УДК 004.8 DOI: 10.18372/2073-4751.85.21089

[1]**Артамонов Є. Б.,**
orcid.org/0000-0002-9875-7372,
eart@kai.edu.ua,
[2]**Петренко С. О.,**
orcid.org/0009-0004-0603-8883,
petrenko.srh@gmail.com,
[3]**Жултинська А. К.,**
orcid.org/0000-0001-9178-897X
angelinaremark1@gmail.com,
[4]**Плотиця С. В.,**
orcid.org/0009-0007-3323-9316,
stepan@plotytsia.com,

ПРИНЦИПИ ВИКОРИСТАННЯ ШІ-АГЕНТІВ В ПОБУДОВІ ШАБЛОНІВ ПРОГРАМНИХ КОДІВ

[1,2,3]Державний університет "Київський авіаційний інститут",
[2]Grid Dynamics Holdings

Вступ

Розвиток інформаційних технологій характеризується стрімким зростанням складності програмних систем, що зумовлено переходом до мікросервісних архітектур, широким використанням розподілених обчислень та інтеграцією штучного інтелекту в програмні продукти. У цих умовах особливої актуальності набуває автоматизація процесів розробки програмного забезпечення, зокрема генерації коду та формування його шаблонів.

Поява великих мовних моделей (*Large Language Models*, *LLM*), здатних генерувати програмний код на основі текстових описів, відкрила нові можливості для автоматизації програмування. Дослідження показують, що сучасні моделі можуть успішно виконувати задачі генерації коду, аналізу програм та синтезу алгоритмів, демонструючи результати, близькі до рівня професійних розробників [1]. Подальший розвиток цього напрямку представлений у відкритих моделях генерації коду, які забезпечують масштабованість та доступність таких рішень [2], а також у підходах, що дозволяють досягати рівня складних алгоритмічних задач [3].

Спеціалізовані моделі Значний внесок у розвиток технологій генерації коду зроблено завдяки спеціалізованим моделям, орієнтованим на програмування, які враховують структуру та семантику програмного коду [4]. Крім того, сучасні підходи до програмного синтезу демонструють можливість використання *LLM* для автоматичного створення програм на основі формальних або напівформальних описів задач [5]. Розвиток цих ідей привів до появи моделей нового покоління, що поєднують генерацію та розуміння коду, забезпечуючи більш високий рівень узгодженості результатів [6].

Разом із цим, використання ШІ у програмній інженерії не обмежується окремими моделями. Значну увагу привертає концепція агентних систем, у яких великі мовні моделі виступають як автономні або напівавтономні агенти, здатні виконувати складні послідовності дій. Останні дослідження демонструють потенціал таких систем у вирішенні комплексних задач, включаючи розробку програмного забезпечення, де агенти можуть взаємодіяти між собою, планувати дії та перевіряти результати [7].

Формування багатокомпонентних агентних систем, орієнтованих на виконання задач програмування, стало логічним етапом еволюції *LLM*-технологій. У таких системах реалізується розподіл ролей між агентами, що дозволяє досягти більшої надійності та якості результатів у порівнянні з використанням окремих моделей [8, 9].

Незважаючи на значні досягнення у сфері генерації програмного коду, існує низка невирішених проблем, що обмежують практичне застосування таких підходів. Однією з ключових є нестабільність результатів генерації, яка проявляється у варіативності та відсутності гарантованої якості коду навіть за однакових вхідних умов [10]. Це ускладнює використання *LLM* у критичних програмних системах.

Іншою важливою проблемою є відсутність формалізованих підходів до побудови шаблонів коду. Більшість сучасних рішень орієнтовані на генерацію окремих фрагментів програм, а не на створення узгоджених шаблонів, які можуть бути використані як основа для розробки складних систем. При цьому шаблони коду відіграють ключову роль у забезпеченні стандартизації, повторного використання та підтримуваності програмного забезпечення.

У практиці використання інструментів генерації коду виявлено проблеми, пов'язані з безпекою та коректністю результатів, так дослідження [11] показують, що автоматично згенерований код може містити вразливості або не відповідати вимогам безпеки, і спостерігається розрив між очікуваннями розробників та реальними можливостями інструментів генерації коду, що ускладнює їх інтеграцію в процес розробки [12].

Важливим аспектом є взаємодія розробників з інструментами генерації коду, в якій ефективність таких інструментів значною мірою залежить від контексту використання та здатності системи адаптуватися до стилю програмування користувача [13, 14].

Метою даної роботи є розробка принципів використання ШІ-агентів для побудови шаблонів програмного коду, що забезпечують підвищення ефективності, якості та узгодженості процесу розробки програмного забезпечення.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати сучасні підходи до генерації програмного коду та використання *LLM* у програмній інженерії.
2. Дослідити концепції використання ШІ-агентів у задачах розробки програмного забезпечення.
3. Визначити основні обмеження існуючих підходів до формування шаблонів коду.
4. Сформулювати принципи використання ШІ-агентів для побудови шаблонів програмного коду.
5. Запропонувати архітектурну модель системи, що реалізує зазначені принципи.

