Інноваційні підходи до зменшення технічного боргу в середовищі мікросервісної архітектури

Абакумов Роман Валерійович [1]

Опубліковано	Секція	УДК
09.10.2025	Соціальні та поведінкові науки	004.4’242.2:004.9

DOI: https://doi.org/10.5281/zenodo.17305325

Анотація. Актуальність дослідження зумовлена тим, що технічний борг у мікросервісних архітектурах впливає на якість цифрових продуктів, швидкість їхньої розробки та надійність супроводу. Його ускладнює інтеграцію сервісів, знижує передбачуваність релізів та підвищує ризики безпеки, що безпосередньо впливає на конкурентоспроможність підприємств. Метою статті є обґрунтування та визначення універсальних підходів до зменшення технічного боргу на основі балансування швидкості впровадження інновацій і довгострокової стабільності систем. Методологія базується на системному й порівняльному аналізі, узагальненні практик DevOps, моделюванні архітектурних рішень, використанні статичного аналізу коду, контрактного тестування, централізованого документування, моніторингу та аудиту. Результати показали, що поєднання стандартизації, моніторингу й планового рефакторингу забезпечує контрольованість боргу. Перспективи досліджень полягають у розробленні інтегрованих метрик та адаптивних моделей управління у гібридних середовищах.

Ключові слова: DevOps-практики, архітектурні рішення, контрактне тестування, стандартизація API, моніторинг продуктивності, рефакторинг, управління якістю, CI/CD-процеси.

Innovative approaches to reducing technical debt in microservices

Annotation. Relevance of the study is determined by the fact that technical debt in microservice architectures significantly affects the quality of digital products, the speed of their development, and the reliability of maintenance. As systems scale, the accumulation of debt manifests itself in increased complexity of service integration, deployment conflicts, reduced release predictability, and heightened security risks, which directly define the competitiveness of enterprises in the digital economy.

The purpose of the article is to theoretically substantiate and practically identify ways to reduce technical debt in microservice architectures based on universal management methods that ensure a balance between the speed of innovation implementation and the long-term stability of software systems.

Research methodology is based on the application of system and comparative analysis of scientific sources, examination of companies’ practical experience in software engineering, as

[1] Software Engineer, DispatchHealth Management, LLC, 3827 North Lafayette Street, Denver, CO 80205, USA, roman.abakumov@gmail.com, https://orcid.org/0009-0004-2743-3815

well as the use of methods for modeling architectural solutions, generalization of DevOps approaches, and quality management principles. The study relies on tools that have proven effective in a wide range of practices: static code analysis, contract testing, centralized documentation of architectural decisions, continuous performance monitoring, and periodic architecture audits.

Research results have shown that technical debt has a systemic nature, formed under the influence of both technological factors (heterogeneity of technology stacks, outdated dependencies, limited control tools) and organizational factors (distributed teams, lack of unified standards, staff turnover). It has been revealed that the implementation of universal practices – API standardization, integration of checks into CI/CD processes, maintenance of architectural decision bases, and planned refactoring – ensures debt controllability and mitigates its negative impact.

Conclusions demonstrate that addressing technical debt cannot be limited to local solutions. The introduction of universal approaches that combine standardization, monitoring, contract testing, and planned dependency updates is required. Such integration reduces maintenance costs, increases release stability, and ensures predictability in the development of software systems.

Prospects for further research lie in the development of integrated indicators for measuring technical debt, the use of intelligent algorithms to predict the risks of its accumulation, and the creation of adaptive management models capable of functioning in hybrid environments that combine microservices, cloud solutions, and legacy components.

Keywords: DevOps practices, architectural governance, contract testing, API standardization, dependency management, performance monitoring, code refactoring, CI/CD processes.

Вступ

Проблема технічного боргу в середовищі мікросервісної архітектури є однією з найбільш актуальних у сучасній інженерії програмного забезпечення, оскільки безпосередньо впливає на швидкість і якість розробки, стабільність роботи систем та економічну ефективність їхнього супроводу. На відміну від монолітних систем мікросервіси характеризуються високим рівнем децентралізації та багатомовністю технологічних стеків, що, з одного боку, забезпечує гнучкість та масштабованість, а з іншого – створює умови для накопичення структурних та організаційних недоліків, які формують технічний борг. Останній проявляється у вигляді надмірної складності інтеграції сервісів, дублювання функціоналу, зростання витрат на тестування й оновлення, а також зниження передбачуваності релізів.

З наукового погляду, дослідження механізмів зменшення технічного боргу в мікросервісних архітектурах пов’язане з пошуком моделей управління складними децентралізованими системами, інтеграцією підходів безперервної перевірки архітектурних рішень, застосуванням формальних методів опису залежностей і розробкою метрик, що дозволяють кількісно оцінити рівень боргу. Ці завдання перетинаються з проблематикою кібербезпеки, оскільки накопичення боргу часто призводить до вразливостей, а також із галуззю штучного інтелекту, яка пропонує алгоритмічні підходи до автоматизованого виявлення архітектурних аномалій.

