Bridging agility and automation: enhancing model-driven engineering with LLMs in scrum

Leila Samimi-Dehkordi[1,†], Shekoufeh Kolahdouz Rahimi[2,†]

[1]*Department of Computer Engineering, Faculty of Technology and Engineering, Shahrekord University, Iran*
[2]*School of Arts, University of Roehampton, London, United Kingdom*

Abstract

By concentrating on high-level models that simplify design and automate coding, Model-Driven Engineering (MDE) assists developers in tackling challenging software projects. However, in projects that are dynamic and fast-paced, its rigid procedures may seem cumbersome. Scrum and other agile methods provide the necessary flexibility, but integrating them with MDE can be challenging because modeling frequently requires labor-intensive manual work. To make MDE more agile within Scrum, we suggest utilizing Large Language Models (LLMs), which allows these intelligent tools to take care of tasks like building models or checking their accuracy. Through a case study on a ticket reservation system, we show how LLMs can speed up modeling and make sprints more adaptable. While hurdles like ensuring LLMs get domain details right or syncing them with tools like Papyrus remain, our approach paves the way for combining precision of MDE with responsiveness of Agile, with plans to test it on larger, more complex systems.

Keywords

Model-Driven Engineering, Agile Development, Scrum, Large Language Models, Automation

1. Introduction

Model-Driven Engineering (MDE) has changed software development by enabling developers to create high-level models to simplify complex designs and automate coding tasks [1]. By focusing on models instead of raw code, MDE boosts productivity and makes systems easier to maintain. However, in fast-moving projects where requirements shift quickly, MDE's structured approach can feel like a roadblock, as its tools and workflows often lack the flexibility needed [2]. Modern software development thrives on quick iterations and adaptability—qualities that Agile methods like Scrum deliver through short cycles, teamwork, and constant feedback [3]. The problem is that it can be difficult to combine MDE's heavier processes with Scrum's nimble approach because most MDE tools are not designed for Agile's fast-paced, adaptable vibe [4].

Large Language Models (LLMs) are AI tools that have the potential to revolutionize the way we connect Agile and MDE. LLMs succeed at converting unstructured inputs, such as user requirements, into structured outputs, such as models, thanks to their extensive training on large datasets [5]. Because of this, they are ideal for automating MDE tasks, such as creating a UML diagram from a user story or checking the consistency of a design, while keeping up with Agile's demands for speed and flexibility [6]. Scrum seems like a perfect fit for MDE's methodical model refinement because of its distinct roles, such as Product Owner, and short, iterative sprints. Surprisingly, however, little research has examined how LLMs can be incorporated into a Scrum-MDE workflow to increase project agility in real-world projects.

By including LLMs in the Scrum mix, we are putting forth a novel approach to combining the precision of MDE with the speed of Agile. Imagine LLMs as smart collaborators who accelerate the development and refinement of models while preserving the lightweight, iterative nature of Scrum. An LLM could,

Joint Proceedings of the STAF 2025 Workshops: OCL, OOPSLE, LLM4SE, ICMM, AgileMDE, AI4DPS, and TTC. Koblenz, Germany, June 10–13, 2025.

[†]These authors contributed equally.

samimi@sku.ac.ir (L. Samimi-Dehkordi); Shekoufeh.Rahimi@roehampton.ac.uk (S. K. Rahimi)
http://github.com/leilasamimi/ (L. Samimi-Dehkordi);
https://pure.roehampton.ac.uk/portal/en/persons/shekoufeh-kolahdouz-rahimi (S. K. Rahimi)
0000-0002-2842-0256 (L. Samimi-Dehkordi); 0000-0002-0566-5429 (S. K. Rahimi)

for instance, take a User Story like "reserve a ticket" and produce a draft UML diagram during Sprint Planning, facilitating the team to start working without having to labor through manual modeling. In order to save time and increase adaptability, LLMs can contribute to the sprint by writing documents, validating models, or even creating code snippets. By hooking LLMs up with tools like Papyrus [7], we observe a seamless workflow that enables Agile teams to produce excellent models and systems with less effort.

In order to make MDE more agile and user-friendly, this paper aims to first outline a practical method for integrating LLMs into Scrum. Secondly, it uses a case study based on a real-world setup, such as a ticket reservation system, to demonstrate how it works. We're advocating for more automation in MDE, less modeling, and real-world application in business, offering an innovative viewpoint on how AI can reshape model-driven development. The paper is organized as follows: Section 2 reviews related work; Section 3 details our approach; Section 4 presents a case study; and Section 5 concludes with challenges and future work.