Аналіз досліджень

Початкові підходи до автоматизації розробки програмного забезпечення базувалися на використанні заздалегідь визначених шаблонів та правил, що дозволяло формувати типові структури програмного коду. Такі підходи широко застосовуються у вигляді інструментів підтримки (scaffolding), генераторів каркасів проєктів та систем автоматичного створення типових компонентів.

Основною перевагою шаблонних підходів є їх детермінованість та передбачуваність результатів, що забезпечує стабільну якість згенерованого коду. Однак вони мають суттєві обмеження, зокрема низьку гнучкість та відсутність адаптації до контексту конкретного проєкту. Зі зростанням складності програмних систем такі підходи стають недостатніми,

оскільки не враховують специфіку задач та взаємодію компонентів.

Наступним етапом розвитку автоматизації генерації коду стало застосування методів машинного навчання та нейронних мереж. У цих підходах програмний код розглядається як послідовність токенів або структурованих елементів, що дозволяє застосовувати моделі послідовностей для його генерації.

Ранні дослідження у цій галузі продемонстрували можливість автоматичного синтезу програм на основі статистичних моделей та нейронних мереж, які враховують синтаксичну структуру коду [1]. Подальший розвиток підходів привів до створення моделей, що враховують не лише послідовність токенів, але й структурні залежності, зокрема потоки даних у програмі, що підвищує якість генерації [2].

Значним досягненням стало використання великих датасетів програмного коду для навчання моделей, що дозволило підвищити узагальнюючу здатність систем. Такі моделі здатні генерувати коректні фрагменти коду для типових задач, однак вони все ще обмежені у здатності працювати з комплексними програмними системами.

Сучасний етап розвитку технологій генерації коду пов’язаний із застосуванням великих мовних моделей, які демонструють високий рівень ефективності у задачах програмного синтезу. Такі моделі навчаються на великих обсягах текстових та програмних даних, що дозволяє їм враховувати як синтаксичні, так і семантичні особливості коду.

Дослідження показують, що великі мовні моделі здатні виконувати складні задачі генерації коду, включаючи розв’язання алгоритмічних задач та створення функціональних програмних модулів [3]. Розвиток відкритих моделей генерації коду, таких як CodeGen, дозволив зробити ці технології доступнішими та придатними для інтеграції у практичні системи [4].

Подальші дослідження спрямовані на підвищення якості генерації за рахунок урахування контексту, що реалізується у спеціалізованих моделях, орієнтованих на програмний код [5]. Такі моделі поєднують можливості генерації та розуміння коду, що дозволяє підвищити узгодженість та коректність результатів [6].

Разом із тим, незважаючи на значні досягнення, LLM мають обмеження, пов’язані з нестабільністю результатів та відсутністю гарантій якості, що ускладнює їх використання для формування стандартизованих шаблонів програмного коду.

З розвитком великих мовних моделей сформувався новий напрям досліджень, пов’язаний із використанням ШІ-агентів, які здатні виконувати складні задачі шляхом взаємодії з середовищем та іншими агентами. У цьому контексті агент визначається як автономна система, що може приймати рішення, планувати дії та виконувати їх для досягнення поставленої мети.

Сучасні дослідження демонструють, що поєднання великих мовних моделей з агентними підходами дозволяє створювати системи, здатні виконувати складні послідовності дій, включаючи розробку програмного забезпечення [7]. Такі системи можуть аналізувати вимоги, генерувати код, перевіряти результати та вносити зміни у процесі виконання задачі.

Одним із ключових напрямів розвитку архітектури агентних систем для розробки ПЗ є побудова мультиагентних систем, у яких різні агенти виконують спеціалізовані функції. Дослідження показують, що розподіл функцій між агентами дозволяє досягти більш високого рівня якості результатів у порівнянні з використанням окремих моделей [8]. Крім того, взаємодія агентів може бути організована у вигляді ітеративного процесу, що включає

генерацію, перевірку та вдосконалення результатів. Підходи до побудови таких систем активно розвиваються у напрямі використання комунікативних агентів, які обмінюються інформацією та координують свої дії для досягнення спільної мети [9].

З розвитком технологій генерації коду з'явилися інструменти, що дозволяють автоматично формувати шаблони програмного коду на основі заданих параметрів або описів. Такі системи можуть використовувати як шаблонні підходи, так і методи машинного навчання.

Дослідження показують, що автоматизовані генератори дозволяють значно скоротити час розробки та підвищити продуктивність програмістів [10]. Разом із тим, такі системи можуть генерувати код, що не відповідає вимогам безпеки або містить помилки, що підтверджується результатами досліджень у сфері аналізу якості згенерованого коду [11].

Постановка проблеми

Сучасні підходи до генерації програмного коду, незважаючи на значний розвиток, не забезпечують достатнього рівня узгодженості, повторюваності та адаптивності результатів у контексті побудови шаблонів програмних систем. Це зумовлює необхідність формалізації задачі генерації шаблонів коду як окремої підзадачі програмної інженерії.

У загальному вигляді задачу генерації шаблонів коду можна представити як відображення:

$$F:(R,C,S) \rightarrow T, \qquad (1)$$

де:

R – множина вимог до програмної системи;

C – контекст розробки;

S – технологічний стек;

T – структурований шаблон програмного коду.