Практична площина проблеми стосується здатності підприємств забезпечувати стабільний розвиток цифрових продуктів в умовах високої конкуренції та постійного оновлення технологій. Інноваційні методи керування технічним боргом у мікросервісних середовищах прямо пов’язані з ефективністю DevOps-практик, здатністю компаній мінімізувати ризики зупинок бізнес-процесів, швидко реагувати на зміну вимог ринку та підтримувати довгострокову якість програмних рішень. Отже, постановка проблеми поєднує фундаментальний вимір – розробку нових підходів до

моделювання складних систем – із прикладним, що стосується формування стійких бізнес-моделей у сфері цифрової економіки.

Аналіз сучасних досліджень із проблематики інноваційних підходів до зменшення технічного боргу в середовищі мікросервісної архітектури дозволяє виокремити чотири взаємопов'язані напрями. Перший напрям стосується наукових розвідок, пов'язаних з емпіричним вивченням природи технічного боргу та його проявів у мікросервісних системах. Зокрема, Р. Су (R. Su) показує, що технічний борг у хмарно орієнтованих застосунках проявляється як на рівні програмного коду, так і в загальній архітектурі, впливаючи на якість і підтримуваність систем [1]. Колектив авторів на чолі з В. Ленардуцці (V. Lenarduzzi et al.) з'ясовує, що перехід від монолітної архітектури до мікросервісів не завжди зменшує технічний борг, оскільки виникають нові виклики, пов'язані з інтеграцією та оркестрацією [2]. Науковці К. Борова (K. Borowa), А. Ратковскі (A. Ratkowski), Р. Вердеккія (R. Verdecchia) на основі кейсу великої промислової мікросервісної архітектури доводять, що технічний борг часто має прихований характер, а його «вартість» зростає у процесі масштабування системи [3].

Другий напрям охоплює наукові напрацювання, у яких розглядається систематизація досліджень і здійснюється пошук методів виявлення боргу. Наявні підходи до аналізу боргу картографують А. Вілла та співавтори (A. Villa et al.), вказуючи на недостатність єдиної методології для мікросервісного контексту [4]. Кількісні та якісні дані за допомогою змішано-методового аналізу поєднують Р. Вердеккья та колеги (R. Verdecchia et al.). Це дозволяє комплексно оцінювати вплив боргу на розвиток системи [5]. Науковець Й. Богнер та співавтори (J. Bogner et al.), досліджуючи практики підтримуваності, показують, що індустрія застосовує широкий спектр технік контролю боргу, проте їхня ефективність значно залежить від контексту сервісної архітектури [6].

До третього напряму належать роботи, зосереджені на стратегіях зменшення технічного боргу та його впливі на еволюційність систем. Колектив учених на чолі з С. С. де Толедо (S. S. de Toledo et al.) демонструє, що погашення архітектурного боргу сприяє істотному зниженню кількості інцидентів у мікросервісних системах [7]. Дослідник Й. Богнер та співавтори (J. Bogner et al.) акцентують на тому, що забезпечення еволюційності мікросервісів потребує балансування між швидкістю впровадження змін і контролем боргу [8]. Науковці В. Ленардуцці (V. Lenarduzzi) та Д. Таїбі (D. Taibi) доводять, що процентна ставка технічного боргу в мікросервісах може стрімко зростати без належної архітектурної дисципліни [9]. Учений С. С. Де Толедо та колеги (S. S. De Toledo et al.) стверджують, що в процесі міграції компаній до мікросервісів борг накопичується й потребує пріоритизації з огляду на критичність сервісів [10]..

Четвертий напрям охоплює дослідження, які стосуються соціотехнічні та організаційні виміри технічного боргу. Науковиця Н. Саарімякі та співавтори (N. Saarimäki et al.) підкреслюють, що успіх у зниженні боргу при міграції залежить не лише від архітектурних рішень, але й від узгодженості командних практик [11]. Учений М. Васім (M. Waseem et al.) з колегами доводять, що значна частина проблем у мікросервісних системах має організаційні корені, зокрема недостатнє управління та комунікацію [12]. Дослідник Р. Капілла (R. Capilla et al.) зі співавторами з'ясовують, що виявлення архітектурного боргу в проєктах із відкритим кодом вимагає врахування організаційних і координаційних чинників [13]. Колектив науковців на чолі з Н. Ніколаїдісом (N. Nikolaidis et al.) демонструє, що в сервісно орієнтованих системах технічний борг часто набуває форми інтеграційних перешкод [14]. Дослідники Т. Рінта-Кахіла (T. Rinta-Kahila), Е. Пенттінен (E. Penttinen) та К. Лийтинен (K. Lyytinen) аналізують соціотехнічні чинники та пропонують поняття «пастки технічного боргу», що ускладнює модернізацію застарілих систем [15].

