Vibe Modeling: Challenges and Opportunities

Jordi Cabot[1,2]

[1] Luxembourg Institute of Science and Technology, Esch-sur-Alzette, Luxembourg
`jordi.cabot@list.lu`
[2] University of Luxembourg, Esch-sur-Alzette, Luxembourg

Abstract. There is a pressing need for better development methods and tools to keep up with the growing demand and increasing complexity of new software systems. New types of user interfaces, the need for intelligent components, sustainability concerns, ... bring new challenges that we need to handle.

In the last years, model-driven engineering (MDE) has been key to improving the quality and productivity of software development, but models themselves are becoming increasingly complex to specify and manage. At the same time, we are witnessing the growing popularity of vibe coding approaches that rely on Large Language Models (LLMs) to transform natural language descriptions into running code at the expenses of code vulnerabilities, scalability issues and maintainability concerns.

In this paper, we introduce the concept of *vibe modeling* as a novel approach to integrate the best of both worlds (AI and MDE) to speed up the development of reliable complex systems. We outline the key concepts of vibe modeling and highlight the opportunities and open challenges it presents for the future of modeling.

Keywords: Vibe Modeling · Low-modeling · Low-code · DSL · Artificial Intelligence · Model-driven · Vibe Coding.

1 Introduction

Current software development projects face a growing demand for advanced features, including support for new types of user interfaces (augmented reality, virtual reality, chat and voice interfaces,...), intelligent behavior to be able to classify/predict/recommend information based on user's input or the need to face new security and sustainability concerns, among many other new types of requirements.

To tame this complexity, software engineers typically choose to work at a higher abstraction level [2] where technical details can be ignored, at least during the initial development phases. *Low-code platforms* are the latest incarnation of this trend, promising to accelerate software delivery by dramatically reducing the amount of hand-coding required. Low-code can be seen as a continuation or specific style of other model-based approaches [6, 28], where high-level software models are used to (semi)automatically generate the running software system.

However, even models themselves are becoming more and more complex due to the increasing complexity of the underlying systems being modeled. Beyond "classical" data and behavioral aspects, we now need to come up with new models to define the new types of UIs or all the smart features of the system. Note that AI elements are hard to specify [25], architect, test and verify [26] and low-code systems have so far paid little attention to the modeling and development of smart systems.

This hampers the adoption of model-driven processes as it reduces the (perceived?) return on investment of modeling activities due to the increase cost of modeling[3]. Even more at a time when a growing number of tools offer a vibe coding approach to generate code from natural language descriptions. Vibe coding is an approach to producing software by using Large Language Models (LLMs), where a person describes a problem in a few natural language sentences as a prompt that is then sent to an LLM tuned for coding. The LLM generates software based on the description, shifting the programmer's role from manual coding to guiding, testing, and refining the AI-generated source code [4]. With the new agentic capabilities provided by many IDEs and agent-based systems built on top of those LLMs, even the testing and verification of the generated code is becoming easier, where the agent itself creates tests, runs them and refines the code if it detects any issues. While the generated code is not always correct, the quickness of the process makes it ideal for prototyping, exploratory development and personal projects.

These were some scenarios where, until now, we were claiming low-code and model-driven approaches had a great product-market fit due to the possibility of having an automatic code generation phase. However, now vibe coding appears as a solid competitor here. While the final quality is not the same and vibe coding introduces plenty of safety, scalability and non-determinism issues, it is clear that the vibe coding approach is here to stay and will continue to grow. Indeed, this type of "vibing" approach resonates with plenty of developers and even with non-technical people who see a way to create their own apps.

In this paper, we explore whether we could combine both trends. To do so, we introduce the concept of *vibe modeling* as a novel approach to integrate the best of both worlds (AI/LLMs and MDE) to speed up the development of reliable complex systems. With vibe modeling, you can still start from a natural language description that an agent will help to transform into a set of models as part of a conversation between you and the agent. While at the same time, you will later use deterministic and certified code-generation techniques to generate the final software system from the models safely.

The next sections are organized as follows. Section 2 reviews the state of the art in applying LLMs and AI to modeling activities. Section 3 discusses the concept of vibe modeling and presents a vibe-modeling-centered development process, including its integration with low-code approaches. Next, Section 4 ex-

---

[3] Note that the adoption of modeling practices is a complex sociotechnical problem [18].

