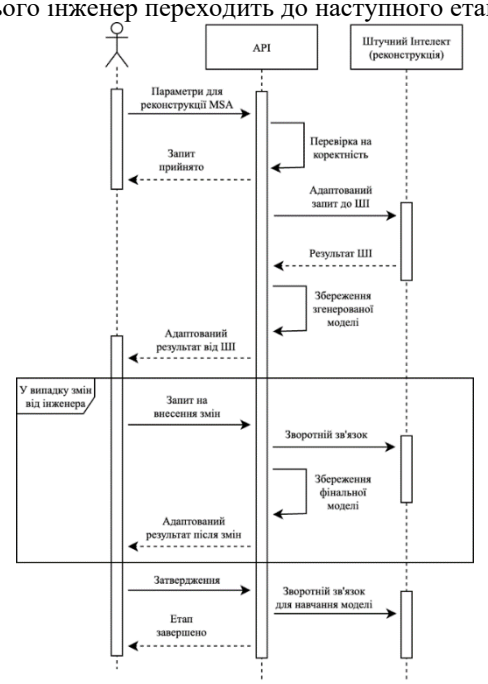

Рис. 3. Діаграма послідовності першого етапу – реконструкція MSA

Після завершення першого етапу користувачу надається графічне представлення моделі мікросервісної архітектури (MSA). Візуалізація може охоплювати як повну структуру системи (включно з усіма мікросервісами та їх складовими), так і бути обмеженою до конкретної функціональної вимоги. Наприклад, у системі онлайн-замовлення страв функціональну вимогу можна сформулювати так: «Користувач може переглядати список ресторанів у радіусі одного кілометра від поточної локації». На рис. 4 наведено приклад реалізації цієї вимоги на рівні Container моделі C4. Такий підхід дозволяє зосередитися на окремому аспекті функціональності системи у разі потреби в аналізі або внесенні змін. Це також сприяє локалізації архітектурних модифікацій та підвищує ефективність управління змінами на рівні окремих функціональних сценаріїв.

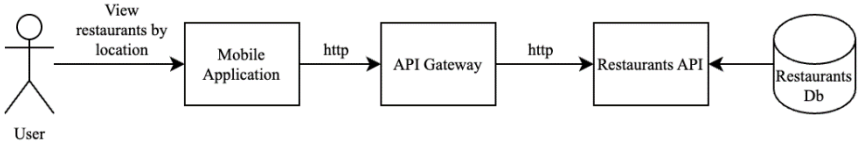

Рис. 4. Рівень Container для мікросервісів, що реалізують приклад вимоги «Користувач може бачити список ресторанів у радіусі одного кілометру від поточної локації»

Для внесення змін до моделі мікросервісної архітектури (MSA) користувачу пропонується описати бажані модифікації у спеціально відведеному текстовому полі. Зміни можуть стосуватися як конкретної функціональної вимоги, так і архітектури загалом - наприклад, при впровадженні нових сценаріїв або розширенні системи. Цей етап є другим у запропонованому підході. Введений текст обробляється за допомогою технологій обробки природної мови (Natural Language Processing) та штучного інтелекту. Завдання ШІ — виокремити семантичну інформацію, яка визначає тип змін (створення нового чи модифікація наявного мікросервісу), пов'язані сервіси, а також потребу у

додаткових компонентах. Це можливо завдяки тому, що мова моделювання MSA описує властивості та атрибути кожного мікросервісу, фактично формуючи онтологічну модель системи.

Оркестрація запитів ШІ виконується за аналогією з першим етапом. На основі обробленої інформації система формує план змін, який візуалізується для перегляду інженером: мікросервіси, що підлягають модифікації, позначаються, наприклад, помаранчевим кольором, а нові - зеленим. Таке подання забезпечує наочність змін та спрощує аналіз. Наприклад, до вимоги «Користувач може бачити список ресторанів у радіусі одного кілометра від поточної локації» може бути додана нова: «Ресторани повинні відображатися відповідно до алгоритмів, що зберігаються в окремій базі даних» (рис. 5). Інженер має змогу дослідити оновлену модель, обрати відповідний фрагмент архітектури та внести корективи у разі потреби.