Попри наявність численних досліджень, низка аспектів проблеми технічного боргу в мікросервісних архітектурах потребує більш ґрунтовного вивчення. Зокрема,

недостатньо системних підходів до визначення сутності боргу в умовах багатомовності технологічних стеків, обмежено розроблено універсальні метрики для його кількісного вимірювання, а практики автоматизованого моніторингу та контрактного тестування демонструють неоднакову ефективність у різних середовищах. Додатковими викликами є децентралізоване управління, організаційна фрагментація команд та висока плинність кадрів, що ускладнюють впровадження єдиних стратегій і зумовлюють накопичення прихованого боргу.

Запропоноване дослідження спрямоване на подолання цих прогалин шляхом поєднання теоретичного аналізу й узагальнення даних. Використання системного та порівняльного аналізу, а також моделювання архітектурних рішень дає змогу розширити уявлення про природу технічного боргу та оцінити вплив організаційних і технологічних факторів. Удосконалення методів моніторингу, стандартизації прикладних програмних інтерфейсів (Application Programming Interface, далі – API), інтеграції CI/CD та формування практичних рекомендацій забезпечить розробку комплексної стратегії управління, здатної знизити ризики та підвищити стійкість цифрових продуктів.

Мета статті – обґрунтувати теоретичні засади та практичні підходи до зменшення технічного боргу в мікросервісних архітектурах шляхом інтеграції інноваційних методів управління, що забезпечують баланс між швидкістю розробки, якістю програмного продукту та довгостроковою стабільністю системи.

Завдання статті:

1) розкрити сутність технічного боргу в мікросервісній архітектурі та узагальнити сучасні наукові підходи й інструменти його контролю;

2) дослідити організаційні та технологічні чинники накопичення технічного боргу й окреслити обмеження механізмів його зменшення;

3) розробити рекомендації щодо комплексної стратегії управління технічним боргом для забезпечення стійкості та конкурентоспроможності цифрових продуктів.

Матеріали та методи

Матеріали та методи дослідження базуються на системному й порівняльному аналізі технічного боргу в мікросервісній архітектурі, класифікації його проявів (нестандартизовані API, застарілі залежності, надмірна кількість сервісів, відсутність правил, низька якість документації) та оцінці впливу на інтеграцію, стабільність релізів і безпеку на основі аналізу сучасних підходів і практик управління боргом, зокрема використання метрик коду й архітектури, автоматизованих інструментів моніторингу й CI/CD (SonarQube, ArchUnit, Pact, SBOM, APM, CodeScene) та врахування організаційних і технологічних факторів.

Результати

Технічний борг у мікросервісній архітектурі розглядається як сукупність відкладених рішень, компромісів або недоліків у проєктуванні й реалізації програмних систем, що виникають під час пріоритетизації швидкості впровадження функціоналу над довгостроковою якістю й стійкістю. На відміну від монолітних систем у мікросервісах його накопичення має складніший характер через велику кількість незалежних сервісів, різноманітність мов програмування та інфраструктурних рішень. Це призводить до посилення проблем інтеграції, підвищення витрат на підтримку, ускладнення масштабування та зростання ймовірності відмов. Водночас технічний борг не завжди є негативним явищем: у певних випадках він дозволяє компаніям швидко виводити продукт на ринок, однак потребує подальшого планового «погашення» через рефакторинг або архітектурну перебудову (табл. 1).

Таблиця 1

Основні прояви технічного боргу в мікросервісній архітектурі та їхні наслідки

Прояв технічного боргу	Характеристика	Наслідки для життєвого циклу програмного забезпечення	Приклади в практиці
Нестандартизовані API	Різні підходи до побудови інтерфейсів між сервісами	Ускладнене тестування та інтеграція	Вимушене дублювання бізнес-логіки
Застарілі залежності	Використання бібліотек без оновлень	Підвищені ризики вразливостей і несумісності	Відсутність патчів безпеки
Надмірна кількість сервісів	Дрібна декомпозиція без контролю	Зростання складності моніторингу та управління	Збільшення кількості затримок у мережі
Відсутність централізованих правил	Неврегульовані стандарти коду та інфраструктури	Зростання витрат на підтримку й навчання нових команд	Конфлікти при (Continuous Integration / Continuous DeliveryCI/CD)
Низька якість документації	Обмежена або несистемна архітектурна документація	Втрата знань при зміні складу команди	Складнощі при онбордингу

Джерело: сформовано автором на основі [1, p. 66–67; 2; 3; 6, p. 125–126; 7, p. 196–197; 9; 10, p. 37422–37423]

У сучасній практиці управління мікросервісними системами технічний борг проявляється не лише як набір локальних проблем, а як системне явище, що впливає на всю архітектурну екосистему. Наприклад, відсутність стандартизованих API у великих фінтех-компаніях призводить до створення численних «шлюзів-перекладачів», які тимчасово розв'язують проблему інтеграції, але з часом накопичують каскадну складність: кожен новий сервіс вимагає окремих адаптацій, що збільшує час виходу оновлень на ринок [7, p. 197]. Використання застарілих залежностей особливо критично проявляється у сфері e-commerce, де нерідко відкладене оновлення бібліотек безпеки призводить до витоків даних клієнтів і необхідності термінових та витратних міграцій.