[4] https://en.wikipedia.org/wiki/Vibe_coding

tends the core concept to cover collaborative scenarios, while Section 5 comments on the infrastructure required to support vibe modeling. Finally, we discuss open challenges and future directions before concluding the paper.

2 State of the art

The modeling community has embraced with real interest the idea of using LLMs to assist in modeling activities [3, 4, 27] as part of ongoing efforts to reduce the cost of modeling itself and improve its Return on Investment [5].

Many of these works focus on inferring a (partial) model from a natural language description. For instance, research conducted in [8], [15], and [10] evaluated the potential to create domain models from textual descriptions using prompting. Camara et al. used zero-shot prompting to create UML class diagrams with few syntactic errors, however, the worst results were found when the model required abstractions, such as using inheritance instead of attributes or creating association classes. Fill et al. used GPT-4 to create domain models for Entity Relationship diagrams for conceptual modeling, BPMN diagrams for business processes, and Heraklit models for embedded systems by providing one example of the desired output [15]. Chaaben et al [10] experimented with the Few-shot technique using between 2 and 4 examples for the recommendation of concepts for different domain contexts, and to assist with static and dynamic domain modeling. Other approaches tried to go beyond simple prompting techniques and experiment with chain-of-thought [11] and tree-of-thought [29] prompting techniques for better accuracy.

Despite advances, LLM-based domain modeling solutions still face noticeable limitations. For example, in UML class diagram modeling, accurately identifying relationships among classes remains challenging [12, 29, 31].

In line with this, research such as [20] proposed a human-in-the-loop (HITL) approach to LLM-based modeling, aiming to exploit user feedback to refine and eventually enhance the quality of domain models created by LLMs. Moreover, this work was focused on process models and assumes the user is a modeling expert.

In this paper, we aim to generalize and contextualize these partial approaches in a fully interactive model-driven workflow, leveraging current agentic capabilities to better integrate AI advances in modeling before generating the code with rule-based deterministic code-generators. To the best of our knowledge, ours is the first work that focuses on exploring this integration.

3 A vibe-modeling-centered development process

We define *vibe modeling* as the process of building software through conversational interaction with an LLM tuned for modeling, not coding. Once the models are created, a standard model-based / low-code approach can be used to generate deterministic code from those "vibed models". Think of vibe modeling as a model-driven vibe coding approach.

Indeed, in vibe modeling, the LLM does not aim to generate code, but rather to produce models. Then, the model-to-code step is performed with "classical" rule-based code-generation templates (or any other type of precise and semantically equivalent executable modeling [22] techniques).

As such, vibe modeling has two major advantages over vibe coding:

– **Understandable output**. A user is able to validate the quality of the LLM output (the models), even without coding expertise. Models are more abstract and closer to the domain and, therefore, a user should be able to understand them with limited effort. True, some basic modeling knowledge may still be required, but for sure it's much easier to validate a model than a list of lines of code.
– **Reliable code-generation**. The generation process is deterministic. If the model is good, we know the code will be good (assuming a certain level of quality in the code-generation templates but that should only be verified once and for all) and there is no need to check it every time we regenerate the system, saving plenty of time.

Combining the two, we see that, in contrast to vibe coding, vibe modeling can be useful to both technical and non-technical experts. Even the latter will have better chances to generate a reliable software by validating the models while, in a vibe coding approach, due to their lack of coding expertise, there would have no other option but to blindly trust the LLM.

Figure 1 illustrates the process in more detail. The user (the domain expert) starts by describing the system to the agent (internally relying on a LLM). Based on this conversation (and any other material provided by the user such as interviews, guidelines,...) the agent will propose a model. The model itself is uncertain, so the agent and the user can start a conversation to clarify and refine it. If a modeling expert is available and the complexity or criticality of the domain is high, the modeler can either discuss with the user to clarify ambiguities or incompleteness aspects of the description (that could then be used by the agent to improve the model) or directly refine the model. In both cases, the modeler's role is to reduce the uncertainty on the quality of the model. The iteration continues until the model is considered good enough. At that point, the deterministic part of the process triggers in and the code-generation templates produce the final software system, including the code itself, the database (if needed), the deployment scripts (if needed), etc.