Вхідні дані задачі генерації шаблонів коду мають комплексний характер і включають кілька взаємопов'язаних компонентів.

Множина вимог R визначає функціональні та нефункціональні характеристики програмної системи. До функціональних вимог належать опис бізнес-логіки, сценарії використання та очікувана поведінка системи. Нефункціональні вимоги включають обмеження щодо продуктивності, безпеки, масштабованості та підтримуваності.

Контекст розробки C охоплює:

– архітектурний стиль;

– структуру системи та взаємодію компонентів;

– стиль програмування та внутрішні стандарти організації;

– наявні бібліотеки та фреймворки.

Контекст відіграє ключову роль у забезпеченні узгодженості шаблонів із існуючою системою та визначає допустимі варіанти реалізації.

Технологічний стек S включає:

– мову програмування;

– фреймворки та бібліотеки;

– інструменти збірки та розгортання;

– середовище виконання.

Результатом виконання задачі є структурований шаблон програмного коду T, який представляє собою формалізований опис програмної системи або її компонентів.

Шаблон T включає:

– ієрархію файлів та директорій;

– опис модулів, класів та функцій;

– заготовки коду з визначеними інтерфейсами;

– конфігураційні файли;

– інтеграційні елементи (*API*, бази даних, сервіси).

Важливою характеристикою шаблону є його узагальненість та параметризованість, що дозволяє використовувати його повторно у різних проєктах або сценаріях. Крім того, шаблон повинен бути придатним для подальшого автоматичного або напівавтоматичного розширення.

Формально шаблон можна представити як:

$$T=\{M,F,I,Cfg\}, \quad (2)$$

де:

M – множина модулів;

F – множина функцій та методів;

I – інтерфейси взаємодії;

Cfg – конфігураційні параметри.

Процес генерації шаблонів коду обмежується рядом факторів, які необхідно враховувати при побудові системи.

До основних обмежень належать:

– обмеження технологічного стеку (сумісність бібліотек, версії середовищ);

– обмеження архітектурного характеру;

– обмеження ресурсів (час генерації, обчислювальні ресурси);

– обмеження, пов'язані з якістю навчальних даних моделей.

Для оцінювання якості згенерованого шаблону вводиться система критеріїв:

1. Коректність (*Correctness*) – відповідність синтаксису та семантики коду.

2. Узгодженість (*Consistency*) – відповідність шаблону архітектурі системи.

3. Повторюваність (*Reusability*) – можливість повторного використання шаблону.

4. Розширюваність (*Extensibility*) – можливість подальшої модифікації.

5. Безпечність (*Security*) – відсутність вразливостей.

З урахуванням цих критеріїв задачу генерації шаблонів коду можна розглядати як задачу багатокритеріальної оптимізації:

$$T^* = \underset{T}{argmax}\left(Q_c + Q_s + Q_r + Q_e + Q_{sec}\right), \quad (3)$$

де Q_i – відповідні показники якості.

Незважаючи на значний прогрес у сфері автоматизованої генерації програмного коду, сучасні підходи мають ряд суттєвих обмежень, які ускладнюють їх застосування для побудови шаблонів програмних систем. Аналіз існуючих рішень дозволяє виділити ключові проблеми, пов'язані з відсутністю контекстної узгодженості, нестабільністю результатів генерації та недостатністю механізмів валідації та контролю якості.

Однією з основних проблем сучасних систем генерації коду є недостатнє врахування контексту розробки. Більшість моделей, зокрема великі мовні моделі, генерують код на основі локального текстового запиту, що обмежує їх здатність враховувати глобальну архітектуру програмної системи.

У результаті цього виникає ситуація, коли згенеровані фрагменти коду:

– не узгоджуються між собою;

– не відповідають прийнятим архітектурним рішенням;

– порушують внутрішні стандарти проєкту.

Особливо критичною ця проблема є при генерації шаблонів, оскільки шаблон передбачає не окремий фрагмент коду, а узгоджену структуру всієї системи або її підсистеми. Відсутність контекстної обізнаності призводить до того, що шаблони мають фрагментарний характер і потребують значного ручного доопрацювання.

Іншою суттєвою проблемою є нестабільність результатів генерації коду. Великі мовні моделі характеризуються стохастичним характером роботи, що призводить до варіативності результатів навіть при однакових вхідних даних.

Це проявляється у таких аспектах:

– різні варіанти реалізації однієї і тієї ж функціональності;

– відсутність детермінованості у структурі коду;

– зміна стилю програмування та архітектурних рішень.

Для задачі генерації шаблонів ця проблема є особливо критичною, оскільки шаблон повинен бути стандартизованим і повторюваним.

Нестабільність генерації унеможливлює використання результатів як базових шаблонів без додаткової обробки.

Запропоноване рішення

Запропоноване рішення базується на використанні багаторівневої системи ШІ-агентів для генерації шаблонів програмного коду, що забезпечує інтеграцію контексту розробки, стабільність результатів та автоматичний контроль якості.

На відміну від традиційних підходів, у яких генерація коду здійснюється однією моделлю, запропонована концепція передбачає розподіл функціональності між спеціалізованими агентами. Кожен агент виконує окрему роль у процесі формування шаблону, що дозволяє підвищити якість та узгодженість результатів.

