УДК 004.439:330.526.33 **DOI:** 10.31673/2412-9070.2023.061315

В. О. МИКОЛАЄНКО, аспірант;
К. П. СТОРЧАК, доктор техн. наук, професор,
Державний університет інформаційно-комунікаційних технологій, Київ

КОДОГЕНЕРАЦІЯ ДЛЯ ВЕЛИКИХ КОМЕРЦІЙНИХ ПРОЄКТІВ: ЯКИЙ ВИГЛЯД МАТИМЕ КОМЕРЦІЙНЕ РОЗРОБЛЕННЯ В МАЙБУТНЬОМУ

Кодогенерація — це процес автоматичного створення коду з вищих рівнів абстракції, який активно набирає обертів у сучасному програмуванні. Особливо це відчутно у великих комерційних проєктах, де важливо швидко адаптуватися до змінюваних вимог бізнесу, водночас забезпечуючи високу якість коду та знижуючи можливість появи помилок.

Ключові слова: кодогенерація; комерційне розроблення; автоматизація; OpenAPI; Swagger; ROSLYN; технологічні досягнення.

Вступ

У світі програмування, де все змінюється з неймовірною швидкістю, автоматизація стає головним фактором успіху. Кодогенерація, яка колись була просто цікавим експериментом, сьогодні стає інструментом для ефективного розроблення. Це процес автоматичного створення коду на основі визначених специфікацій, шаблонів або інших вихідних даних. Вона використовується для поліпшення продуктивності, забезпечення узгодженості коду та зменшення можливості появи помилок. До її переваг можна додати:

• ***швидкість розроблення:*** за допомогою інструментів кодогенерації розробники можуть швидко створити базові компоненти програми, зокрема класи, інтерфейси або базові модулі;

• ***узгодженість та стандарти:*** інструменти кодогенерації можуть забезпечити, щоб усі частини коду відповідали певним стандартам і найкращим практикам;

• ***змінення помилок:*** якщо код генерується автоматично, існує менший ризик появи помилок, пов'язаних із ручним уведенням.

Аналіз дослідження. Основні інструменти та платформи, що сьогодні використовують для кодогенерації, охоплюють OpenAPI, Swagger, ROSLYN від Microsoft тощо. У світі програмування, де технологічний прогрес не знає меж, автоматизація та кодогенерація визначають напрямки розвитку. Кодогенерація — це не просто автоматичне створення коду, а методологія, яка може кардинально змінити підхід до розроблення ПЗ. Розглянемо докладніше деякі з цих інструментів та методологій, які формують сучасний підхід до кодогенерації.

♦ *Сучасні інструменти*, зокрема OpenAPI, Swagger або GraphQL, уможливлюють автоматичне генерування клієнтського та серверного кодів для вебсервісів. Інші інструменти, такі як Yeoman чи ROSLYN, надають можливість створювати шаблони для різних типів проєктів.

♦ *Модельно-орієнтоване розроблення* (MDD): дає змогу розробникам створювати високорівневі моделі застосунків, які потім можна перетворити на вихідний код за допомогою спеціальних інструментів.

♦ *Code scaffolding:* цей підхід містить автоматичне створення базової структури проєкту, що значно спрощує старт нового проєкту.

♦ *Оптимізація та рефакторинг:* деякі інструменти кодогенерації можуть автоматично оптимізувати код, видаляючи непотрібні частини або рефакторячи його для підвищення продуктивності та читабельності.

♦ *Безпека:* інструменти кодогенерації здатні також допомогти автоматично виявляти та виправляти потенційні проблеми безпеки, що робить розроблення безпечнішим.

Використання кодогенерації може значно зменшити час розроблення, забезпечити високий стандарт коду та зменшити ризик появи помилок. Однак, як і з будь-яким інструментом, важливо правильно використовувати кодогенерацію, щоб не втратити гнучкість та можливість адаптації до змін.

Мета дослідження. Визначити ключові переваги та виклики кодогенерації в контексті великих комерційних проєктів та її потенційний вплив на майбутнє розроблень програмного забезпечення.

Основна частина

Як і будь-яка нова технологія, кодогенерація привносить із собою власні виклики, особливо в контексті великих комерційних проєктів. Далі розглянемо деякі з цих викликів та дізнаємося, як вони можуть вплинути на майбутні комерційні розроблення.

♦ *Складність інтеграції.* Великі проєкти здебільшого містять у собі різні модулі, які розробляються всіма командами. Це може ускладнити інтеграцію згенерованого коду.