2. Background and Related Work

By simplifying complex designs and producing code from high-level models, MDE expedites software development [8]. Its structured procedures, however, can come across as inflexible, which hinders adoption in projects that require rapid adaptation [9]. Although lightweight modeling is the goal of attempts to integrate MDE with Agile practices, such as Scrum [10, 11], tools like Papyrus frequently still require a large amount of manual labor [7]. Given their ability to generate models from text descriptions, including user requirements, LLMs exhibit promise in this regard [12]. However, their contribution to Agile MDE is still mainly unexplored, especially for safety-critical systems where accuracy is essential [13]. The absence of tools that automate modeling while maintaining model accuracy and keeping up with Scrum's short sprints is a significant obstacle. The Agile methodology focus on responsiveness and speed contrasts with traditional MDE toolchains, which mainly rely on manual updates. By automating model creation and adapting to changing needs, LLMs could close this gap and open the door for our methodology.

3. Proposed Approach: Integrating LLMs with Scrum for Agile MDE

Our approach brings LLMs into Scrum to enhance MDE, making it more agile and efficient. LLMs automate essential tasks like creating models, checking their accuracy, and producing code, aligning smoothly with Scrum's iterative cycles [10]. During a sprint, the Product Owner crafts a User Story, such as *"As a user, I want to reserve a ticket"*, which the LLM transforms into a UML model, like a Use Case Diagram with a `User` actor and `Reserve Ticket` use case. To ensure precision, we use prompt engineering, incorporating domain-specific examples—for example, *"a ticket reservation system with a user reserving and canceling tickets"*—to guide the LLM toward accurate models. As the sprint unfolds, the team refines the model, while the LLM checks consistency, such as suggesting a `Payment` class to complete the design, and generates initial code. These targeted prompts reduce errors, like omitting key elements such as a `price` attribute. At the Sprint Review, the LLM quickly incorporates feedback, such as adding a `Cancel Reservation` use case, enabling rapid iterations for small to medium-sized projects.

The architecture, depicted in Figure 1, integrates an LLM (such as GPT) with an MDE toolchain like Papyrus [7] to streamline Scrum sprints. User Stories feed into the LLM, which generates UML models and initial code snippets, such as Python functions for a ticket reservation system. Papyrus processes these models, exported in XMI format, allowing the team to edit and validate them using its support for incremental updates that align with Scrum's iterative flow. The Development Team reviews the LLM's outputs, adjusts models as needed, and provides feedback to refine future iterations. This cycle, LLM generating models and code, Papyrus handling model processing, and the team offering feedback, creates an efficient, flexible workflow. To address integration challenges, like ensuring XMI compatibility with Papyrus or managing API delays during real-time sprint tasks, we use tailored prompts with detailed instructions, such as "Generate a UML Class Diagram in XMI format for a *ticket reservation system with a user reserving and canceling tickets*, including a `Ticket` class with `price`." We are also exploring custom plugins to enhance seamless interaction. This approach reduces modeling time, boosts adaptability, and supports lightweight MDE, letting teams prioritize high-level design decisions [11].

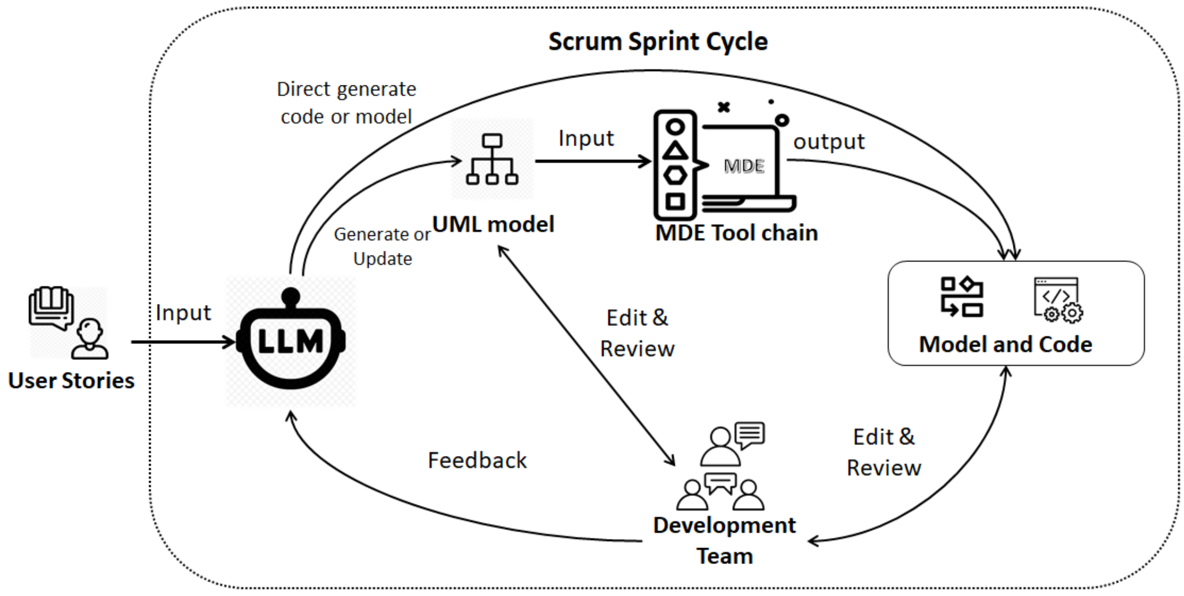

Figure 1: Proposed architecture for integrating LLMs with an MDE toolchain in Scrum.