Такий підхід дозволяє перейти від одноетапної генерації до ітеративного процесу побудови шаблону, що враховує як вхідні вимоги, так і результати проміжних перевірок.

У запропонованій системі кожен агент виконує чітко визначену функцію, що відповідає окремому етапу обробки даних.

Виділяються такі основні типи агентів:

1. Агент аналізу (*Analysis Agent*), що виконує обробку вхідних даних, інтерпретує вимоги, визначає ключові характеристики системи та формує внутрішнє представлення задачі.

2. Агент планування (*Planning Agent*), що формує структуру майбутнього шаблону, визначає модулі, компоненти та взаємозв'язки між ними. На цьому етапі реалізується декомпозиція задачі.

3. Агент генерації (*Generation Agent*), що створює безпосередньо програмний код та структуру шаблону на основі сформованого плану.

4. Агент валідації (*Validation Agent*), що перевіряє коректність згенерованого коду, включаючи синтаксичну правильність, відповідність стандартам та узгодженість компонентів.

5. Агент оптимізації (*Optimization Agent*), що виконує рефакторинг, покращення структури коду та адаптацію шаблону до заданих критеріїв якості.

Такий розподіл ролей дозволяє реалізувати принцип модульності та забезпечує можливість незалежного вдосконалення окремих компонентів системи.

Взаємодія агентів організована у вигляді ітеративного процесу, що забезпечує поступове вдосконалення результату генерації.

На першому етапі агент аналізу формує структуроване представлення вхідних даних, яке передається агенту планування. Агент планування, у свою чергу, створює модель майбутнього шаблону, що визначає структуру та взаємозв'язки компонентів.

Далі агент генерації створює початковий варіант шаблону, який передається агенту валідації. У разі виявлення невідповідностей або помилок результат повертається на попередні етапи для корекції. Такий механізм реалізує зворотний зв'язок у системі.

Запропонований підхід має низку переваг у порівнянні з традиційними методами генерації коду:

– забезпечується контекстна узгодженість, оскільки аналіз та планування виконуються окремими агентами, які враховують глобальну структуру системи;

– досягається стабільність результатів завдяки ітеративному процесу генерації та використанню механізмів зворотного зв'язку;

– реалізується контроль якості, що забезпечується наявністю окремого агента валідації, який перевіряє результати генерації.

– система має модульну структуру, що дозволяє масштабувати її та адаптувати до різних задач.

– забезпечується гнучкість та розширюваність, оскільки нові агенти

можуть бути додані без зміни загальної архітектури системи.

Запропонована система генерації шаблонів програмного коду базується на сукупності принципів, що визначають її архітектуру, логіку функціонування та взаємодію компонентів. На відміну від існуючих підходів, де генерація здійснюється переважно на основі окремих моделей, запропонована система орієнтована на інтеграцію контексту, ітеративну обробку результатів та модульну організацію агентів.

Сформульовані принципи забезпечують усунення виявлених недоліків, зокрема нестабільності генерації, відсутності контекстної узгодженості та механізмів контролю якості.

Принцип контекстної обізнаності передбачає, що процес генерації шаблонів коду повинен здійснюватися з урахуванням повного контексту розробки, включаючи архітектурні, технологічні та організаційні аспекти.

Контекст C формується як інтегрована множина:

$$C=\{A,S,G,H\}, \quad (4)$$

де:

A – архітектура системи;

S – технологічний стек;

G – правила та стандарти програмування;

H – історія змін і попередні реалізації.

Відповідно до цього принципу, агенти повинні:

– використовувати інформацію з репозиторіїв коду;

– враховувати існуючі шаблони та патерни;

– адаптувати результати до стилю проєкту.

Реалізація принципу контекстної обізнаності забезпечує узгодженість згенерованих шаблонів із загальною структурою програмної системи.

Принцип багаторівневої генерації передбачає, що генерація шаблонів коду повинна здійснюватися на кількох рівнях абстракції, що дозволяє забезпечити структурну узгодженість системи.

Виділяються такі рівні генерації:

– рівень архітектури (структура системи);

– рівень модулів (компоненти та їх взаємодія);

– рівень класів і сервісів;

– рівень функцій та методів.

Формально процес генерації можна представити як послідовність перетворень:

$$T=G_4(G_3(G_2(G_1(R)))), \quad (5)$$

де G_i – оператор генерації на відповідному рівні.

Такий підхід дозволяє:

– забезпечити узгодженість між компонентами;

– уникнути конфліктів між рівнями абстракції;

– підвищити керованість процесу генерації.

Принцип ітеративної валідації передбачає, що генерація шаблонів коду повинна супроводжуватися постійною перевіркою та корекцією результатів на кожному етапі.

Процес описується циклом:

$$T_{i+1}=V(O(T_i)), \quad (6)$$

де

T_i – поточний шаблон;

O – оператор оптимізації;

V – оператор валідації.

Валідація включає:

– синтаксичну перевірку;

– аналіз відповідності стандартам;

– перевірку архітектурної узгодженості;

– тестування базових сценаріїв.

Ітеративний характер процесу дозволяє:

– зменшити кількість помилок;

– підвищити якість шаблонів;

– забезпечити стабільність результатів.

