# Towards Multi-Agentic AI for automated software design and modelling: challenges and opportunities

Hoa Khanh Dam
*Decision System Lab*
*School of Computing and Informaton Technology*
*University of Wollongong, Australia*
hoa@uowmail.edu.au

***Abstract*—Agentic AI has recently emerged as a new paradigm where autonomous AI agents, empowered with Large Language Model capabilities, are capable of perceiving, reasoning and acting independently to pursue goals. In this paper, we explore the key challenges in developing Agentic AI for automated software design and modelling. These challenges include translating natural language requirements into design models, resolving ambiguities, maintaining consistency within and between design models, and managing conflicts and inconsistencies when merging different versions of a design model. To address those challenges, we propose a conceptual Multi-Agentic AI framework in which autonomous, goal-driven AI agents capable of interpreting requirements, generating design artefacts and collaborating on software design models.**

## I. Introduction

Powerful AI technologies such as Generative AI [1] and Large Language Models (LLMs) [2] are a game changer for automating many software engineering tasks such as code generation [3], testing [4], debugging [5], bug detection [6] and automated repairs [7]. Recent work in LLMs integrate them into autonomous agents that can interact, plan, discuss and make decision to solve problems or perform complex tasks together [8]. Software engineering is moving towards an *Agentic* world [9] where a team will consist of many *virtual AI-powered software engineering agents*, some specialising in coding, some in designing, and others in testing. Those agents and their human teammates would work together to continuously design, build and release software. For example, our recent work [10] proposes a multi-agent framework where AI-powered agents coordinate, communicate and discuss with human developers and other agents to reach a consensus on user story estimation.

Recent research has explored the application of LLMs in generating Unified Modelling Language (UML) [11] design models. The work in [12] investigated how ChatGPT was used to generate UML sequence diagrams from natural language requirements. Their study found that the generated models generally adhere to UML standards and are understandable, however completeness and correctness issues are found in many cases where requirements are ambiguous. Another study [13] investigated how LLMs can assist novice analysts in creating UML models (use case, class and sequence diagrams). Their study also found correctness and completeness problems in the generated models. Those findings are similar to those presented in [14] in which a large number of semantic and textual errors were found in LLM-generated UML models.

Those studies indicate a growing interest in leveraging LLMs for software design and modelling. Their findings show both the potential benefits and limitations of integrating LLMs as a tool into the software design process. However, existing work only considers LLMs as passive tool assistants, which lack autonomy and are only invoked on demand by software developers. In this paper, we take a significant step further into an Agentic future where AI agents, empowered by LLMs, will be fully embedded in the software design process and modelling environment. They are goal-driven, operate independently and make their own design decisions. They can perceive their modelling environment and respond to changes (e.g. new model elements are created or modified). In addition, they interact with other AI agents and human stakeholders when developing a design model for a software system.

In this paper, we first identify and examine those key challenges in details (Section II). We then propose a Multi-Agentic AI framework in which AI agents autonomously create, modify and validate design models. They also can coordinate, negotiate, and generate explanations and interact effectively with human stakeholders (Section III). Finally, we outline the next steps for implementing and evaluating this framework (Section IV).

## II. Challenges

### A. Natural language requirements to structured design

Software requirements, such as user stories, acceptance criteria and use case scenarios, are essential input to the software design process. Designing an Agentic AI system capable of autonomously creating UML design models from software requirements presents an important research challenge. First, software requirements are often written in informal, natural language, thus they are inherently ambiguous. For example, let us consider a user story for a video-on-demand software: “As a registered user, I want to play a video from the catalog so that I can watch it instantly on my device”. Terms like “registered user” and “catalog” are vague and open to multiple interpretations, such as which users are included (any user with an account or only those with a paid subscription),

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00063

and what content is accessible (all available videos or only a subset based on region). In addition, user stories often assume domain-specific background which is not explicitly stated. These ambiguities and incompleteness pose significant challenges for translating requirements into software design. AI agents must be capable of understanding and reasoning about incomplete, vague user stories, and infer context (e.g., what "user" refers to in a specific domain).

Second, an AI designer agent needs advanced semantic parsing capabilities and contextual reasoning to extract meaningful design elements such as actors, classes, functions, interfaces, and interactions from those requirements. The agent should interpret textual information and determine how to represent those design elements using appropriate UML models such as class diagrams for structure, use case diagrams for functionality, or sequence diagrams for interactions. This requires the AI designer agent to reason about architectural design choices, identify entities and relationships, and understand system dynamics. These should often be achieved without having all the necessary information explicitly stated. For example, the above user story requires modelling such as content licensing, access rights, and streaming technology, however the user story does not state this. To fill this gap, the agent needs knowledge of the domain and relevant design patterns to guide their decision making.