Надмірна кількість дрібних сервісів добре ілюструється в практиці великих логістичних платформ: прагнучи максимально декомпозувати функціонал, команди створюють десятки дрібних сервісів для окремих операцій (наприклад, підтвердження замовлення чи розрахунку доставлення), що в перспективі ускладнює їхній моніторинг і підвищує ймовірність технічних проблем у пікові періоди, як-от «чорна п'ятниця». Відсутність централізованих правил, відповідно, є відчутною проблемою в організаціях, де паралельно працює кілька команд: різні підходи до CI/CD, стилістики коду чи обробки помилок призводять до того, що обслуговування є дорогим і непередбачуваним [2].

Особливо показовим є приклад із неякісною або неповною документацією. Зокрема, у стартапах із високим темпом розробки документація часто відкладається «на потім», але в разі зміни складу команди це призводить до втрати архітектурних знань.

Нові розробники змушені витрачати тижні на розуміння логіки системи, що затримує випуск продукту й створює додаткові витрати. Таким чином, наведені в таблиці 1 прояви технічного боргу, в реальній практиці не є ізольованими інцидентами, а формують комплексні ризики для бізнесу – від збільшення часу на розробку й тестування до фінансових втрат через простої або компрометацію безпеки. З огляду на це, науковий інтерес до цієї проблеми поєднується з високою практичною значущістю, оскільки розробка ефективних стратегій управління технічним боргом прямо визначає конкурентоспроможність цифрових компаній у глобальному середовищі.

Контроль технічного боргу варто розуміти як багаторівневий процес, що поєднує методологічні підходи інженерії програмного забезпечення з практичними інструментами автоматизації. Технічний борг розглядається не лише як накопичення дефектів у коді, а як системний ризик, що охоплює архітектурні рішення, залежності, API та операційні характеристики системи. Для його вимірювання пропонуються метрики на рівні коду й архітектури, показники надійності сервісів та інтегровані індекси боргу, що відображають вплив на бізнес-процеси. У практичній площині ефективність забезпечується шляхом безперервного моніторингу та автоматизованого оцінювання, які інтегруються в цикли CI/CD. Це дозволяє перетворити управління боргом на регулярний процес, що підтримує стабільність цифрових продуктів (табл. 2).

Таблиця 2

Сучасні підходи та інструменти контролю технічного боргу в мікросервісах

Підхід / інструмент	**Що вимірює та як інтерпретує борг**	**Автоматизація та інтеграція в CI/CD**	**Типові метрики та індикатори**
Статичний аналіз якості коду (SonarQube)	«Запахи» коду, складність, дублювання, тестове покриття; відображає «основну суму» боргу	Автоматичні перевірки в pull-request; блокування злиття змін	Maintainability Index, Code Smells/KLOC, Coverage, Debt Ratio
Архітектурні фітнес-функції (ArchUnit)	Порушення архітектурних правил, некоректні залежності, відхилення від шарування	Запуск у пайплайнах CI; зупинка релізу при критичних порушеннях	Кількість порушень правил, стабільність модулів
Контрактне тестування (Pact, Schema Registry)	Несумісності API та схем даних; ризик зламів між сервісами	Автотести контрактів, перевірка зворотної сумісності	% сумісних релізів, кількість порушень контрактів
Аналіз залежностей і SBOM (Software Bill of Materials)	Використання застарілих бібліотек, вразливості, ліцензійні ризики	Сканування в CI; автооновлення залежностей	CVE-score, % застарілих залежностей, час оновлення
Теле метрія виконання (APM, Service Mesh)	Операційні «відсотки» боргу: латентність, помилки, ретраї	Алерти за SLO (Service Level Objectives); інтеграція в моніторинг	LAT P95, error rate, бюджет помилок

Підхід / інструмент	Що вимірює та як інтерпретує борг	Автоматизація та інтеграція в CI/CD	Типові метрики та індикатори
Репо-майнінг (CodeScene)	«Хотспоти» змін: високочастотні коміти, когнітивна складність, ризики володіння	Періодичні скани, дешборди для пріоритизації рефакторингу	Change Coupling, Bus Factor, Hotspot Score