4   Collaborative vibe modeling

A logical evolution of our proposal is vibe modeling with, not one, but a full community of agents collaborating in real-time as the final result outperforms single-agent solutions [16]. This is starting to be explored for a variety of software engineering tasks [17]. In the specific case of modeling, this would imply to work with a community of modeling agents, each one potentially specialized in a different domain, type of model, modeling phase, etc. Non-functional factors
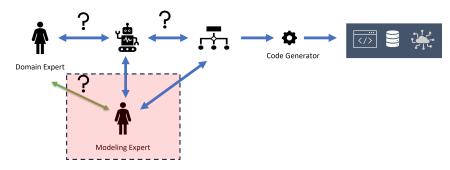
Fig. 1: The vibe modeling process as part of a low-code architecture. The question mark shows interactions that carry uncertainty

(like the cost, availability, sustainability or linguistic capabilities) of the agents should also play a role in the selection of the agents to be used in the collaborative modeling process.

How many agents should be involved, depending on the complexity of the model to be inferred, how much should they be specialized and whether we should put in place a purely collaborative or also a competitive approach are still open questions. Ideally, and assuming they can afford the cost and accept the sustainability concerns, the more the merrier as the diversity of solutions proposed by the agents should lead to a better final model. Indeed, a purely collaborative approach has too many single points of failure, as the success of each modeling task relies on the output of a single agent. Instead, a competitive approach, where more than one agent is assigned the same task, is more robust but it requires putting in place a mechanism to select the best solution from the different proposals.

This "mechanism" could simply be a human in the loop. But this would require too much work from the user and may not even be feasible when the user is not a technical expert able to judge by himself the quality of the solutions. An infrastructure like the one proposed in the Mosaico EU project [5], involving different types of agents (solution, supervision and consensus) could help here. In this example (see Figure 2) the solution agents are responsible for generating a (part of the) model, the supervision agents would then score each proposal stating which one they think it is best while the consensus agent would use these evaluations to choose the final solution. This choice could be based on a governance policy defined at the beginning of the modeling project, where we could state whether the consensus agent should simply take the one with more votes in favor or try to force more of a common agreement between the supervision agents.

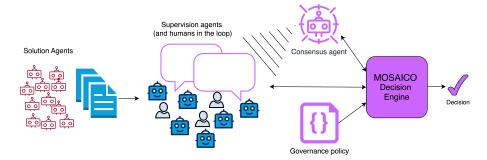[5] https://mosaico-project.eu/

Fig. 2: Overview of the AI Agent community in the Mosaico EU project

5   Vibe modeling infrastructure

There is a key infrastructure element to enable vibe modeling: an easy way for agents to create and manipulate models. We can (and should) train specialized agents to become great modelers but the agents themselves should not embed the modeling stack. Same as human modelers. We do not have a modeling tech stack, such as the Eclipse Modeling Framework [6], inside us. Instead, we work with Eclipse tools that expose core modeling services through different interfaces built on top of the core Eclipse components when we need to create models, view them, generate code from them, etc.

The same applies to agents. We do not want to reimplement the modeling stack as part of each agent code. Agents should be able to communicate with the modeling platform/s we want to use in our development project and benefit from the tool capabilities (e.g. to perform model validation, model rendering and many other basic model manipulation operations that are common to most modeling scenarios). But implementing a direct bridge between each agent and each modeling platform quickly triggers the MxN integration problem [7].

The Model Context Protocol (MCP) [8] is a popular open protocol that standardizes how applications provide context to agents. Therefore, We propose to use MCP to bridge the gap between our modeling agents and the modeling platforms. As seen in Figure 3, the agent embeds an MCP client able to communicate with any MCP server, in particular, the one implemented by the modeling platform, exposing the modeling services to the agent. Once a modeling platform offers an MCP server, any agent can use it to chat with it, and the platform does not need to adapt to the type of agent or the LLM used by such agent.

---

[6] https://www.eclipse.org/modeling/emf/

[7] The MxN integration problem refers to the challenge of connecting M different AI applications to N different external tools without a standardized approach https://huggingface.co/learn/mcp-course/en/unit1/key-concepts. It is a recurrent problem in information systems development that also appears in the context of vibe modeling.

