УДК 004.8:004.415

ІНТЕГРАЦІЯ ГЕНЕРАТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ В ПРОЦЕСИ АВТОМАТИЗАЦІЇ ІНЖЕНЕРІЇ ВИМОГ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Поліщук О. В.
olenapolishchuk022@gmail.com
Черкаський державний фаховий бізнес-коледж
Марченко С. В.
м.Черкаси, Україна

Створення сучасного програмного забезпечення – це багаторівневий процес, де навіть незначна неточність на етапі проєктування може призвести до критичних наслідків. Інженерна практика доводить: етап збору вимог (Requirements Engineering) слугує базовим фундаментом розробки [3]. Практичний досвід IT-компаній підтверджує, що виправлення хибно зрозумілої вимоги під час написання коду чи тестування потребує найбільших фінансових та часових витрат [7]. Тому життєздатність проєкту прямо залежить від якості складеної специфікації.

Традиційно технічні завдання (Software Requirements Specifications) формулюються природною мовою. Це логічний компроміс, який дозволяє замовнику та команді знайти спільне розуміння задачі [6]. Водночас такий формат створює значну проблему. Природна мова відрізняється високим рівнем неоднозначності, містить надмірну інформацію, через що критичні технічні нюанси можуть просто загубитися [1]. Аби програмісти отримали чітку структуру, системні аналітики змушені витрачати години на ручний переклад цих об’ємних текстів у візуальні моделі (наприклад, UML-діаграми). Такий рутинний процес є тривалим і часто супроводжується помилками через «людський фактор» [2]. З огляду на це, автоматизація трансформації сирого тексту в графічні моделі стає необхідним кроком. Це не лише прискорить темпи розробки, але й суттєво зменшить імовірність архітектурних прорахунків.

Проаналізувати наявні інженерні методи обробки вимог, оцінити можливості штучного інтелекту в задачах аналізу тексту та спроєктувати

архітектуру системи для автоматичного перетворення технічних завдань у код побудови діаграм.

Останніми роками підходи до проєктування зазнали суттєвих змін. Індустрія поступово відмовляється від ручного малювання схем, переходячи до парадигми «Діаграми як код» (Diagrams as Code). Концепція полягає в тому, що інженер пише скрипт, а система генерує візуальну схему. Інструменти на зразок PlantUML або Mermaid.js набули широкого поширення, оскільки дозволяють зберігати історію змін архітектури безпосередньо в репозиторіях Git разом із первинним кодом.

Попри всі переваги, ці інструменти мають серйозне обмеження: вони працюють виключно як візуалізатори. Їм потрібен ідеальний машинний синтаксис, і вони не здатні самостійно інтерпретувати звичайний текст. Відповідно, найскладніша аналітична частина – прочитати вимоги, виділити бізнес-логіку та написати код діаграми – все ще залишається зоною відповідальності людини.

Вирішення цієї проблеми стало можливим завдяки великим мовним моделям (LLM) [9]. У їхній основі лежить архітектура Transformer, яка використовує механізм «уваги» (Self-Attention). На відміну від класичних алгоритмів, трансформери аналізують весь документ цілісно. Вони здатні знаходити взаємозв’язки між вимогами, навіть якщо ті знаходяться в різних розділах специфікації [4].

На практиці процес виглядає так: після передачі абзацу з бізнес-вимогами на вхід моделі (наприклад, сімейства GPT), алгоритм самостійно ідентифікує акторів системи, бази даних та правила їхньої взаємодії [1]. Після цього миттєво генерується синтаксис, який відразу трансформується у графічну схему [2].

Проте застосування базових моделей має свої ризики. Прямий запит до нейромережі часто призводить до генерації синтаксичних помилок або додавання зайвих коментарів (так званих «галюцинацій») [9]. Для забезпечення детермінованого результату фахівці застосовують Prompt Engineering (інженерію підказок) [5]. Нейромережі надають кілька еталонних

прикладів очікуваного результату (Few-Shot Prompting) [8] та встановлюють суворі системні обмеження. Це змушує алгоритм ігнорувати розмовний стиль і генерувати виключно чистий, готовий до компіляції код діаграми [5].

Для практичного застосування проєктується мікросервісний вебдодаток. Серверна частина (backend) відповідає за взаємодію з ШІ через API: вона отримує згенерований код, очищає його від можливих артефактів і передає на клієнтську сторону (frontend). Браузер виконує лише рендеринг графіки. Завдяки цьому системний аналітик уникає необхідності працювати з промптами чи синтаксисом − він просто вводить текст і отримує готову архітектурну модель.

Точність генерації можна додатково підвищити шляхом донавчання (fine-tuning) моделі на базах реальної технічної документації [10]. Завдяки цьому штучний інтелект краще адаптується до специфічної інженерної термінології, а відсоток синтаксичного браку зводиться до мінімуму [1].

Впровадження великих мовних моделей у процес інженерії вимог дозволяє суттєво оптимізувати проєктування програмного забезпечення. Штучний інтелект бере на себе роль інтелектуального транслятора між природною мовою замовника та строгим кодом візуальних моделей. Це не лише економить ресурси команди, але й допомагає виявляти логічні прогалини в архітектурі ще до початку написання коду. Зрештою, це створює надійне підґрунтя для переходу до повної кодогенерації додатків на основі затверджених текстових специфікацій.

Список використаних джерел:

1. Ray T., Cole A., Pinon Fischer B. F., White A. P., Mavris D. N. Agile Methodology for the Standardization of Engineering Requirements Using Large Language Models. *Systems*. 2023. Vol. 11, No. 7. Art. no. 352. URL: https://doi.org/10.3390/systems11070352 (дата звернення: 16.03.2026).
2. Generative AI in Software Requirements Engineering: Opportunities, Challenges, and Future Directions / G. Boussaidi et al. *HAL open science*. 2024.

URL: https://u-bourgogne.hal.science/LABSOC/hal-04483279v1 (дата звернення: 16.03.2026).

3. INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities. 4th ed. / ed. by D. D. Walden et al. Hoboken, NJ : John Wiley & Sons, 2015. 304 p. URL: https://doi.org/10.1002/9781119287552 (дата звернення: 16.03.2026).
4. Vaswani A., Shazeer N., Parmar N., et al. Attention Is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)*. 2017. Vol. 30. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 16.03.2026).
5. White J., Fu Q., Hays S., et al. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. *arXiv preprint*. 2023. URL: https://arxiv.org/abs/2302.11382 (дата звернення: 16.03.2026).
6. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. IEEE, 2018. 104 p. URL: https://doi.org/10.1109/IEEESTD.2018.8559686 (дата звернення: 16.03.2026).
7. Pohl K. Requirements Engineering: Fundamentals, Principles, and Techniques. Berlin: Springer Publishing, 2010. 814 p. URL: https://doi.org/10.1007/978-3-642-12578-2 (дата звернення: 16.03.2026).
8. Language Models are Few-Shot Learners / T. Brown et al. *Advances in Neural Information Processing Systems*. 2020. Vol. 33. P. 1877–1901. URL: https://arxiv.org/abs/2005.14165 (дата звернення: 16.03.2026).
9. Sparks of Artificial General Intelligence: Early experiments with GPT-4 / S. Bubeck et al. *arXiv preprint*. 2023. URL: https://arxiv.org/abs/2303.12712 (дата звернення: 16.03.2026).
10. Ouyang L., Wu J., Jiang X., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*. 2022. Vol. 35. P. 27730–27744. URL: https://arxiv.org/abs/2203.02155 (дата звернення: 16.03.2026).