Джерело: сформовано автором на основі [4, p. 182–183; 5, p. 204–205; 6, p. 125–126; 8, p. 546–547; 12, p. 201–202; 13, p. 394–395]

Контроль технічного боргу в мікросервісних архітектурах є прикладом того, як науково обґрунтовані методи стають основою для практичного управління складними цифровими екосистемами. Якщо в традиційних монолітних системах борг проявляється у вигляді повільних релізів чи надлишкової складності коду, то в мікросервісах він перетворюється на комплексну проблему, що охоплює десятки незалежних сервісів, різні технологічні стеки та розподілені команди розробників [3]. З огляду на це, застосування автоматизованих інструментів на кшталт статичного аналізу чи архітектурних фітнес-функцій набуває не лише технічного, а й стратегічного значення, адже вони запобігають внесенню дефектів на ранніх етапах, зменшують ризики технічних порушень і гарантують сумісність API.

У практиці глобальних компаній ці інструменти використовуються як запобіжники фінансових і репутаційних втрат. Наприклад, у фінтех-секторі навіть незначна несумісність між сервісами під час оновлення CI/CD може призвести до блокування транзакцій, що вартує мільйонів доларів. Контрактне тестування дозволяє уникати таких ситуацій, оскільки воно гарантує відповідність форматів обміну даними між сервісами. Аналіз залежностей через формування SBOM дає змогу швидко виявляти застарілі бібліотеки та закривати вразливості, що особливо критично для банківських платформ чи державних реєстрів [6, p. 126]. У сфері електронної комерції репо-майнінг допомагає виявляти «гарячі точки» коду, які постійно змінюються, та є джерелом помилок у пікові сезони розпродажів. У логістичних компаніях використання сервіс-мешів із телеметрією дозволяє вчасно побачити накопичення «відсотків» технічного боргу у вигляді зростання латентності чи частоти відмов. Описані методи не варто сприймати як набір розрізнених технік. Вони формують інтегровану систему управління технічним боргом, що поєднує науково обґрунтовані метрики з реальними бізнес-ефектами. Їхня цінність полягає в можливості перетворити борг із невидимого «тягаря» на вимірюваний показник, інтегрований у стратегічні рішення компанії. Це робить управління мікросервісами не лише ефективним у короткостроковій перспективі, але й життєздатним у довгостроковій, сприяючи підвищенню надійності цифрових продуктів і збереженню конкурентних позицій на глобальному ринку.

У процесі масштабування мікросервісних систем проблема технічного боргу набуває особливої актуальності, оскільки поєднання організаційних і технологічних чинників створює сприятливе середовище для його накопичення. З одного боку, децентралізована природа мікросервісної архітектури передбачає автономність команд та гнучкість у виборі інструментів, що забезпечує швидкість впровадження нових функцій. З іншого – відсутність єдиних стандартів, різнорідність технологічних стеків та обмеженість ресурсів для уніфікації процесів призводять до поступового зростання боргу. Додатковим викликом є організаційна фрагментованість: розподілені команди ухвалюють локальні рішення, які в короткостроковій перспективі оптимізують розробку, але в довгостроковій – створюють додаткові витрати на підтримку та інтеграцію (табл. 3).

Таблиця 3

Організаційні та технологічні фактори накопичення технічного боргу в масштабованих мікросервісних системах

Категорія факторів	Приклад прояву	Потенційні наслідки
Організаційні	Відсутність єдиних стандартів документування й API	Несумісність між сервісами, складність навчання нових розробників
	Розподілені команди з різними підходами до CI/CD (Continuous Integration / Continuous Delivery)	Конфлікти при інтеграції, затримки релізів
	Орієнтація на швидкість постачання продукту без урахування довгострокової підтримки	Зростання «невидимого» боргу, ускладнене рефакторингом у майбутньому
Технологічні	Різнорідність стеків (різні мови програмування, фреймворки, бази даних)	Висока вартість інтеграції, збільшення потреби в спеціалізованих фахівцях
	Залежність від застарілих бібліотек та інструментів	Підвищені ризики безпеки, витрати на термінові оновлення
	Відсутність централізованого моніторингу та уніфікованих метрик	Неможливість вчасного виявлення критичних відхилень

Джерело: сформовано автором на основі [11; 12, p. 201–202; 13, p. 394–395; 14, p. 270–271; 15, p. 1–2]