Принцип модульності агентів передбачає, що система повинна бути побудована як набір незалежних, але взаємодіючих агентів, кожен з яких виконує окрему функцію.

Формально система агентів може бути представлена як множина:

$$A=\{A_{an},A_{pl},A_{gen},A_{val},A_{opt}\}, \quad (7)$$

де:

A_{an} – агент аналізу;

A_{pl} – агент планування;

A_{gen} – агент генерації;

A_{val} – агент валідації;

A_{opt} – агент оптимізації.

Модульність забезпечує:

– незалежність компонентів;

– можливість масштабування;

– спрощення розробки та тестування;

– гнучкість при зміні архітектури системи.

Принцип інтеграції з *DevOps* передбачає інтеграцію системи генерації шаблонів коду з процесами *DevOps*, що забезпечує автоматизацію перевірки та впровадження результатів.

Система повинна підтримувати:

– інтеграцію з системами контролю версій;

– автоматичний запуск тестів;

– аналіз якості коду (*linting*, *static analysis*);

– автоматичне розгортання шаблонів.

Формально інтеграцію можна представити як розширення функції генерації:

$$T'=D(T), \quad (8)$$

де D – оператор *DevOps*-перевірки та інтеграції.

Реалізація цього принципу дозволяє:

– забезпечити контроль якості на рівні інфраструктури;

– інтегрувати генерацію коду у реальні процеси розробки;

– підвищити довіру до результатів системи.

У системі реалізуються три основні типи потоків даних:

1. Прямий потік – це основний конвеєр обробки, у межах якого дані проходять усі етапи трансформації – від вимог і контексту до інтегрованого результату. Цей потік відображає основну логіку роботи системи.

2. Зворотний потік – якщо агент валідації або агент оптимізації виявляє помилки, неповноту або неузгодженість, формується зворотний потік до попередніх модулів:

– від валідації до генерації – якщо шаблон потребує повторного формування;

– від оптимізації до планування – якщо виявлено структурні недоліки;

– від валідації до аналізу – якщо проблема закладена ще у некоректній інтерпретації вимог.

3. Контекстний потік – паралельно з основним потоком у систему надходять дані з зовнішніх джерел:

– репозиторіїв коду;

– внутрішніх стандартів проєкту;

– бібліотек шаблонів;

– документації;

– *CI*/*CD*-правил;

– історії змін.

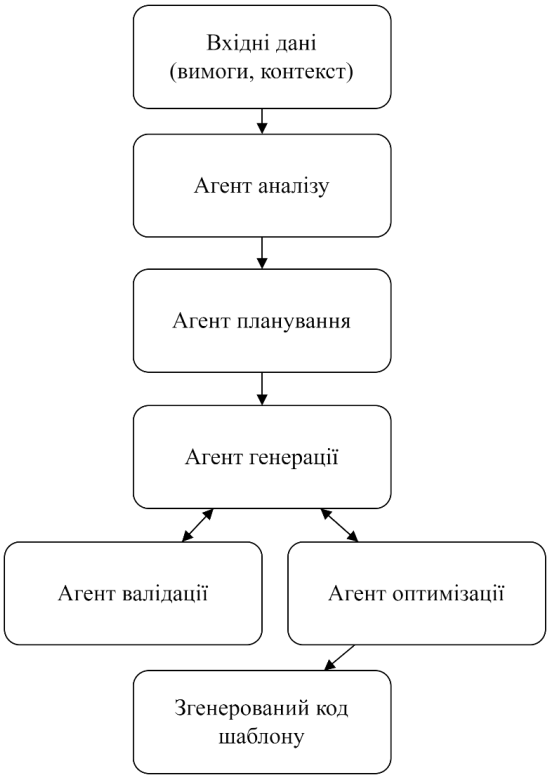

Рис. 1. Архітектура системи ШІ-агентів для генерації шаблонів програмного коду з потоками даних

Цей потік забезпечує контекстну обізнаність агентів і підвищує узгодженість генерації (рис. 1). Архітектура системи генерації шаблонів програмного коду реалізує послідовну обробку вхідних вимог через агент аналізу, агент планування, агент генерації, агент валідації та агент оптимізації.

Основний потік даних забезпечує перетворення опису задачі у структурований шаблон коду, тоді як зворотні потоки підтримують ітеративне уточнення результату у разі виявлення помилок або невідповідностей. Додатково архітектура передбачає інтеграцію з *DevOps*-середовищем, що дозволяє автоматизувати перевірку та впровадження згенерованих шаблонів у практичний процес розробки.

Алгоритм роботи (рис. 2) запропонованої системи генерації шаблонів програмного коду реалізує поетапну обробку вхідних даних із використанням взаємодії ШІ-агентів. Процес має ітеративний характер і передбачає можливість повернення до попередніх етапів у разі виявлення помилок або невідповідностей.

У загальному вигляді алгоритм складається з трьох основних етапів: планування, генерації та перевірки з оптимізацією.

На етапі планування здійснюється формування структури майбутнього шаблону на основі вхідних даних.

Оцінювання ефективності системи здійснюється на основі сукупності критеріїв, що відображають ключові вимоги до генерації шаблонів програмного коду.