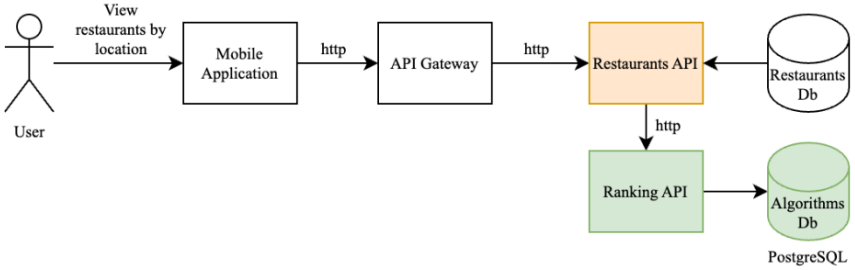

Рис. 5. Приклад відображення плану змін для нової вимоги: «Ресторани повинні бути відображені відповідно до алгоритмів, що зберігаються в окремій базі даних»

Третій етап передбачає оновлення архітектурної моделі відповідно до затвердженого плану змін. Модифікована версія зберігається у базі даних зі статусом “нерозгорнута”, що дозволяє здійснювати подальший перегляд, затвердження та аналіз до фактичної інтеграції. Інженер може використати згенерований план для формування документації, зокрема Architectural Decision Records (ADR), яка фіксує обґрунтування прийнятих рішень.

Система також підтримує генерацію альтернативних варіантів архітектурних змін за запитом інженера. Наприклад, можуть бути згенеровані декілька конфігурацій баз даних (PostgreSQL, MongoDB) або різні підходи до взаємодії між сервісами (синхронна чи асинхронна комунікація). Альтернативні моделі MSA візуально відображаються у середовищі користувача для зручного порівняння. Варіанти, що були відхилені, автоматично зберігаються в системі із зазначенням причин - вони можуть бути включені до ADR як зафіксовані, але неприйняті рішення для історичного аналізу.

Модель архітектури, описана у формальній мові, може бути збережена також у зовнішньому репозиторії для контролю версій (наприклад, GitHub), що забезпечує інтеграцію з інструментами CI/CD і повноцінне відстеження змін.

Завершальний етап передбачає генерацію інфраструктурних артефактів на основі фінальної, затвердженої архітектурної моделі. Для нових мікросервісів автоматично створюється окремий репозиторій коду (наприклад, GitHub) із шаблонною структурою, що відповідає стандартам організації. Генеруються відповідні артефакти для розгортання: docker-compose, Terraform, Ansible, Helm-скрипти - залежно від обраної хмарної інфраструктури.

Команда розробників може розпочати реалізацію нового функціоналу на основі згенерованого шаблону. Для сервісів, які підлягають оновленню, система за допомогою ШІ пропонує послідовність дій, необхідних для внесення змін. Ці дії можуть бути подані у вигляді Pull Request до відповідних репозиторіїв, що спрощує інтеграцію змін у рамках командної розробки.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

У статті запропоновано концепцію програмного забезпечення для комплексного проєктування мікросервісної архітектури (MSA), що інтегрує технології штучного інтелекту з формальним архітектурним моделюванням. Рішення охоплює повний життєвий цикл: від реконструкції чинної архітектури до генерації артефактів для розгортання, забезпечуючи автоматизацію ключових процесів та збереження узгодженості системних компонентів.

Основним науковим внеском є використання великих мовних моделей (LLM) та обробки природної мови (NLP) для аналізу вимог, трансформації моделей і підтримки прийняття архітектурних рішень. Це дозволяє знизити ризики, пов’язані з людським фактором, підвищити точність та пришвидшити адаптацію систем до нових вимог.

Запропонований підхід дозволяє зменшити витрати на підтримку архітектури, покращити гнучкість і повторне використання рішень, автоматизувати документацію та забезпечити інтеграцію з DevOps-інструментами. Впровадження концепції сприятиме покращенню якості проєктування складних розподілених систем.