♦ *Обслуговування та зміни.* Код, який було згенеровано автоматично, може бути важко змінити або налаштувати, особливо, якщо команда розробників непоінформована про механізм генерації.

♦ *Продуктивність.* Згенерований код може бути менш оптимізованим відповідно до коду, написаного програмістом вручну, що може призвести до проблемної продуктивності.

♦ *Безпека.* Автоматична генерація коду може мати вразливості, якими здатні скористатися зловмисники.

♦ *Залежність від інструментів.* Великі проєкти можуть залежати від конкретних інструментів для кодогенерації, стаючи проблемою, якщо ці інструменти перестануть підтримуватися.

♦ *Документація та навчання.* Згенерований код може бути важко розуміти без належної документації. Також може з'явитися потреба в навчальній команді розробників для роботи з інструментами кодогенерації.

♦ *Стандартизація.* Великі комерційні проєкти часто мають відповідати різним стандартам, ускладнюючи використання стандартних інструментів для кодогенерації.

♦ *Сумісність версій.* З тим, як проєкт розвивається, з'являтимуться нові версії інструментів кодогенерації. Це може призвести до конфліктів сумісності між різними частинами проєкту.

♦ *Відсутність контролю.* Коли код генерується автоматично, розробники можуть втратити деякий контроль над деталями реалізації, що може бути критичним для застосунків високого завантаження або тих, що мають строгі вимоги до безпеки.

♦ *Інтелектуальна власність.* Застосування сторонніх інструментів для кодогенерації може порушувати питання інтелектуальної власності, якщо згенерований код вважається власністю виробника інструменту.

♦ *Висока вартість міграції.* Якщо команда вирішує перейти від одного інструменту кодогенерації до іншого, процес міграції може бути складним та дорогим.

З огляду на стрімкий розвиток науки та техніки людство починає розуміти, що межі можливого постійно зміщуються. Те, що кілька десятиріч тому здавалося науковою фантастикою, сьогодні стає реальністю. Це стосується і світу програмування. Уже зараз ми бачимо, як штучний інтелект допомагає розробникам у їхніх щоденних завданнях. Але яке майбутнє нас чекає? Яким буде новий світ, де межі між людиною та машиною, між творцем та інструментом, що стають дедалі менш помітними?

Кодогенерація майбутнього може стати ключем до відповіді на ці питання. Цей процес може радикально змінити спосіб, яким ми підходимо до розроблення програмного забезпечення, перетворюючи ідеї на код миттєво та автоматично. Це новий світ, де креативність розробника поєднується з обчислювальною потужністю машин для створення застосунків, які здатні змінити світ. Можна припустити кілька прогнозів щодо кодогенерації в мабутньому.

Шаблонна кодогенерація. Інструменти, які дають змогу розробникам створювати шаблони для різних аспектів розроблення (наприклад, створення API, баз даних тощо), будуть дедалі популярнішими. Це дасть можливість командам швидко налаштовувати та адаптувати свої системи. Формулу для шаблонної кодогенерації можна подати в такому вигляді:

$$C = S \times (P + A + T),$$

де C — готовий код; S — базовий шаблон; P — параметри конфігурації; A — додаткові модулі/бібліотеки; T — технологічний стек.

Сучасне розроблення програмного забезпечення вже використовує деякі форми шаблонної кодогенерації. Наприклад, ORM (*Object-Relational Mapping*) інструменти часто застосовують кодогенерацію для створення класів на основі структур баз даних. Однак у майбутньому цю ідею можна взяти на новий рівень, послуговуючись таким:

• *адаптивною кодогенерацією:* інструменти можуть стати «розумнішими», даючи змогу розробникам створювати більш гнучкі шаблони, які можуть адаптуватися до конкретних потреб проєкту;

• *інтеграцією з хмарними сервісами:* інструменти кодогенерації можуть автоматично інтегруватися з хмарними сервісами, автоматично налаштовуючи інфраструктуру та інші сервіси;

• *візуальним програмуванням:* графічні інтерфейси можуть дозволити розробникам «будувати» програми, використовуючи блоки коду, які автоматично генеруються та оптимізуються.

Кодогенерація бізнес-логіки зі штучним інтелектом (ШІ). У майбутньому ми всі очікуватиме, що ШІ буде здатний генерувати бізнес-логіку на основі вимог та специфікацій. Розробники зазначать цілі та обмеження, а ШІ створить оптимальний код для завдання.