До основних критеріїв належать:

1. Час генерації (T) – характеризує швидкість формування шаблону. Менше значення відповідає більш ефективній системі.

2. Якість коду (Q) – включає синтаксичну коректність, відповідність стандартам, читабельність та структурованість.

Рис. 2. Алгоритм роботи системи генерації шаблонів програмного коду

3. Рівень узгодженості (C) – відображає відповідність шаблону архітектурі системи та контексту розробки.

4. Рівень повторного використання (R) – визначає можливість використання шаблону в інших проєктах або сценаріях.

5. Рівень помилок (E) – характеризує кількість синтаксичних,

логічних та архітектурних помилок у шаблоні.

6. Рівень автоматизації (A) – визначає ступінь участі людини у процесі генерації.

Враховуючи різну природу критеріїв, їх доцільно нормалізувати до єдиної шкали, де більші значення відповідають кращим показникам (для E використовується обернене значення).

Запропонований підхід до генерації шаблонів програмного коду на основі ШІ-агентів демонструє ряд переваг у порівнянні з традиційними методами та підходами, що базуються виключно на великих мовних моделях. Зокрема, поєднання багаторівневої генерації, контекстної обізнаності та ітеративної валідації дозволяє підвищити якість результатів і забезпечити їх стабільність.

Однією з ключових переваг запропонованого підходу є підвищення якості згенерованого коду. На відміну від одноетапної генерації, агентна система забезпечує багаторівневу обробку, що включає аналіз, планування, генерацію та перевірку результатів.

Дослідження показують, що використання окремих інструментів генерації коду може призводити до появи некоректних або небезпечних конструкцій у програмному коді [15]. Запропонований підхід частково усуває цю проблему за рахунок введення окремого етапу валідації та оптимізації.

Використання ШІ-агентів дозволяє значно скоротити час розробки програмного забезпечення за рахунок автоматизації процесів створення шаблонів коду.

Згідно з дослідженнями, застосування інструментів генерації коду може зменшити час виконання типових задач програмування, однак ефективність таких інструментів залежить від їх інтеграції у процес розробки [17]. У запропонованій системі ця проблема вирішується шляхом інтеграції генерації, перевірки та оптимізації в єдиний процес.

Модульна структура агентної системи забезпечує її масштабованість та адаптивність до різних задач. Додавання нових агентів або розширення функціональності існуючих компонентів може здійснюватися без суттєвого впливу на інші частини системи.

Дослідження у сфері мультиагентних систем підтверджують, що розподіл функцій між агентами дозволяє підвищити ефективність обробки складних задач та забезпечити гнучкість системи [18]. У контексті генерації коду це означає можливість адаптації системи до різних типів проєктів, мов програмування та архітектурних підходів.

Незважаючи на переваги, запропонований підхід має певні обмеження та пов'язані з ним ризики, які необхідно враховувати при його практичному застосуванні.

Ефективність агентної системи значною мірою залежить від якості великих мовних моделей, які використовуються для генерації та аналізу коду.

Як показують дослідження, навіть сучасні моделі можуть генерувати некоректні або неповні рішення, особливо у випадку складних або нетипових задач [19]. У таких умовах якість результатів агентної системи безпосередньо залежить від точності та узагальнюючої здатності базових моделей.

Незважаючи на наявність механізмів валідації, система не гарантує повну відсутність помилок у згенерованому коді. Помилки можуть виникати як на етапі генерації, так і внаслідок некоректної інтерпретації вхідних даних.

Дослідження показують, що інструменти генерації коду можуть створювати рішення, які виглядають коректними, але містять приховані логічні помилки [20]. Це підкреслює необхідність використання додаткових механізмів перевірки, зокрема

автоматичного тестування та статичного аналізу.

Висновки

У роботі розглянуто проблему підвищення ефективності процесу генерації шаблонів програмного коду в умовах зростаючої складності сучасних програмних систем. Проведений аналіз існуючих підходів показав, що традиційні шаблонні методи, а також сучасні рішення на основі великих мовних моделей, не забезпечують достатнього рівня узгодженості, стабільності та контролю якості результатів, що обмежує їх застосування у задачах побудови структурованих шаблонів програмного забезпечення.

У процесі дослідження сформульовано задачу генерації шаблонів програмного коду як задачу перетворення сукупності вимог, контексту та технологічного стеку у структурований результат із заданими характеристиками якості. На основі цього запропоновано підхід до використання ШІ-агентів, що передбачає розподіл функцій між спеціалізованими компонентами системи та реалізацію ітеративного процесу генерації, перевірки та оптимізації.

Розроблено систему принципів побудови агентної системи, яка включає контекстну обізнаність, багаторівневу генерацію, ітеративну валідацію, модульність та інтеграцію з DevOps-процесами. Зазначені принципи забезпечують усунення виявлених недоліків існуючих рішень та формують основу для побудови узгоджених і придатних до використання шаблонів програмного коду.

Запропоновано архітектуру системи генерації шаблонів коду, яка реалізує багаторівневу взаємодію агентів аналізу, планування, генерації, валідації та оптимізації. Описано алгоритм функціонування системи, що забезпечує поетапну обробку даних із можливістю зворотного зв'язку та ітеративного вдосконалення результатів.