Подальші дослідження зосереджуватимуться на розробці спеціалізованої мови моделювання MSA, реалізації програмного прототипу, оцінці його ефективності в прикладних проєктах та інтеграції з інфраструктурними рішеннями DevOps-екосистеми.

Література

1. Ünlü H., et al. Microservice-based Projects in Agile World: A Structured Interview // Information and Software Technology. – 2023. – P. 107334. – https://doi.org/10.1016/j.infsof.2023.107334

2. Dragoni N., Giallorenzo S., Lafuente A. L., Mazzara M., Montesi F., Mustafin R., Safina L. Microservices: Yesterday, today, and tomorrow // Present and Ulterior Software Engineering. 2017. С. 195–216. – https://doi.org/10.1007/978-3-319-67425-4_12

3. Newman S. Building microservices: Designing fine-grained systems / S. Newman. — 2-nd ed. — Sebastopol : O'Reilly Media, 2021. — 612 p. — (Computer software engineering). — ISBN 978-1-4920-3402-5.

4. Bonér J. Reactive Microservices Architecture: Design Principles for Distributed Systems / J. Bonér. — Sebastopol : O'Reilly Media, 2016. — 54 p. — ISBN 978-1-4919-5977-90.

5. Wang Y., Kadiyala H., Rubin J. Promises and challenges of microservices: an exploratory study // Empirical Software Engineering. – 2021. – Vol. 26, № 4. – Article 63. – https://doi.org/10.1007/s10664-020-09910-y

6. Söylemez M., Tekinerdogan B., Kolukısa Tarhan A. Challenges and solution directions of microservice architectures: A systematic literature review // Applied Sciences. – 2022. – Vol. 12, № 11. – Article No. 5507. – https://doi.org/10.3390/app12115507

7. Su R., Li X., Taibi D. From Microservice to Monolith: A Multivocal Literature Review // Electronics. – 2024. – Vol. 13, № 8. – Article No. 1452. – DOI: https://doi.org/10.3390/electronics13081452

8. Waseem M., Rauf A., Porres I., Uddin M. Design, monitoring, and testing of microservices systems: The practitioners' perspective // Journal of Systems and Software. – 2021. – Vol. 182. – Article No. 111061. – https://doi.org/10.1016/j.jss.2021.111061

9. Vázquez-Ingelmo A., García-Holgado A., García-Peñalvo F. J. C4 model in a Software Engineering subject to ease the comprehension of UML and the software // Proceedings of the 2020 IEEE Global Engineering Education Conference (EDUCON), Porto, Portugal, 27–30 April 2020. – IEEE, 2020. – P. 919–924. – https://doi.org/10.1109/EDUCON45650.2020.9125335

10. Bakhtin A., Li X., Soldani J., Brogi A., Cerny T., Taibi D. Tools Reconstructing Microservice Architecture: A Systematic Mapping Study // Software Architecture. ECSA 2023 Tracks, Workshops, and Doctoral Symposium / eds. Tekinerdoğan B., Spalazzese R., Sözer H., Bonfanti S., Weyns D. – Cham : Springer, 2024. – (Lecture Notes in Computer Science ; vol. 14590). – P. 3–22. – DOI: https://doi.org/10.1007/978-3-031-66326-0_1

11. Esparza-Peidro J., Muñoz-Escoí F. D., Bernabéu-Aubán J. M. Modeling microservice architectures // Journal of Systems and Software. – 2024. – Vol. 205. – Article No. 112041. – DOI: https://doi.org/10.1016/j.jss.2024.112041

12. Granchelli G., Cardarelli M., Di Francesco P., Malavolta I., Iovino L., Di Salle A. MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems // Proceedings of the IEEE International Conference on Software Architecture Workshops (ICSAW), Gothenburg, Sweden, 3–7 April 2017. – IEEE, 2017. – P. 298–302. – https://doi.org/10.1109/ICSAW.2017.9