У сучасних умовах ці фактори тісно взаємодіють, формуючи замкнене коло. Наприклад, у фінансових організаціях, де команди працюють у різних часових зонах, часто відсутня централізована координація архітектурних рішень. Це призводить до того, що сервіси розробляються незалежно, без єдиних правил для API, що з часом створює труднощі інтеграції [1, p. 68–69; 11]. У стартапах плинність кадрів є важливим організаційним фактором: коли команда змінюється, документація часто виявляється неповною або застарілою, а нові розробники змушені витрачати час на «зворотну інженерію» системи. Технологічні фактори особливо виразно проявляються в масштабних логістичних і e-commerce платформах: зростання кількості сервісів неминуче збільшує кількість мережевих залежностей, що призводить до додаткових затримок і нестабільності під час пікових навантажень, як-от у період масових акцій чи святкових розпродажів [7, p. 196–197]. Гетерогенність технологічних стеків також посилює проблему, адже відсутність єдиних процесів CI/CD ускладнює синхронізацію релізів, що часто призводить до конфліктів у пайплайнах [8, p. 546–547]. Відсутність узгоджених стандартів документації поглиблює всі зазначені проблеми, оскільки робить систему менш прозорою для нових команд і збільшує витрати на підтримку. Так, організаційні та технологічні чинники накопичення технічного боргу в умовах масштабування мікросервісних систем не є ізольованими, а утворюють складну мережу взаємопов'язаних ризиків, що потребують системного управління та стратегічної інтеграції інструментів контролю.

Упровадження ефективних механізмів зменшення технічного боргу в умовах мікросервісної архітектури значно ускладнюється низкою системних обмежень, що посилюються багатомовністю технологічних стеків та децентралізованим управлінням. Поєднання різних мов програмування та фреймворків створює труднощі уніфікації

стандартів коду, інструментів статичного аналізу та засобів тестування [4, p. 182–183; 5, p. 204–205]. Неможливість побудови єдиного середовища контролю якості призводить до фрагментації підходів і знижує ефективність метрик, які застосовуються для вимірювання технічного боргу [6, p. 125–126; 9]. Додатково зростає складність інтеграції механізмів автоматизованого моніторингу та CI/CD-процесів, адже окремі інструменти можуть підтримувати лише частину використовуваних мов і платформ, що спричиняє розбіжності в наскрізному контролі.

Децентралізована модель управління підсилює цю проблему, оскільки відсутність єдиного центру архітектурних рішень породжує суперечності між командами. У таких умовах складно нав'язати уніфіковані правила для API чи узгодити політику оновлення залежностей, і навіть за наявності локальних практик зменшення боргу вони втрачають ефективність на рівні всієї системи [11; 14, p. 270–271]. Кожна команда орієнтується на власні пріоритети, що може призводити до конфліктів між вимогами швидкого релізу та необхідністю проведення рефакторингу. Висока плинність кадрів та різний рівень зрілості окремих груп розробників підсилюють накопичення прихованого боргу, адже нові члени команд часто не мають доступу до повної історії архітектурних рішень [15, p. 1–2].

Важливим обмеженням є також висока вартість інтеграції механізмів зменшення боргу в уже наявні системи. Великі платформи з різнорідною інфраструктурою вимагають значних ресурсів для впровадження централізованих інструментів спостережуваності, автоматичного оновлення залежностей чи застосування архітектурних фітнес-функцій [5, p. 204–205; 7, p. 196–197]. Часто такі ініціативи відсуваються на другий план порівняно з бізнес-завданнями, що зумовлює накопичення довгострокових ризиків. Додаткову складність становить відсутність достатньої кількості кваліфікованих фахівців, здатних підтримувати багатомовні екосистеми та координувати взаємодію незалежних команд. Це призводить до того, що навіть формально впроваджені механізми управління боргом працюють лише частково, а значна частина проблем залишається невидимою до моменту критичного збою.

Управління технічним боргом у мікросервісних архітектурах потребує системного й багаторівневого підходу, оскільки поодинокі практики на кшталт локального рефакторингу чи оновлення залежностей не забезпечують довгострокової стійкості цифрового продукту. Комплексна стратегія повинна інтегрувати технічні, організаційні та бізнес-орієнтовані інструменти, що дозволяють одночасно контролювати виникнення боргу, забезпечувати його прозоре вимірювання та своєчасне зменшення. Це критично важливо в умовах швидкої цифрової конкуренції, коли якість архітектури безпосередньо впливає на здатність компанії масштабуватися, підтримувати інноваційність і гарантувати безпеку (рис. 1).