Finally, AI designer agents should be able to engage ongoing dialogue with human stakeholders to clarify ambiguities, gather missing information, co-design the software and validate their modelling decisions. They should be explain their choices and adapt to human stakeholders' feedback. Hence, there is a need to design interfaces and communication protocols that enable this interactive human-agent collaboration in creating design models.

**Challenge 1: Developing a Multi-Agentic AI for creating software design from requirements involves interpreting ambiguous and incomplete natural language requirements, mapping them to formal design constructs, and enabling explainable, interactive collaboration with human stakeholders.**

### *B. Consistency management in software design and modelling*

Design languages, such as the Unified Modelling Language (UML), must adhere to diverse consistency requirements, some of which are defined as part of the language while others reflect modelling philosophies, domain restrictions, or even application needs. Consistency conditions are usually specified in terms of constraints. Consistency constraints for UML are typically expressed in the standard Object Constraint Language (OCL) [15].

To ensure their design are well-formed and consistent, AI agents should be able to interpret and evaluate OCL constraints on UML models. They should have the capability of parsing OCL expressions, understanding their semantics and reasoning about the constraints. However, OCL constraints are written in a formal declarative language, while LLM-based agents are primarily trained on natural language. Thus, the agent needs to be equipped with symbolic reasoning and integrated with formal verification tools such as an OCL checker.

Another important challenge for an AI designer agent is change propagation: once the agent makes some initial changes to the design, it may need to make additional, secondary, changes to maintain consistency. Changes made to a design model may violate a number of consistency constraints, and resolving those constraint violations would lead to changes to other parts of the design, which may in turn break other constraints, resulting in further changes and so on [16].

**Challenge 2: Multi-Agentic AI needs to understand and reason over formal OCL constraints even though LLM-based agents are trained on natural language. They need to maintain consistency across design changes while managing their cascading effects on the design model.**

### *C. Collaboration in software design and modelling*

Software design and modelling are inherently a collaborative process. Multiple AI agents may work on the same design model at the same time, thus creating different versions of the same design. Hence, Multi-Agentic AI systems require merging versions of software design models. AI agents need to reconcile structural and behavioural differences across potentially conflicting models while preserving consistency and intent. The agents need to preserve consistent merges, report inconsistencies, and explore alternative merging solutions for inconsistencies (attempting to maximise consistency in the merged model while preserving as many changes made by the individual agents as possible). They should be able to detect conflicts and inconsistencies, reason about those and resolve intelligently. They also need to propagate changes appropriately, and avoid introducing new inconsistencies or redundancies. In collaborative settings involving human developers, the agents should also be able to explain their merge decisions and seek clarification when ambiguities arise.

**Challenge 3: Collaboration in software design and modelling poses key challenges for Multi-Agentic AI as the agents need to merge their work, which requires resolving structural and behavioural conflicts, and maintaining consistency in design models.**

## III. A Multi-Agentic AI framework for software design and modelling

In this section, we propose a Multi-Agentic AI approach to address the above challenges. In our framework, the agents autonomously modify, merge and validate design models. They are also capable of negotiating and coordinating changes between multiple contributors to maintain semantic and syntactic consistency between design models. In addition, they

can generate explanations and communicate with human stakeholders.

### A. Agent architecture

Figure 1 shows the architecture of a Design and Modelling agent (hereafter called ***DM-Agent***). Each agent has goals which it aims to achieve. Goals are derived from user stories or other forms of software requirements. Goals can be decomposed into smaller, more manageable subgoals. For example, "Design a feature" is a ***DM-Agent***'s goal, which has several subgoals: "Understand requirements", "Translate to design components", "Generate UML models" and "Validate design".