Наукова новизна отриманих результатів полягає у формалізації принципів використання ШІ-агентів для побудови шаблонів програмного коду, розробці багаторівневої агентної архітектури та удосконаленні підходу до оцінювання ефективності систем генерації без проведення експериментальних досліджень.

Практичне значення роботи полягає у можливості застосування запропонованого підходу при створенні інтелектуальних систем підтримки програмування, інтеграції в середовища розробки та DevOps-процеси, а також у використанні для стандартизації розробки програмного забезпечення в корпоративних умовах.

Подальший розвиток запропонованого підходу може бути спрямований на вдосконалення як окремих компонентів системи, так і її загальної архітектури.

Одним із перспективних напрямів є адаптація системи до конкретних умов організації шляхом навчання агентів на корпоративних репозиторіях коду.

Іншим важливим напрямом є інтеграція системи з сучасними середовищами розробки, що дозволить забезпечити інтерактивну взаємодію користувача з агентами.

Література

1. *Chen*, *M*., *Tworek*, *J*., *Jun*, *H*., *et al*. *Evaluating Large Language Models Trained on Code*. *arXiv preprint*. *DOI*: *https*://*doi*.*org*/10.48550/*arXiv*.2107.03374

2. *Nijkamp*, *E*., *Hayashi*, *H*., *Xie*, *C*., *et al*. *CodeGen*: *An Open Large Language Model for Code Generation*. *ACM Transactions on Machine Learning Research*, 2023. *DOI*: *https*://*doi*.*org*/10.48550/*arXiv*.2203.13474

3. *Li*, *Y*., *Wang*, *S*., *Wang*, *H*., *et al*. *Competition-Level Code Generation with AlphaCode*. *Science*, 2022. *DOI*: https://doi.org/10.1126/science.abq1158

4. *Rozière*, *B*., *Lachaux*, *M*.-*A*., *Chanussot*, *L*., *Lample*, *G*. *Code Llama*: *Open Foundation Models for Code*. *arXiv* /

Meta AI, 2023. *DOI*: *https*://*doi.org*/10.48550/*arXiv*.2308.12950

5. *Austin*, *J.*, *Odena*, *A.*, *Nye*, *M.*, *et al*. *Program Synthesis with Large Language Models*. *NeurIPS*, 2023. *DOI*: *https*://*doi.org*/10.48550/*arXiv*.2108.07732

Zhang, *F.*, *Chen*, *X.*, *Wan*, *Y.*, *et al*. *CodeT*5+: *Open Code Large Language Models for Code Understanding and Generation*. *IEEE Transactions on Software Engineering*, 2024. *DOI*: *https*://*doi.org*/10.48550/*arXiv*.2308.12950

6. *Guo*, *D.*, *Tang*, *D.*, *Duan*, *N.*, *et al*. *GraphCodeBERT*: *Pre-training Code Representations with Data Flow*. *ICLR / IEEE applications*, використовується у 2023–2025. *DOI*: *https*://*doi.org*/10.48550/*arXiv*.2009.08366

7. *Wang*, *Y.*, *Wang*, *W.*, *Joty*, *S.*, *Hoi*, *S. CodeGeeX*: *A Pre-trained Model for Code Generation*. *KDD / ACM*, 2023. *DOI*: https://doi.org/10.1145/3580305.3599790

8. *Yin*, *P.*, *Neubig*, *G. A Syntactic Neural Model for General-Purpose Code Generation*. *ACL / IEEE usage*, актуалізовано в нових роботах. *DOI*: *https*://*doi.org*/10.18653/*v*1/*P*17-1041

9. *Dong*, *Y.*, *Chen*, *X.*, *Wan*, *Y.*, *et al*. *Coder Reviewer Reranking for Code Generation*. *ACL Findings*, 2023. *DOI*: *https*://*doi.org*/10.18653/*v*1/2023.*findings-acl*.313

11. *Xi*, *Z.*, *Chen*, *W.*, *Guo*, *X.*, *et al*. *The Rise and Potential of Large Language Model Based Agents*: *A Survey*. *arXiv*, 2023. *DOI*: *https*://*doi.org*/10.48550/*arXiv*.2309.07864

12. *Wang*, *L.*, *Ma*, *Y.*, *Zhang*, *X.*, *et al*. *A Survey on Large Language Model Based Autonomous Agents*. *Frontiers of Computer Science*, 2024. *DOI*: *https*://*doi.org*/10.1007/*s*11704-024-3309-2

13. *Park*, *J. S.*, *O'Brien*, *J. C.*, *Cai*, *C. J.*, *et al*. *Generative Agents*: *Interactive Simulacra of Human Behavior*. *ACM UIST*, 2023. *DOI*: *https*://*doi.org*/10.1145/3586183.3606763

14. *Hong*, *S.*, *Zhu*, *Y.*, *Chen*, *X.*, *et al*. *MetaGPT*: *Meta Programming for Multi-Agent Collaborative Framework*. *arXiv*, 2023. *DOI*: *https*://*doi.org*/10.48550/*arXiv*.2308.00352