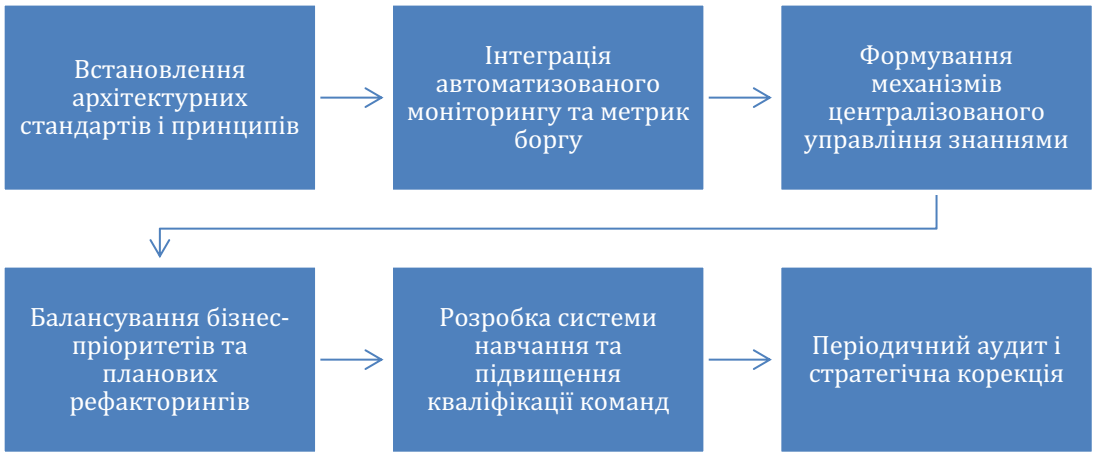

Рис. 1. Покрокова стратегія управління технічним боргом у мікросервісах

Джерело: власна розробка автора

Запропонована стратегія функціонує як безперервний цикл. На першому етапі визначаються архітектурні принципи та стандарти, які задають «рамки» для команд і зменшують ризики хаотичного накопичення боргу. Наступним кроком є впровадження автоматизованих засобів моніторингу – від статичного аналізу коду до метрик продуктивності та надійності, що дозволяють зробити борг вимірюваним і прозорим для управління. Паралельно формується централізована база знань у вигляді архітектурних рішень, документації та ADR (Architecture Decision Records), що забезпечує спадкоємність між командами та знижує ризик «невидимого» боргу.

Четвертий етап передбачає узгодження бізнес-пріоритетів із технічними завданнями: у спринти планово закладаються активності з рефакторингу, що дозволяє уникати критичних відставань і технічних перешкод у довгостроковій перспективі. Наступним кроком є розвиток компетенцій команд через навчання й упровадження єдиних практик DevOps та CI/CD, адже саме різнорідність підходів часто блокує ефективність контролю боргу. Завершальним етапом є періодичний аудит, що дозволяє зіставляти фактичний рівень боргу з прийнятими показниками стійкості та за потреби коригувати стратегію.

У практичному вимірі реалізація цієї схеми дає змогу створити систему «керованого боргу», коли він не накопичується стихійно, а знаходиться під постійним контролем. Очікуваними результатами є скорочення витрат на підтримку, підвищення стабільності релізів, зменшення ризику критичних порушень і забезпечення довгострокової конкурентоспроможності цифрових продуктів. Такий підхід трансформує технічний борг із неконтрольованого ризику в прогнозовану величину, що може бути інтегрована в бізнес-моделі розвитку компанії.

Висновки

У дослідженні встановлено, що технічний борг у мікросервісних архітектурах є не лише наслідком локальних недоліків коду, а комплексним системним ризиком, який формується під впливом організаційних та технологічних чинників. Його основними проявами є нестандартизовані API, використання застарілих залежностей, надмірна фрагментація сервісів, конфлікти при CI/CD та слабка документація. Такі чинники безпосередньо впливають на життєвий цикл програмного забезпечення, ускладнюють інтеграцію й тестування, знижують передбачуваність релізів і створюють додаткові фінансові та безпекові ризики.

До основних проблем належать труднощі уніфікації багатомовних стеків, обмеженість автоматизованого моніторингу, фрагментація управлінських підходів у розподілених командах та висока вартість упровадження централізованих механізмів контролю боргу. Встановлено, що децентралізоване управління й орієнтація на швидкість релізів у короткостроковій перспективі поглиблюють накопичення прихованого боргу та ускладнюють його системне зменшення.

Рекомендовано застосовувати комплексну стратегію управління, що поєднує стандартизацію архітектурних рішень, автоматизований статичний аналіз, контрактне тестування, аналіз залежностей і телеметрію з формуванням централізованих баз знань та єдиних DevOps-процесів. Такий підхід дозволяє зробити технічний борг вимірюваним, контрольованим і прогнозованим.

Перспективи подальших досліджень пов’язані з розробкою інтегрованих індексів технічного боргу, використанням алгоритмів штучного інтелекту для прогнозування та виявлення архітектурних аномалій, а також із моделюванням адаптивних систем управління для гібридних цифрових середовищ. Це сприятиме підвищенню стійкості цифрових продуктів і збереженню конкурентних позицій компаній у глобальній економіці.