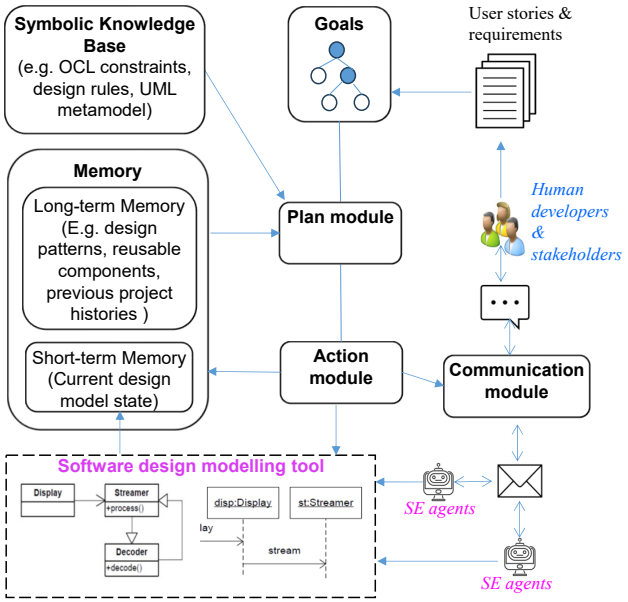


Fig. 1. ***DM-Agent***'s architecture

The goal-driven approach offers robustness and flexibility, e.g., in case of failures, ***DM-Agent*** will try other means to achieve the goal. Each ***DM-Agent*** has a Plan Module which consists of plans for achieving the agent's goals and subgoals. For example, a plan achieving "Understand requirement" would involve parsing the user story and identifying key elements such as actors, objects and actions. Plans consist of external actions or internal actions that are generated by the Action Module. External actions can directly change the design model (e.g., adding or deleting model elements) through a software design modelling tool. Internal actions interact with internal memories such as reading from long-term memory (retrieval), updating short-term working memory (reasoning) and writing to long-term memory (learning).

***DM-Agent***'s memory module is structured into two main layers: short-term memory and long-term memory. Short-term memory captures the current design state such as the UML model elements and diagrams in the current design. Long-term memory stores patterns, reusable components, and histories of previous projects. This memory is useful for ***DM-Agent*** to generate design plans based on proven design solutions and past experiences. In addition, ***DM-Agent***'s symbolic knowledge base includes formal elements such as OCL constraints, design rules, and the UML metamodel. This symbolic layer enables reasoning, validation, and model transformation based on well-defined structural and behavioral semantics. The memory and symbolic knowledge modules allow ***DM-Agent*** to generate consistent and well-formed design models that adhere to current software design best practices.

The agent's communication module is designed to support both inter-agent and human-agent communication, which enables seamless coordination and collaboration across ***DM-Agents*** and human software designers. ***DM-Agents*** communicate with each other in terms of exchanging messages. A message can be in the form of information exchange, or goal delegation (e.g., goal "Validate design" is delegated to a consistency-checking agent), or sharing design plans. The human-agent communication module of ***DM-Agent*** is based on existing LLM capabilities for interpreting user inputs, asking clarifying questions and providing explanations.

### B. ***DM-Agent***'s coordination

We propose a multi-agent coordination framework (see Figure 2) which enables ***DM-Agents*** and human designers independently and concurrently work on the same software design model in a collaborative manner. Each ***DM-Agent*** operates autonomously and focuses on different aspects (e.g., one agent working on playing back feature while the other on recording feature for a video-on-demand system) or components of the overall design (e.g., one agent on class diagrams while the other on sequence diagrams). Each agent may act based on different interpretations, assumptions, or goals (e.g., designing different features), and can explore alternative solutions or work on complementary features simultaneously.

As ***DM-Agents*** and/or human designers modify the same design model (e.g. adding new features, modifying existing features or fixing inconsistencies) based on their specific goals and expertise, they create multiple versions of the design model. To support conflict resolution and consistency preservation, our framework has a *Merger* agent which is responsible for merging those model versions into a unified, consistent and semantically valid design. This agent ensures that the merged model retains consistency and coherence, and accurately reflects the collective contributions of all ***DM-Agents*** and human designers. It serves as a critical mediator in the collaborative workflow.

The *Merger* agent delegates the task of checking for inconsistencies in a design model to the *Consistency Checker* agent. It checks for structural correctness using UML metamodel compliance and for OCL constraint violations. The identified consistencies will be brought to the attention of the *Merger* agent. This agent can also generate plans for fixing an inconsistency, and share them with the *Merger* agent.

Our *Merger* agent merges the ***DM-Agents***' and human designers' versions fully automatically if they are consistent and free of conflicts. The agents or human designers are notified

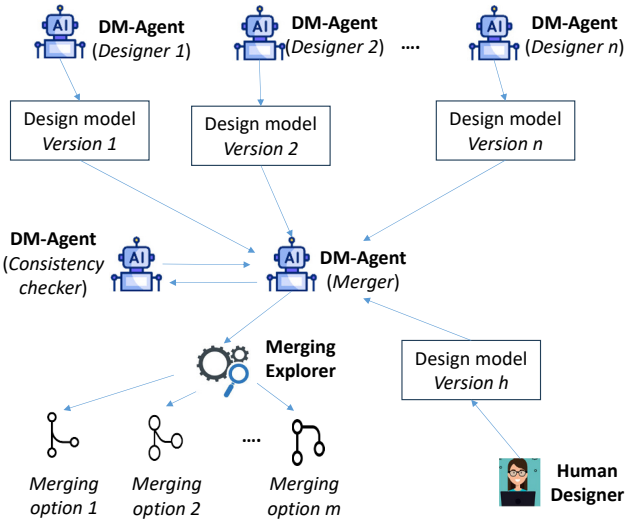


Fig. 2. An example of ***DM-Agent***'s goals and subgoals

only if there are conflicts or inconsistencies. Our *Merger* ***DM-Agent*** employs a fast, automated search technique [17] to quickly gauge whether a compromise is possible to solve the merging problem by taking some (but not all) of the agent's changes. It is useful to automate this initial compromise to avoid bias. However, since merging may involve trade-offs where negotiation and communication are required, this approach provides the ***DM-Agents*** and human designers with all feasible alternative compromises in order to help them discuss and negotiate to reach an informed, consistent merging decisions. To reach agreement, the agents need to engage in a structured negotiation process. There are several multi-agent negotiation protocols that can be employed here such as voting or argumentation.