15. *Qian*, *C.*, *Cong*, *X.*, *Yang*, *W.*, *et al*. *Communicative Agents for Software Development*. *arXiv*, 2023. *DOI*: https://doi.org/10.48550/arXiv.2307.07924

16. *Pearce*, *H.*, *Ahmad*, *B.*, *Tan*, *B.*, *et al*. *Asleep at the Keyboard*? *Assessing the Security of GitHub Copilot*. *IEEE Symposium on Security and Privacy*, 2023. *DOI*: *https*://*doi.org*/10.1109/*SP*46215.2023.10179387

17. *Vaithilingam*, *P.*, *Zhang*, *T.*, *Glassman*, *E. Expectation vs Reality*: *Evaluating AI Code Generation Tools*. *CHI Conference*, 2023. *DOI*: *https*://*doi.org*/10.1145/3544548.3581382

18. *Barke*, *S.*, *James*, *M. B.*, *Polikarpova*, *N. Grounded Copilot*: *How Programmers Interact with Code-Generating Models*. *OOPSLA / ACM*, 2023. *DOI*: https://doi.org/10.1145/3586030

19. *Sobania*, *D.*, *Briesch*, *M.*, *Hanna*, *C.*, *et al*. *An Analysis of Code Generation by ChatGPT*. *IEEE Access*, 2023. *DOI*: *https*://*doi.org*/10.1109/*ACCESS*.2023.3273868

20. *Dakhel*, *A. M.*, *Majdinasab*, *V.*, *Nikanjam*, *A.*, *et al*. *GitHub Copilot AI Pair Programmer*: *Asset or Liability*? *Journal of Systems and Software*, 2023. *DOI*: https://doi.org/10.1016/j.jss.2023.111734

Артамонов Є.Б., Петренко С.О., Жултинська А.К., Плотиця С.В.

ПРИНЦИПИ ВИКОРИСТАННЯ ШІ-АГЕНТІВ В ПОБУДОВІ ШАБЛОНІВ ПРОГРАМНИХ КОДІВ

Стаття присвячена дослідженню принципів використання ШІ-агентів для побудови шаблонів програмного коду в умовах зростаючої складності сучасних програмних систем. Актуальність теми зумовлена обмеженнями традиційних підходів до генерації коду, такими як відсутність контекстної узгодженості, нестабільність результатів та недостатній рівень

контролю якості. У роботі проаналізовано сучасні підходи до автоматизації програмування, включаючи шаблонні методи, нейромережеві моделі та великі мовні моделі (LLM), а також досліджено концепції мультиагентних систем у програмній інженерії. Запропоновано формалізацію задачі генерації шаблонів коду як відображення множини вимог, контексту та технологічного стеку у структурований шаблон програмного забезпечення. Розроблено концепцію багаторівневої агентної системи, яка включає агентів аналізу, планування, генерації, валідації та оптимізації, що взаємодіють у рамках ітеративного процесу. Сформульовано систему принципів побудови такої системи, зокрема принципи контекстної обізнаності, багаторівневої генерації, ітеративної валідації, модульності та інтеграції з DevOps. Запропоновано архітектуру системи та алгоритм її функціонування з урахуванням прямих, зворотних та контекстних потоків даних. Для оцінювання ефективності підходу розроблено формалізовану багатокритеріальну модель, що дозволяє здійснювати теоретичне порівняння з альтернативними рішеннями. Отримані результати можуть бути використані при створенні інтелектуальних систем підтримки програмування та автоматизації процесів розробки програмного забезпечення.

***Ключові слова**: ШІ-агенти, генерація програмного коду, шаблони програмного забезпечення, мультиагентні системи, автоматизація розробки ПЗ, DevOps інтеграція.*

Artamonov Y.B., Petrenko S.O., Zhultynska A.K., Plotytsia S.V.

PRINCIPLES OF USING AI AGENTS IN THE CONSTRUCTION OF SOFTWARE CODE TEMPLATES

The paper is devoted to the study of principles for using AI agents in the construction of software code templates under conditions of increasing complexity of modern software systems. The relevance of the topic is determined by the limitations of traditional code generation approaches, such as the lack of contextual consistency, instability of results, and insufficient quality control. The study analyzes modern approaches to programming automation, including template-based methods, neural network models, and large language models (LLMs), and also examines the concepts of multi-agent systems in software engineering.

A formalization of the code template generation problem is proposed as a mapping of a set of requirements, context, and technology stack into a structured software template. A concept of a multi-level agent-based system is developed, which includes analysis, planning, generation, validation, and optimization agents interacting within an iterative process.

A system of principles for constructing such a system is formulated, including the principles of contextual awareness, multi-level generation, iterative validation, modularity, and integration with DevOps processes. The architecture of the system and the algorithm of its operation are proposed, taking into account direct, feedback, and contextual data flows.

To evaluate the effectiveness of the proposed approach, a formalized multi-criteria model is developed, enabling theoretical comparison with alternative solutions. The obtained results can be used in the development of intelligent programming support systems and the automation of software development processes.

***Keywords**: AI agents, code generation, software templates, multi-agent systems, software development automation, DevOps integration.*

Стаття подана до редакції: 23/03/2026
Стаття прийнята до опублікування: 27/03/2026
Стаття опублікована: 27/04/2026