Державний вищий навчальний заклад

«Донецький національний технічний університет»

«Наукові досягнення та відкриття сучасної молоді»

V Всеукраїнська наукова конференція студентів та молодих вчених

Присвячена 105 річниці Державного вищого навчального закладу

«Донецький національний технічний університет»

Збірник матеріалів

12 травня 2026 року

м. Дрогобич

УДК 001-053.6
Н 34

Наукові досягнення та відкриття сучасної молоді [Електронний ресурс] : зб.матер. IV Всеукр. наук. конф. студ. та молодих вчених (Дрогобич, 12 трав. 2026 р.) / Держ. вищ. навч. заклад «Донецький національний технічний університет». – Дрогобич : ДВНЗ «ДонНТУ», 2026. – 172 с.

ISBN: 978-966-377-260-8

Збірник містить доповіді учасників V Всеукраїнської науково-практичної конференції «Наукові досягнення та відкриття сучасної молоді», які розподілені за секціями: «Комп'ютерні та технічні науки», «Соціо-гуманітарні науки, освіта, педагогіка, бізнес, адміністрування та право», «Цивільна та екологічна безпека».

Видання може бути корисним здобувачам вищої освіти, молодим науковцям та викладачам. Усі матеріали друкуються в авторській редакції і відповідальність за їх зміст несуть автори. Оргкомітет конференції претензії з цього приводу не приймає.

Відповідальна за випуск:
Микита АЛЕКСАНДРОВ – Голова Ради молодих вчених ДВНЗ «ДонНТУ», Ph.D, доцент кафедри штучного інтелекту і кібербезпеки.

Організаційний комітет

Попова Ольга Юріївна – д-р екон. наук, професор кафедри управління і фінансово-економічної безпеки, проректор з наукової роботи, **голова оргкомітету**.

Александров Микита Олександрович – Ph.D., доцент кафедри штучного інтелекту і кібербезпеки, **заступник голови оргкомітету.**

Дорогий Ярослав Юрійович – д-р тех. наук, професор, академік МКА, завідувач кафедри штучного інтелекту і кібербезпеки.

Кутняшенко Олексій Ігорович – к.т.н., доцент, доцент кафедри природоохоронної діяльності.

Ігнаткін Валерій Устинович – д.т.н. професор, професор кафедри автоматики і телекомунікацій.

Жуковська Дар'я Олександрівна – старший викладач кафедри автоматики та телекомунікацій.

Єжова Єлизавета Олексіївна – асистент кафедри штучного інтелекту і кібербезпеки.

Хома Дмитро Юрійович – асистент кафедри штучного інтелекту і кібербезпеки.

 © ДВНЗ «ДонНТУ», 2026

СЕМАНТИЧНИЙ ПІДХІД ДО ВІДНОВЛЕННЯ БІЗНЕС-ЛОГІКИ В ПОЛІГЛОТНИХ ENTERPRISE-СИСТЕМАХ

[1]Дорогий Ярослав Юрійович, д.т.н., проф., зав. каф. ШІК
[2]Понепаляк Андрій Андрійович, аспірант
ДВНЗ «Донецький національний технічний університет»
[2]andrii.ponepaliak.asp@donntu.edu.ua

У процесі супроводу великих корпоративних програмних систем однією з найскладніших задач залишається відновлення бізнес-логіки. Особливо гостро ця проблема проявляється в Enterprise-системах із тривалим життєвим циклом, адже такі системи майже ніколи не залишаються незмінними: вони поступово адаптуються до нових вимог, проходять через технологічні оновлення, інтегруються з іншими сервісами, а інколи частково перебудовуються на архітектурному рівні. Така динаміка узгоджується із законами еволюції програмного забезпечення М. Лемана та Л. Беладі, відповідно до яких програмні системи, що функціонують у реальному середовищі, змушені постійно змінюватися, тоді як їхня складність без систематичного контролю має тенденцію до зростання [1].

Однак сама тривалість існування системи ще не пояснює всієї складності її подальшого супроводу. Важливішим є інше. Програмне забезпечення має властивість «невидимості», на якій наголошував Ф. Брукс: структура програмної системи не має єдиного фізичного представлення, а будь-яка схема, діаграма або модель відображає лише одну з можливих проєкцій цієї системи [2]. Саме тому навіть повний доступ до вихідного коду, конфігурацій, документації та інтеграційних описів не гарантує автоматичного отримання цілісної картини її бізнес-поведінки. Розробник у такій ситуації змушений відновлювати не лише послідовність викликів, а й контекст їхнього виконання, приховані залежності, історію технічних рішень і зв'язок між програмною реалізацією та бізнес-вимогами.