## IV. The Next Steps

Future work involves implementing the proposed architecture using existing agent frameworks that support modular, goal-oriented, and conversational agents. We also plan to develop a benchmarking dataset for software design and modelling to evaluate the performance of not only our ***DM-Agents*** but also other AI agents provided by the research and industry communities. We will also perform a human user study to investigate the effectiveness of human-agent collaboration in software design and modelling, and the perception of human developers towards autonomous ***DM-Agents*** as team members. Consented participants will join collaborative sessions where they will work with our ***DM-Agents*** to complete a design task. We will then perform a quantitative analysis to assess participants' perceptions of usefulness, usability and trust. We will also conduct a qualitative analysis using thematic coding. This process involves identifying recurring themes, patterns, and insights from participants' responses. By combining both quantitative and qualitative analyses, we aim to gain a holistic view of the impact and acceptance of autonomous AI agents in software design and modelling.

## References


[1] R. Gozalo-Brizuela and E. E. G. Merchan, "A survey of generative ai applications," *Journal of Computer Science*, vol. 20, no. 8, pp. 801–818, May 2024. [Online]. Available: https://thescipub.com/abstract/jcssp.2024.801.818

[2] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023. [Online]. Available: http://arxiv.org/abs/2303.18223

[3] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.

[4] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Transaction on Software Engineering*, vol. 50, no. 4, p. 911–936, Apr. 2024. [Online]. Available: https://doi.org/10.1109/TSE.2024.3368208

[5] L. Zhong, Z. Wang, and J. Shang, "Debug like a human: A large language model debugger via verifying runtime execution step by step," in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 851–870.

[6] M. M. Mohajer, R. Aleithan, N. S. Harzevili, M. Wei, A. B. Belle, H. V. Pham, and S. Wang, "Skipanalyzer: A tool for static code analysis with large language models," in *Proceedings of the ACM on Programming Languages (OOPSLA)*, 2023.

[7] F. Zubair, M. Al-Hitmi, and C. Catal, "The use of large language models for program repair," *Computer Standards Interfaces*, vol. 93, p. 103951, 2025.

[8] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large language model based multi-agents: A survey of progress and challenges," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, K. Larson, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2024, pp. 8048–8057, survey Track. [Online]. Available: https://doi.org/10.24963/ijcai.2024/890

[9] J. He, C. Treude, and D. Lo, "Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, May 2025. [Online]. Available: https://doi.org/10.1145/3712003

[10] L. Bui, H. K. Dam, and R. Hoda, "An llm-based multi-agent framework for agile effort estimation," in *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2025. IEEE, 2025.

[11] Object Management Group, "UML 2.0 Superstructure and Infrastructure Specifications," http://www.omg.org/technology/uml/, 2004.

[12] A. Ferrari, S. Abualhaijal, and C. Arora, "Model generation with llms: From requirements to uml sequence diagrams," in *REW 2024 : Proceedings of the IEEE 32nd International Requirements Engineering Conference Workshops*. Reykjavik, Iceland: IEEE, 2024, pp. 291–300.

[13] B. Wang, C. Wang, P. Liang, B. Li, and C. Zeng, " How LLMs Aid in UML Modeling: An Exploratory Study with Novice Analysts ," in *Proceedings of IEEE International Conference on Software Services Engineering (SSE)*. Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2024, pp. 249–257. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SSE62657.2024.00046

[14] D. De Bari, G. Garaccione, R. Coppola, M. Torchiano, and L. Ardito, "Evaluating large language models in exercises of uml class diagram modeling," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 393–399. [Online]. Available: https://doi.org/10.1145/3674805.3690741

[15] Object Management Group, "Object Constraint Language (OCL) 2.0 Specification," http://www.omg.org/docs/ptc/03-10-14.pdf, 2006.

[16] H. K. Dam and M. Winikoff, "Supporting change propagation in UML models," in *Proceedings of the 26th IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[17] H. K. Dam, A. Egyed, M. Winikoff, A. Reder, and R. E. Lopez-Herrejon, "Consistent merging of model versions," *Journal of Systems and Software*, vol. 112, pp. 137–155, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412121500134X