13. Narváez D., Battaglia N., Fernández A., Rossi G. Designing Microservices Using AI: A Systematic Literature Review // Software. – 2025. – Vol. 4, № 1. – Article No. 6. – https://doi.org/10.3390/software4010006

References

1. Ünlü H., et al. Microservice-based Projects in Agile World: A Structured Interview // Information and Software Technology. – 2023. – P. 107334. – https://doi.org/10.1016/j.infsof.2023.107334

2. Dragoni N., Giallorenzo S., Lafuente A. L., Mazzara M., Montesi F., Mustafin R., Safina L. Microservices: Yesterday, today, and tomorrow // Present and Ulterior Software Engineering. 2017. С. 195–216. – https://doi.org/10.1007/978-3-319-67425-4_12

3. Newman S. Building microservices: Designing fine-grained systems / S. Newman. — 2-nd ed. — Sebastopol : O'Reilly Media, 2021. — 612 p. — (Computer software engineering). — ISBN 978-1-4920-3402-5.

4. Bonér J. Reactive Microservices Architecture: Design Principles for Distributed Systems / J. Bonér. — Sebastopol : O'Reilly Media, 2016. — 54 p. — ISBN 978-1-4919-5977-90.

5. Wang Y., Kadiyala H., Rubin J. Promises and challenges of microservices: an exploratory study // Empirical Software Engineering. – 2021. – Vol. 26, № 4. – Article 63. – https://doi.org/10.1007/s10664-020-09910-y

6. Söylemez M., Tekinerdogan B., Kolukısa Tarhan A. Challenges and solution directions of microservice architectures: A systematic literature review // Applied Sciences. – 2022. – Vol. 12, № 11. – Article No. 5507. – https://doi.org/10.3390/app12115507

7. Su R., Li X., Taibi D. From Microservice to Monolith: A Multivocal Literature Review // Electronics. – 2024. – Vol. 13, № 8. – Article No. 1452. – DOI: https://doi.org/10.3390/electronics13081452

8. Waseem M., Rauf A., Porres I., Uddin M. Design, monitoring, and testing of microservices systems: The practitioners' perspective // Journal of Systems and Software. – 2021. – Vol. 182. – Article No. 111061. – https://doi.org/10.1016/j.jss.2021.111061

9. Vázquez-Ingelmo A., García-Holgado A., García-Peñalvo F. J. C4 model in a Software Engineering subject to ease the comprehension of UML and the software // Proceedings of the 2020 IEEE Global Engineering Education Conference (EDUCON), Porto, Portugal, 27–30 April 2020. – IEEE, 2020. – P. 919–924. – https://doi.org/10.1109/EDUCON45650.2020.9125335

10. Bakhtin A., Li X., Soldani J., Brogi A., Cerny T., Taibi D. Tools Reconstructing Microservice Architecture: A Systematic Mapping Study // Software Architecture. ECSA 2023 Tracks, Workshops, and Doctoral Symposium / eds. Tekinerdoğan B., Spalazzese R., Sözer H., Bonfanti S., Weyns D. – Cham : Springer, 2024. – (Lecture Notes in Computer Science ; vol. 14590). – P. 3–22. – DOI: https://doi.org/10.1007/978-3-031-66326-0_1

11. Esparza-Peidro J., Muñoz-Escoí F. D., Bernabéu-Aubán J. M. Modeling microservice architectures // Journal of Systems and Software. – 2024. – Vol. 205. – Article No. 112041. – DOI: https://doi.org/10.1016/j.jss.2024.112041

12. Granchelli G., Cardarelli M., Di Francesco P., Malavolta I., Iovino L., Di Salle A. MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems // Proceedings of the IEEE International Conference on Software Architecture Workshops (ICSAW), Gothenburg, Sweden, 3–7 April 2017. – IEEE, 2017. – P. 298–302. – https://doi.org/10.1109/ICSAW.2017.9

13. Narváez D., Battaglia N., Fernández A., Rossi G. Designing Microservices Using AI: A Systematic Literature Review // Software. – 2025. – Vol. 4, № 1. – Article No. 6. – https://doi.org/10.3390/software4010006