У таких умовах виникає феномен деградації бізнес-знань, або Business Knowledge Decay. Йдеться не про втрату коду як такого. Код може залишатися доступним, компілюватися й навіть коректно працювати, але поступово втрачається знання про те, чому певне архітектурне або програмне рішення було реалізоване саме таким способом. Вихідний код фіксує кінцевий результат, однак часто не зберігає причинно-наслідкового ланцюга прийняття рішень. Це безпосередньо пов'язано з проблематикою Program Comprehension, оскільки розуміння програми передбачає не лише читання синтаксису, а й побудову ментальної моделі, яка поєднує низькорівневі елементи коду з високорівневими доменними поняттями [3].

Деградацію бізнес-знань доцільно розглядати як багаторівневий процес. Перший рівень пов'язаний із втратою носіїв знань: розробники, архітектори або бізнес-аналітики, які брали участь у створенні системи, залишають проєкт, а їхні ментальні моделі зазвичай не передаються у формалізованому вигляді. Другий рівень полягає у втраті бізнес-контексту. Початкові вимоги змінюються, але

окремі фрагменти старої логіки продовжують існувати в системі як історичні артефакти. Третій рівень становить технологічна ентропія, яка формується через накопичення тимчасових рішень, часткових рефакторингів, неузгоджених змін і технічного боргу. Для розподілених і мікросервісних систем ця проблема є особливо відчутною, оскільки навіть локальні на перший погляд зміни можуть мати непрямі наслідки для інших частин архітектури [4].

Окремим чинником втрати прозорості є використання архітектурних підходів, які свідомо зменшують жорстке зв'язування компонентів. Зокрема, Inversion of Control та Dependency Injection підвищують модульність, спрощують тестування й дають змогу замінювати реалізації без прямої модифікації клієнтського коду [5]. Це важлива перевага. Проте вона має і зворотний бік, оскільки разом із гнучкістю зростає складність відновлення фактичного графа виконання. У традиційному імперативному коді виклик одного компонента іншим зазвичай можна простежити безпосередньо, тоді як у системах із DI-контейнерами реальна реалізація інтерфейсу може визначатися через конфігурації, анотації або правила контейнера вже під час виконання.

Унаслідок цього частина архітектурних зв'язків набуває латентного характеру. Вони існують у фактичній поведінці системи, але не представлені явно у вихідному коді. Для розробника це означає, що синтаксичне читання програми вже не дає повної відповіді на питання, які компоненти реально взаємодіють між собою і які бізнес-правила залежать від цієї взаємодії. Кожна зміна потребує не лише аналізу коду. Потрібно ще реконструювати умови його фактичного виконання, що істотно посилює когнітивне навантаження під час супроводу.

Поліглотність сучасних Enterprise-систем ще більше ускладнює відновлення бізнес-логіки. У цьому контексті поліглотність означає не лише використання кількох мов програмування, наприклад Java, C#, TypeScript або Python, а й співіснування різних способів опису логіки: імперативного коду, декларативних конфігурацій, ORM-мапінгу, анотацій, SQL-запитів, YAML-, XML- або JSON-файлів. Така ситуація суперечить ідеальному уявленню про узгодженість доменної моделі та програмної реалізації, яке лежить в основі Domain-Driven Design [6]. На практиці бізнес-правило рідко залишається локалізованим у межах одного класу чи методу. Воно може бути розподілене між кількома шарами, сервісами, конфігураційними артефактами та зовнішніми залежностями.

Наприклад, правило нарахування знижки для VIP-клієнта може містити перевірку статусу клієнта в імперативному коді, значення відсотка знижки в YAML-конфігурації, а логіку доступу до даних – у репозиторії або ORM-мапінгу. Окремо кожен із цих фрагментів є відносно зрозумілим. Проте жоден із них сам по собі не репрезентує повного бізнес-правила. У результаті бізнес-логіка стає розсіяною, а її цілісне відновлення потребує інтеграції гетерогенних джерел.