Насправді, можливості ШІ у сфері кодогенерації мають великий потенціал. Штучний інтелект здатний використовувати великі набори даних, щоб «навчитися» писати код, аналізуючи зразки та визначаючи найкращі практики. Це відбуватиметься в такий спосіб:

• *співпраця людини та машини.* Розробники можуть зазначати загальні вимоги або алгоритми, а ШІ буде автоматично генерувати оптимізований код;

• *автоматична оптимізація.* На основі аналізу виконання коду ШІ може автоматично рефакторити та оптимізовувати код, поліпшуючи його продуктивність та відповідність стандартам;

• *прогнозування проблем.* ШІ може виявляти потенційні проблеми в коді ще до їх виникнення, попереджаючи розробників про можливі помилки чи вразливості.

Висновки

Ці прогнози можуть змінюватися залежно від технологічних досягнень та потреб ринку, оскільки є загальними. Найголовніше, що кодогенерація, безумовно, відіграватиме важливу роль у майбутніх розробленнях програмного забезпечення.

Список використаної літератури

*1. **Tornhill A.** Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis. The Pragmatic Bookshelf. 2018.*

*2. **Fowler M.** Refactoring: Improving the Design of Existing Code. Addison-Wesley. 2019.*

*3. **Klabnik S., Nichols C.** The Rust Programming Language. No Starch Press. 2018.*

*4. **Redmon J., Farhadi A.** YOLOv3: An Incremental Improvement. arXiv. 2018.*

*5. **One** trillion edges: Graph processing at Facebook-scale / A. Ching, S. Edunov, I. Kabiljo [et al.] // Proceedings of the VLDB Endowment. 2018.*

*6. **Enck W.** Mobile Application Security // IEEE Internet Computing. 2018.*

*7. **A software** engineering experiment in software component generation / R. Kieburtz [et al.] // Proceedings of the 18th international conference on Software engineering. 1996.*

*8. **Czarnecki K., Eisenecker U.** Generative programming: methods, tools, and applications. Addison-Wesley. 2000.*

*9. **France R., Rumpe B.** Model-driven development of complex software: A research roadmap // Future of Software Engineering (FOSE '07). 2007.*

*10. **Greenfield J., Short S.** Software factories: assembling applications with patterns, models, frameworks, and tools. Wiley. 2003.*

*11. **Voelter M.** DSL Engineering - Designing, Implementing and Using Domain-Specific Languages. dslbook.org. 2013.*

*12. **Pizka M.** Straightforward software development: A step by step guide to the principles and pitfalls of modern software development // University of Applied Sciences Rosenheim. 2008.*

*13. **Schmidt D. C.** Model-driven engineering // IEEE Computer. 2006. 39(2). P. 25–31.*

*14. **Stahl T., Völter M.** Model-driven software development: technology, engineering, management. John Wiley & Sons. 2006.*

V. O. Mykolaienko, K. P. Storchak

CODE GENERATION FOR LARGE COMMERCIAL PROJECTS: HOW COMMERCIAL DEVELOPMENT WILL LOOK LIKE IN THE FUTURE

In the rapidly evolving landscape of software development, code generation stands at the forefront as a transformative force, especially within the domain of large-scale commercial projects. This paradigm-shifting technique is fundamental to agile adaptation in response to ever-changing business imperatives, all while upholding stringent code quality standards and reducing error rates. The article initiates a discourse by methodically analyzing the current methodologies and tools of code generation, emphasizing their critical importance for development teams. It delves into the complexities these tools mitigate, such as abstraction of repetitive coding tasks and acceleration of the development lifecycle. The discussion progresses to address the intricacies of scaling code generation processes, including issues of standardization, integration with legacy systems, and ensuring the maintained quality of automatically generated code segments. A particular focus is laid on the prospective role of artificial intelligence in the genesis of business logic, postulating a future where AI extends beyond mere automation, becoming a vital contributor to the creation of self-adapting, precision-targeted business functionalities. I introduce a structured approach to template code generation that underscores the importance of quick development setups and outline a set of principles for the AI-driven derivation of business logic. Moreover, the article offers an insightful prognosis on the potential reformation of software development paradigms by code generation technologies, looking into the crystal ball of this field's evolutionary path. It concludes with a visionary examination of the implications for the software industry, charting out the roadmap for practitioners to navigate and adapt to these upcoming shifts in commercial development methodologies.

Keywords: code generation; commercial development; automation; OpenAPI; Swagger; ROSLYN; technological achievements.