[8] https://modelcontextprotocol.io/

Similarly, an agent embedding an MCP client can automatically discover and use any MCP server tools available in the environment without having to learn and implement code to interact with the internal modeling platform API (see Figure 4). This allows for scenarios where agents could even use, at every step of the collaboration, a different modeling platform specialized on the type of modeling request they are working on.
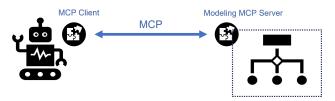

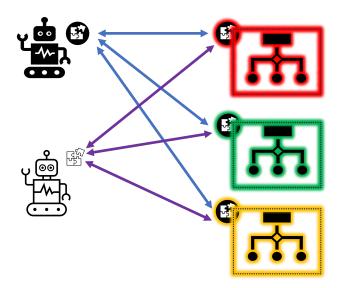Fig. 3: MCP for modeling interactions

Fig. 4: Multi-agent and multi-tool collaboration facilitated by the use of MCP

As a proof of concept, we are implementing an MCP Server for the BESSER low-code platform [1]. Thanks to this MCP server, any agent, the agent mode of Cursor[9] in this case, can discover and use the modeling services offered by

---

[9] Agent is the default and most autonomous mode in Cursor, designed to handle complex coding tasks with minimal guidance `https://docs.cursor.com/chat/agent`

BESSER when a user (or any other agent) requests a task for which one of the BESSER services exposed in its MCP server would be a good fit (see Figure 5).

Fig. 5: Example of creating a new model via a service exposed by the BESSER MCP Server

The actual MCP Server implementation is mostly a thin wrapper on top of the internal tool API where MCP standardizes the way the tool (in a MCP context, *tool* refers to a service the agent can use as a tool to achieve something, so each modeling service would be exposed as an MCP tool) is described (and later discovered and called) by the agent. Listing 1.1 shows a simple example of the MCP Server of BESSER exposing the creation of a new B-UML model. Note that the model is returned serialized. This enables the agent to keep and use the model in a future interaction if needed. An obvious alternative would be to store the model in a database and return the id of the model. Each approach has different trade-offs (e.g., the need to configure a database and make sure the agent has access to it) and, thus, the best approach depends on the modeling scenario.

```
1  @mcp.tool()
2  async def new_model(name: str) -> str:
3      """Creates a new B-UML DomainModel with the specified name and
       ↪ returns it as base64.
4
5      Args:
6          name (str): Name of the new domain model.
7
8      Returns:
9          str: A new domain model instance as base64 string.
10     """
11     try:
12         from besser.BUML.metamodel.structural import DomainModel  #
       ↪ type: ignore
13     except ImportError as exc:
14         raise RuntimeError(
15             "BESSER library must be installed ('pip install besser')."
16         ) from exc
17
18     # Create and return a new DomainModel instance as base64
```

```
19    domain_model = DomainModel(name=name)
20    return serialize_domain_model(domain_model)
```

<div align="center">Listing 1.1: BESSER MCP Server example</div>

6 Roadmap

So far, we have presented the key definitions and infrastructure to put in place a vibe modeling approach. Nevertheless, there are still a number of challenges to address to make this approach more effective and more widely adopted. In what follows, we discuss some open challenges.

6.1 Specialized modeling agents

While there are a number of works on inferring models from natural language descriptions (see Section 2), most are one-shot approaches. The advent of agents and agentic workflows opens the door to interactive vibe modeling approaches like the one discussed here. But we still need to learn how to best leverage agentic capabilities and collaborate with one (or more) agent(s)to infer better models. Aspects like:

– What types of conversations and questions the agents should have with the domain experts to validate the model being inferred? Or to disambiguate and complete the natural language description?
– How to train agents for the modeling domain? What type of Reinforcement learning strategies could be useful to create specialized modeling agents?
– What datasets should be created and provided to the LLMs used by the agents to improve their training (e.g. extending [21])? Or to have fine-tuned LLMs with a better understanding of the modeling concepts and tasks?
– How these agents can effectively collaborate on partial models to complement and improve their own suggestions? How many agents should be involved, depending on the complexity of the model to be inferred?
– How to evaluate the quality of the models inferred by the agents? And how to use that information to choose the best modeling agent for the task at hand?

still need to be addressed.