Класичні методи Architecture Recovery та Reverse Engineering здебільшого працюють із синтаксичним і структурним рівнями системи. Вони дають змогу будувати абстрактні синтаксичні дерева, графи потоку керування, графи залежностей і графи викликів. Такі інструменти справді корисні для виявлення

структурних зв'язків, модульних залежностей, дублювання або порушень архітектурних обмежень. Проте вони не дають прямої відповіді на інше, більш змістове питання: яку саме бізнес-операцію реалізує певний фрагмент коду. Методи реконструкції архітектури формалізують процес відновлення структурних абстракцій, однак їхня ефективність обмежується тоді, коли ключові зв'язки існують не лише в синтаксисі коду, а й у конфігураціях, метаданих або runtime-механізмах [7].

Центральною проблемою стає семантичний розрив між предметною областю та простором програмної реалізації. На рівні бізнес-вимог система описується через поняття «перевірка кредитного ліміту», «верифікація користувача», «нарахування знижки», «обробка замовлення» або «застосування бізнес-обмеження». У програмному коді ті самі операції розпадаються на класи, методи, змінні, умови, виклики сервісів, конфігураційні параметри та правила ін'єкції залежностей. Через це синтаксична подібність не гарантує семантичної подібності. І навпаки: різна форма запису не обов'язково означає різний бізнес-сенс.

Отже, семантичний розрив можна визначити як когнітивну та репрезентативну відстань між словником предметної області й словником програмної реалізації. На одному рівні система мислиться через бізнес-поняття: клієнт, замовлення, знижка, ліміт, право доступу, статус користувача. На іншому рівні вона існує як сукупність класів, методів, інтерфейсів, сервісів, репозиторіїв, конфігураційних ключів і параметрів середовища. Проблема полягає в тому, що між цими рівнями немає автоматичної відповідності. Саме тому аналіз лише структурного рівня є недостатнім для повноцінного відновлення бізнес-логіки.

У цьому контексті перспективним напрямом є формалізація зв'язку між структурними елементами програмної системи та доменними поняттями, які вони реалізують. Йдеться не про заміну класичних методів статичного аналізу. Радше йдеться про їх доповнення семантично орієнтованими моделями, здатними враховувати не тільки сам факт виклику методу або залежності між класами, а й функціональний зміст відповідних фрагментів системи. Особливого значення це набуває для поліглотних Enterprise-систем, де бізнес-сенс формується не в одному файлі, а внаслідок взаємодії коду, конфігурацій, метаданих і зовнішніх інтеграцій.

Таким чином, відновлення бізнес-логіки в корпоративних програмних системах потребує переходу від суто синтаксичного аналізу до семантично орієнтованого подання. Деградація бізнес-знань, розсіювання логіки, runtime-залежності, IoC, Dependency Injection та поліглотність створюють ситуацію, у якій класичні AST-, CFG- та call graph-підходи залишаються необхідними, але вже недостатніми [8]. Подальші дослідження доцільно спрямовувати на побудову формальних моделей, здатних поєднувати структурні характеристики програмних артефактів із доменними поняттями, які ці артефакти реалізують. Це є необхідною передумовою зниження когнітивного навантаження під час супроводу складних програмних систем і підвищення надійності змін у бізнес-критичних компонентах.

ЛІТЕРАТУРА

1. Lehman M. M., Belady L. A. Program evolution: processes of software change. London : Academic Press, 1985. 538 p.
2. Brooks F. P. The mythical man-month: essays on software engineering. Anniversary ed. Boston : Addison-Wesley Professional, 1995. 322 p.
3. von Mayrhauser A., Vans A. M. Program comprehension during software maintenance and evolution. *Computer*. 1995. Vol. 28, No. 8. P. 44–55. DOI: 10.1109/2.402076.
4. Accumulation and prioritization of architectural debt in three companies migrating to microservices / S. S. de Toledo et al. *IEEE Access*. 2022. Vol. 10. P. 37422–37445. DOI: 10.1109/ACCESS.2022.3158648.
5. Seemann M., van Deursen S. Dependency injection principles, practices, and patterns. Shelter Island : Manning Publications, 2019. 528 p.
6. Evans E. Domain-driven design: tackling complexity in the heart of software. Boston : Addison-Wesley, 2003. 560 p.
7. Towards a process-oriented software architecture reconstruction taxonomy / S. Ducasse et al. *Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR 2007)*. New York : IEEE, 2007. P. 261–274. DOI: 10.1109/CSMR.2007.50.
8. Понепаляк А. А. Феномен деградації знань про бізнес-логіку в поліглотних корпоративних системах. *Наука і техніка сьогодні*. 2026. № 4(58). С. 4171–4188. DOI: 10.52058/2786-6025-2026-4(58)-4171-4188.