# Literature reviews

Roman Reimche
RPTU Kaiserslautern, Faculty of Business and Economics

***Note:*** *This report contains a project documentation and reflection on the portfolio task submitted for the lecture Engineering with Generative AI in WiSe 2024-25. This report is an original work and will be scrutinised for plagiarism and potential LLM use.*

## 1 List of papers for literature review of main task

## 2 Reviews on LLM-Based Multi-Language Systems for End-to-End Software Engineering

### 2.0.1 Introduction

Recently, advancements in the field of natural language language processing using large language models has experienced substantial advancements, allowing application not possible before, also in the area of text to text generation – a model receives textual input and produces output that, on multiple benchmarks is close to human performance for the same tasks. This has allowed multi-agent systems to use LLMs as key component for agent design, allowing agentic systems to emulate humanlike information processing, communication and decision taking. One of the known applications of LLMs is software code generation and multiple researchers have recently studied the topic of emulation of software development process using LLM-based multi-agent systems. Such engineering process transcends code generation task by involving complex activities performed by multiple roles of interconnecting actors: requirements engineering, project architecture, project planning and management, generation of multiple heterogeneous software artifacts and quality assurance. Multiple software project paradigms are used in industry, waterfall and agile beign ones of the most common. Moreover, such process in industry can be adapted per case, omitting some and adding other activities, roles and communication rules. In this review we look at 6 recent papers on the topic of LLM-based multi-agent systems for end-to-end software development, that is, full cycle software development with requirements as input and executable system as output, optionally including human interactions in between. We also review critically these contributions in the field and devise current state-of-the-art and future study possibilities.

In Section 2.0.2, we write on our literature selection process. After that, in Section 2.0.3, we summarize the works and in Section 2.0.4 we draw conclusions and present possible future research topics.

### 2.0.2 Literature selection

Since our multi-agent system is not designed to select papers, but only review them, comparing this review with the one generated by the system should exclude comparing literature selection. Besides, the selection process is already described in the project report, we omit it here.

### 2.0.3 Recent works

ChatDev'24 [1], MetaGPT'24 [2], and CodePori [3] implement different variations of the waterfall paradigm of software engineering process. So, ChatDev'24 splits it's straitforward pipeline into design, code and testing phases where in every phase several different agent roles communicate with each other to produce output for the next phase. The authors also introduce a mechanism of communicative dehallucination to reduce the amount hallucination-induced code deffects – agents in every phase discuss the task to make it more detailed. The system is evaluated using the Software Requirement Description Dataset (SRDD) [**?**] and compared with MetaGPT'23[1] and one other framework, showing that ChatDev'24 surpasses the second best competitor (MetaGPT'23) substantially.

MetaGPT'24 [2] employs 5 phases: requirements analysis, system design, preparation of coding tasks, production of code and quality assurance. The team includes one agent per role. The authors also introduce a quality enhancement mechanism included in coding role. This mechanism allows the agent to take prior execution and message history into account to enhance produced software artifacts. The results are evaluated using HumanEval [**?**] and MBPP [4] and achieves 85,9% and 87,7% in pass@1. Besides that, MetaGPT'24 ist compared with several other frameworks including ChatDev'2023 [**?**] in terms of capabilities, like requirements and design generation, code and API generation and others, showing that MetaGPT'24 has a more sophisticated process than other frameworks.

CodePori [3] presents a software development process that, to some extent, can be seen as unifying some of the ideas from the previous two. So, this framework includes planning phase as MetaGPT'24 and also in-phase communication between phase-related agents (e.g. 2 developers discussing their work or 2 quality assurance engineers) similar to ChatDev'24. Otherwise, the information goes through the stages of planning, code generation and two stages of quality assurance, one of which involves the outlines inter-agent communication and the other employs only one agent trying to fix flaws missed in the previous stage. This work compares it's results with some other frameworks using HumanEval [**?**] benchmark and pass@1 metric. Among the other frameworks, both ChatDev (almost 70% on pass@1) and MetaGPT (slightly above 80%) are present and the comparison shows that CodePori performed better than both of them hitting almost 90%. However, authors don't specify which versions of these systems were involved in comparison. Furthermore, authors used their own handcrafted set of tasks and manually assessed the quality of software artifacts produced by their framework concluding that 85% of the tasks were finished with acceptable quality (programs were executable after generation or after minor manual adjustments in generated code).

AgileCoder [5], together with AgileGen [6], implement agile software process paradigm. AgileCoder performs several sprints, each of which includes four phases: planning, development, testing and review. The team includes 5 roles: product manager and scrum master communicate to manage backlog together, developer generates code, tester writes tests and senior developer provides quality assurance feedback. Further, the authors included a dynamic graph generator component into architecture that manages code dependency graph – a graph of interdependencies between code components. According to authors ablation study, this component improves

[1]The authors use MetaGPT version from 2023 [**?**], whereas we review MetaGPT'24 [**?**]

system outputs. The authors evaluate system outputs using HumanEval [**?**], MBPP [**?**] as well as a novel benchmark of their contribution, ProjectDev. The first two are also used to compare the systems generated by AgileCoder with outputs of other frameworks, including ChatDev'23 and MetaGPT'24. AgileCoder achieves better results than the other two with MetaGPT'24 performing better than ChatDev'23.

AgileGen offers another variant of agile software development process and introduces several innovations:

- memory pool of known use cases to offer them to the user for the goal of requirements refinement;
- scenario design mechanism that uses Gherkin[2] [**?**] programming language to operate use cases in a more structured form and can translate between natural language and Gherkin to allow higher quality of interaction with the user;
- Gherkin used for quality assurance;
- frequent interaction with user for refinement of requirements and acceptance criteria.

The authors compare their results with several other methods, including ChatDev'23 and ChatDev'24 using two benchmarking datasets: 50projects50days [**?**] and SRDD [**?**]. On average, AgileGen performs better, than all of the competitors, whereas other methods show better values in several tasks on at least one of the used metrics.

conclusion:

- though the frameworks allow boilerplating, if they can build an almost production ready system using complex real world requirements is still unclear and cen be studied in future

- cost is almostly neglectable compare to humans but unclear how much manual work is required for fixing flaws left by agents

- humaneval and mbpp are too simple, demand for a more real-world benachmark based comparison

- differences on evaluations of the same systems in different papers may be based on different LLMs and different hardware and network conditions

TODO: remove benchmarking values, because too difficult to include the ones from agilecoder?

#### 2.0.4 Critical review and conclusion

## References


[1] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development, 2024.

[2] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024.


[2] A domain-specific language used to define executable scenarios, e.g. test cases. It both structured and human-readable.

[3] Zeeshan Rasheed, Malik Abdul Sami, Kai-Kristian Kemell, Muhammad Waseem, Mika Saari, Kari Systä, and Pekka Abrahamsson. Codepori: Large-scale system for autonomous software development using multi-agent technology, 2024.

[4] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018.

[5] Minh Huynh Nguyen, Thang Phan Chau, Phong X. Nguyen, and Nghi D. Q. Bui. Agilecoder: Dynamic collaborative agents for software development based on agile methodology, 2024.

[6] Sai Zhang, Zhenchang Xing, Ronghui Guo, Fangzhou Xu, Lei Chen, Zhaoyuan Zhang, Xiaowang Zhang, Zhiyong Feng, and Zhiqiang Zhuang. Empowering agile-based generative software development through human-ai teamwork, 2024.