6.2 Vibe modeling of other types of models, especially AI models

Most of the current works evaluate the performance of LLMs on specific types of models, mainly class diagrams and workflow models. But it is unclear how good are LLMs (and the agents built on top of them) when it comes to inferring other types of diagrams, such as (UML) collaboration diagrams, architectural diagrams, user interface models,. . . . These types of models are less popular and

therefore less present in the data the LLMs have been trained on. We need more research to understand the limitations of LLMS when it comes to this kind of models and how the aspects mentioned in the previous point would need to be adapted to cover these types of diagrams.

The situation is even worse when it comes to the modeling of the AI components integrated in a software system, also known as a smart software system [7]. These systems require new types of models. For instance, chatbot models [9,24] for chatbot interfaces, biases requirements for the AI components [13,23], modeling of neural networks [14], etc. The lack of standards for this type of models is an additional challenge when teaching agents how to infer them.

6.3   Uncertainty and traceability

Uncertainty modeling [30] should be considered a first-level concern. Indeed, when agents and LLMs are part of the modeling process, all model proposals come with a certain level of uncertainty. This confidence score should be stored together with the element. And for the same reason, we must be able to explain where that number came from. We need to keep full traceability of the model evolution. We should be able to explain who proposed and approved each model change. Similar to what was proposed in Collaboro [19] but applied at the model level, you could link to the model elements the list of change proposals for that element, the user (human or agent) behind the change and the full list of users that voted in favor of the change when we are in a collaborative scenario.

6.4   Adapting vibe modeling to different user profiles

Vibe modeling could be used by different types of users, from domain experts, with limited technical expertise, to software engineers with deep modeling expertise but limited domain knowledge for the domain targeted by the system-to-be. Each profile may prefer a different type of interaction with the modeling agent/s. In the former, the agent should be able to explain the model in a way that is easy to understand for the domain expert. In the latter, the agent should offer a more direct approach where the user can directly validate model excerpts and where the agent may be more useful as a domain expert answering domain questions from the modeler based on its internal knowledge.

This is similar to the no-code/low-code discussion, where we also have these two types of profiles, and platforms end up offering a combination of both as they are not mutually exclusive. For instance, as depicted in our vibe modeling approach, we could have a non-technical user interacting with the agent with the occasional participation of a modeling expert for more complex modeling decisions, depending on the criticality of the domain.

6.5   Native integration of agents in low-code platforms

To provide a more natural interaction flow, we advocate for the integration of MCP clients in low-code platforms. This would enable the modeler to seamlessly

switch between a "traditional" modeling flow and a vibe modeling one. Having a chatbot widget in the modeling UI would facilitate the adoption of vibe modeling as the user would not feel a disruption when interacting with the embedded agent.

At the moment, very few low-code platforms offer some type of AI assisted modeling support [10], even if we can foresee that more and more tools will be quickly adding this type of capability.

Nevertheless, we expect most of these vendors to release proprietary solutions, with agents directly integrated with the internal tool vendor APIs and focused on the specific types of modeling languages provided by the tool. We hope the popularization of MCP servers, as the one proposed herein, will help to prevent this situation, at least for open source solutions that may be more open to favor modeling interoperability.

Note that, in our vision, the widget embedded in the tool should interact with the MCP server and not directly with the tool itself. We aim for a collaborative scenario where the agent acts as yet another user collaborating with the modelers in real-time. This offers more flexibility and portability as the same agent can be used from within the low-code platform but also from inside other tools (e.g. Cursor or any other IDE) that support the MCP protocol, combining the best of both worlds.

7 Conclusions and further work

This paper has introduced the concept of vibe modeling and how it can enable a new style of low-code software development combining the benefits of AI and model-driven techniques. While this new development approach has still many shortcomings, we believe it shows promise and could contribute to reestablish the importance of (conceptual) modeling in front of current trends favoring direct "vibe coding" of the applications with all the risks this implies for the quality of the final system.

As further work, we plan to address the roadmap outlined above to facilitate the adoption of vibe modeling and continue refining these ideas based on the feedback of the vibe modelers.