Список використаних джерел

1. Su R. Technical Debt and Software Quality in Cloud-Native Applications. *Software Architecture. ECSA 2024 Tracks and Workshops. ECSA 2024. Lecture Notes in Computer Science*. Vol 14937. Springer, Cham, 2024. P. 65–71. DOI: https://doi.org/10.1007/978-3-031-71246-3_8

2. Lenarduzzi V., Lomio F., Saarimäki N., Taibi D. Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software*. 2020. Vol. 169. Article 110710. DOI: https://doi.org/10.1016/j.jss.2020.110710

3. Borowa K., Ratkowski A., Verdecchia R. The Technical Debt Gamble: A Case Study on Technical Debt in a Large-Scale Industrial Microservice Architecture. 2025. *arXiv.* arXiv:2506.16214. DOI: https://doi.org/10.48550/arXiv.2506.16214

4. Villa A., Ocharán-Hernández J. O., Pérez-Arriaga J. C., Limón X. A Systematic Mapping Study on Technical Debt in Microservices. *2022 10th International Conference in Software Engineering Research and Innovation (CONISOFT)* (24-28 October 2022). Ciudad Modelo, San José Chiapa, Mexico, 2022. P. 182–191. DOI: https://doi.org/10.1109/CONISOFT55708.2022.00032

5. Verdecchia R., Maggi K., Scommegna L., Vicario E. Technical Debt in Microservices: A Mixed-Method Case Study. *Software Architecture. ECSA 2023 Tracks, Workshops, and Doctoral Symposium. Lecture Notes in Computer Science*. Vol. 14590. Cham: Springer, 2024. P. 204–220. DOI: https://doi.org/10.1007/978-3-031-66326-0_14

6. Bogner J., Fritzsch J., Wagner S., Zimmermann A. Limiting technical debt with maintainability assurance: an industry survey on used techniques and differences with service- and microservice-based systems. *Proceedings of the 2018 International Conference on Technical Debt* (Gothenburg, May 27–28, 2018). Gothenburg, Sweden, 2018. P. 125–133. DOI: https://doi.org/10.1145/3194164.3194166

7. de Toledo S. S., Martini A., Sjøberg D. I. K., Przybyszewska A., Frandsen J. S. Reducing Incidents in Microservices by Repaying Architectural Technical Debt. *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (Palermo, 01–03 September 2021). Palermo, Italy, 2021. P. 196–205. DOI: https://doi.org/10.1109/SEAA53835.2021.00033

8. Bogner J., Fritzsch J., Wagner S., Zimmermann A. Assuring the Evolvability of Microservices: Insights into Industry Practices and Challenges. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (Cleveland 29 September–04 October 2019*.)* Cleveland, OH, USA, 2019. P. 546–556. DOI: https://doi.org/10.1109/ICSME.2019.00089

9. Lenarduzzi V., Taibi D. Microservices, continuous architecture, and technical debt interest: An empirical study. 2018. *arXiv.* arXiv:1810.10855. DOI: https://doi.org/10.48550/arXiv.1810.10855

10. de Toledo S. S., Martini A., Nguyen P. H., Sjøberg D. I. K. Accumulation and Prioritization of Architectural Debt in Three Companies Migrating to Microservices. *IEEE Access*. 2022. Vol. 10. P. 37422–37445. DOI: https://doi.org/10.1109/ACCESS.2022.3158648

11. Saarimäki N., Lomio F., Lenarduzzi V., Taibi D. Does migrate a monolithic system to microservices decreases the technical debt? 2019. *arXiv/* arXiv:1902.06282. URL: https://research.tuni.fi/app/uploads/2019/03/ff7209ff-1902.06282.pdf (date of access: 27.07.2025).

12. Waseem M., Liang P., Shahin M., Ahmad A., Nassab A. R. On the nature of issues in five open source microservices systems: An empirical study. *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering* (Trondheim, June 21–23, 2021). Trondheim, Norway, 2021. P. 201–210. DOI: 10.1145/3463274.3463337

13. Capilla R., Fontana F. A., Mikkonen T., Bacchiega P., Salamanca V. Detecting Architecture Debt in Micro-Service Open-Source Projects. *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (Durres, 06–08 September 2023). Durres, Albania, 2023. P. 394–401. DOI: https://doi.org/10.1109/SEAA60479.2023.00066

14. Nikolaidis N., Ampatzoglou A., Chatzigeorgiou A., Tsekeridou S., Piperidis A. Technical Debt in Service-Oriented Software Systems. *Product-Focused Software Process Improvement. PROFES 2022. Lecture Notes in Computer Science*. Vol. 13709. Cham: Springer, 2022. P. 270–285. DOI: https://doi.org/10.1007/978-3-031-21388-5_19

15. Rinta-Kahila T., Penttinen E., Lyytinen K. Getting trapped in technical debt: Sociotechnical analysis of a legacy system's replacement. *MIS Quarterly*. 2023. Vol. 47. № 1. P. 1–32. DOI: https://doi.org/10.25300/MISQ/2022/16711