References

1. Alfonso, I., Conrardy, A.D., Sulejmani, A., Nirumand, A., Haq, F.U., Gomez-Vazquez, M., Sottet, J., Cabot, J.: Building BESSER: an open-source low-code platform. In: van der Aa, H., Bork, D., Schmidt, R., Sturm, A. (eds.) Enterprise, Business-Process and Information Systems Modeling - 25th International Conference, BPMDS 2024, and 29th International Conference, EMMSAD 2024, Limassol, Cyprus, June 3-4, 2024, Proceedings. Lecture Notes in Business Information

---

[10] Two exceptions are Mendix, which has recently introduced Maia `https://www.mendix.com/platform/ai/aiad/`, and OutSystems with Mentor `https://www.outsystems.com/low-code-platform/mentor-ai-app-generation/`

Processing, vol. 511, pp. 203–212. Springer (2024). https://doi.org/10.1007/978-3-031-61007-3_16

2. Booch, G.: The history of software engineering. IEEE Softw. **35**(5), 108–114 (2018). https://doi.org/10.1109/MS.2018.3571234

3. Burgueño, L., Cabot, J., Wimmer, M., Zschaler, S.: Guest editorial to the theme section on ai-enhanced model-driven engineering. Softw. Syst. Model. **21**(3), 963–965 (2022). https://doi.org/10.1007/s10270-022-00988-0

4. Burgueño, L., Ruscio, D.D., Sahraoui, H.A., Wimmer, M.: The past, present, and future of automation in model-driven engineering. CoRR **abs/2405.18539** (2024). https://doi.org/10.48550/ARXIV.2405.18539

5. Cabot, J.: Low-modeling of software systems. In: Fill, H., Mayo, F.J.D., van Sinderen, M., Maciaszek, L.A. (eds.) Software Technologies - 18th International Conference, ICSOFT 2023, Rome, Italy, July 10-12, 2023, Revised Selected Papers. Communications in, vol. 2104, pp. 19–28. Springer (2023). https://doi.org/10.1007/978-3-031-61753-9_2

6. Cabot, J.: The low-code handbook: Learn how to unlock faster and better software development with low-code solutions. Self-published, 1st edn. (October 2024), `https://lowcode-book.com/`

7. Cabot, J., Clarisó, R.: Low code for smart software development. IEEE Softw. **40**(1), 89–93 (2023). https://doi.org/10.1109/MS.2022.3211352

8. Cámara, J., Troya, J., Burgueño, L., Vallecillo, A.: On the assessment of generative AI in modeling tasks: an experience report with chatgpt and UML. Softw. Syst. Model. **22**(3), 781–793 (2023). https://doi.org/10.1007/s10270-023-01105-5

9. Cañizares, P.C., López-Morales, J.M., Pérez-Soler, S., Guerra, E., de Lara, J.: Measuring and clustering heterogeneous chatbot designs. ACM Trans. Softw. Eng. Methodol. **33**(4), 90:1–90:43 (2024). https://doi.org/10.1145/3637228

10. Chaaben, M.B., Burgueño, L., Sahraoui, H.: Towards using Few-Shot Prompt Learning for Automating Model Completion. In: 2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER). pp. 7–12. IEEE, Melbourne, Australia (May 2023). https://doi.org/10.1109/ICSE-NIER58687.2023.00008

11. Chen, K., Yang, Y., Chen, B., Hernández López, J.A., Mussbacher, G., Varró, D.: Automated Domain Modeling with Large Language Models: A Comparative Study. In: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS). pp. 162–172. IEEE, Västerås, Sweden (Oct 2023). https://doi.org/10.1109/MODELS58315.2023.00037

12. Chen, R., Shen, J., He, X.: A model is not built by a single prompt: Llm-based domain modeling with question decomposition. arXiv preprint arXiv:2410.09854 (2024)

13. d'Aloisio, G., Sipio, C.D., Marco, A.D., Ruscio, D.D.: How fair are we? from conceptualization to automated assessment of fairness definitions. CoRR **abs/2404.09919** (2024). https://doi.org/10.48550/ARXIV.2404.09919

14. Daoudi, N., Alfonso, I., Cabot, J.: Modelling neural network models. In: Grabis, J., Vos, T.E.J., Escalona, M.J., Pastor, O. (eds.) Research Challenges in Information Science - 19th International Conference, RCIS 2025, Seville, Spain, May 20-23, 2025, Proceedings, Part II. Lecture Notes in Business Information Processing, vol. 548, pp. 130–139. Springer (2025). https://doi.org/10.1007/978-3-031-92471-2_10

15. Fill, H.G., Fettke, P., Köpke, J.: Conceptual Modeling and Large Language Models: Impressions From First Experiments With ChatGPT. Enterprise Modelling

and Information Systems Architectures (EMISAJ) pp. 3:1–15 Pages (Apr 2023). https://doi.org/10.18417/EMISA.18.3, artwork Size: 3:1-15 Pages Publisher: Enterprise Modelling and Information Systems Architectures (EMISAJ)

16. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., Zhang, X.: Large language model based multi-agents: A survey of progress and challenges. arXiv preprint arXiv:2402.01680 (2024)

17. He, J., Treude, C., Lo, D.: Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. ACM Trans. Softw. Eng. Methodol. **34**(5) (May 2025). https://doi.org/10.1145/3712003

18. Hutchinson, J.E., Whittle, J., Rouncefield, M.: Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. Sci. Comput. Program. **89**, 144–161 (2014). https://doi.org/10.1016/j.scico.2013.03.017

19. Izquierdo, J.L.C., Cabot, J.: Collaboro: a collaborative (meta) modeling tool. PeerJ Computer Science **2**, e84 (2016)

20. Kourani, H., Berti, A., Schuster, D., van der Aalst, W.M.P.: Process modeling with large language models. In: van der Aa, H., Bork, D., Schmidt, R., Sturm, A. (eds.) Enterprise, Business-Process and Information Systems Modeling. pp. 229–244. Springer Nature Switzerland, Cham (2024)

21. López, J.A.H., Izquierdo, J.L.C., Cuadrado, J.S.: Modelset: a dataset for machine learning in model-driven engineering. Softw. Syst. Model. **21**(3), 967–986 (2022). https://doi.org/10.1007/s10270-021-00929-3

22. Mellor, S.J., Balcer, M.J.: Executable UML: a foundation for model-driven architecture. Addison-Wesley Professional (2002)

23. Morales, S., Clarisó, R., Cabot, J.: A DSL for testing llms for fairness and bias. In: Egyed, A., Wimmer, M., Chechik, M., Combemale, B. (eds.) Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS 2024, Linz, Austria, September 22-27, 2024. pp. 203–213. ACM (2024). https://doi.org/10.1145/3640310.3674093

24. Planas, E., Daniel, G., Brambilla, M., Cabot, J.: Towards a model-driven approach for multiexperience AI-based user interfaces. Softw. Syst. Model. **20**(4), 997–1009 (2021)

25. Rahimi, M., Guo, J.L., Kokaly, S., Chechik, M.: Toward requirements specification for machine-learned components. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW). pp. 241–244 (2019). https://doi.org/10.1109/REW.2019.00049

26. Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., Tonella, P.: Testing machine learning based systems: a systematic mapping. Empir. Softw. Eng. **25**(6), 5193–5254 (2020). https://doi.org/10.1007/s10664-020-09881-0

27. Rocco, J.D., Ruscio, D.D., Sipio, C.D., Nguyen, P.T., Rubei, R.: On the use of large language models in model-driven engineering. Softw. Syst. Model. **24**(3), 923–948 (2025). https://doi.org/10.1007/S10270-025-01263-8

28. Ruscio, D.D., Kolovos, D.S., de Lara, J., Pierantonio, A., Tisi, M., Wimmer, M.: Low-code development and model-driven engineering: Two sides of the same coin? Softw. Syst. Model. **21**(2), 437–446 (2022). https://doi.org/10.1007/S10270-021-00970-2

29. Silva, J., Ma, Q., Cabot, J., Kelsen, P., Proper, H.A.: Application of the tree-of-thoughts framework to llm-enabled domain modeling. In: Maass, W., Han, H., Yasar, H., Multari, N.J. (eds.) Conceptual Modeling - 43rd International Conference, ER 2024, Pittsburgh, PA, USA, October 28-31, 2024, Proceedings.

Lecture Notes in Computer Science, vol. 15238, pp. 94–111. Springer (2024). https://doi.org/10.1007/978-3-031-75872-0_6

30. Troya, J., Moreno, N., Bertoa, M.F., Vallecillo, A.: Uncertainty representation in software models: a survey. Softw. Syst. Model. **20**(4), 1183–1213 (2021). https://doi.org/10.1007/s10270-020-00842-1

31. Wang, B., Wang, C., Liang, P., Li, B., Zeng, C.: How llms aid in uml modeling: an exploratory study with novice analysts. In: 2024 IEEE International Conference on Software Services Engineering (SSE). pp. 249–257. IEEE (2024)