4. Case Study: Applying LLMs in a Scrum-Based MDE Project

To test our approach, we conducted a case study on a ticket reservation system, built over two Scrum sprints using Papyrus [7] and an LLM. The project centers on a team developing a web-based system, guided by User Stories such as *"As a user, I want to reserve a ticket"* and *"As a user, I want to cancel my reservation."*

- **Sprint 1:** In the first sprint, the LLM creates a Use Case Diagram based on the User Story *"As a user, I want to reserve a ticket"*, featuring a `User` actor connected to a `Reserve Ticket` use case, as shown in Figure 2. We prompted the LLM with: "Generate a Use Case Diagram for a *ticket reservation system with a user reserving and canceling tickets*," producing the initial diagram. The team then refines this into a Class Diagram, defining `Ticket` (with attributes `ticketID`, `status`, `price`) and `Reservation` (with `userID`, `date`), linked by a one-to-many association and constrained by `status` being `active` or `canceled`. This model, with 3 classes and 2 associations, works well for small systems but highlights the need to test scalability in larger projects. The LLM also produces initial Python code, such as `def reserve(self): self.status = "active"` for `Reservation`, with the team verifying outputs to catch errors like missing domain-specific rules (e.g., pricing logic).
- **Sprint 2:** For the second User Story, *"As a user, I want to cancel my reservation"*, the LLM updates the Use Case Diagram by adding a `Cancel Reservation` use case linked to `User`, as shown in Figure 2. The team enhances the Class Diagram by adding a `cancel()` method to `Reservation`, and the LLM generates updated code, including `def cancel(self): self.status = "canceled"`. To ensure accuracy, we used targeted prompts and team reviews to avoid errors, such as omitting constraints. Stakeholders suggested a `Confirm Cancellation` use case, which we noted for Sprint 3.

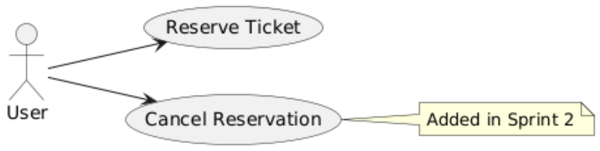

Figure 2: Use Case Diagram for the ticket reservation system, showing the initial model (Sprint 1) and the updated model with `Cancel Reservation` (Sprint 2).

Table 1 presents the results, showing that the LLM cut modeling time from 4 hours to 1 hour and reduced errors from 3 to 2, though it still needed team oversight for domain-specific details, such as pricing logic. The same team performed both manual and LLM-assisted tasks, ensuring a fair comparison by controlling for skill differences. Compared to traditional MDE, the LLM approach sped up iterations by automating model updates, making sprints more efficient. However, success hinges on clear User Stories—vague inputs often produced incomplete models, like missing constraints, which the team had to fix. While the ticket reservation system proves the method's value for small projects, scaling it to larger, more complex systems, such as those with intricate regulatory requirements, requires further testing.

Table 1
Comparison of manual and LLM-assisted modeling.

Metric	Manual	LLM-Assisted
Initial Modeling Time	4 hours	1 hour
Errors Detected	3	2
Code Generation Time	2 hours	30 minutes

5. Discussion and Conclusion

Our approach strengthens Agile MDE by automating model creation and validation, cutting down effort and syncing smoothly with Scrum's iterative cycles [10]. The case study on the ticket reservation system demonstrates faster modeling—slashing time from hours to minutes—and greater flexibility, making MDE more practical for industry teams [11]. However, challenges persist. LLMs sometimes miss domain-specific details, like the `price` attribute in our case study, and maintaining model consistency across sprints (e.g., preserving association constraints) requires careful oversight. Integration with tools like Papyrus [7] also proves tricky, particularly in ensuring LLM-generated XMI files match Papyrus's format and handling API delays during real-time sprint tasks. To address these, we rely on tailored prompts with domain-specific examples, such as *"a ticket reservation system with a user reserving and canceling tickets"*, and team training to craft precise User Stories. In safety-critical domains like healthcare or aviation, LLM inaccuracies—such as overlooking constraints like `status` transitions tied to safety protocols—could lead to serious issues, like flawed system behavior or regulatory violations [13]. Rigorous validation is essential to catch these errors. Fine-tuning LLMs, though resource-intensive (e.g., requiring 10–100 GPU hours for domain-specific datasets), and developing custom Papyrus plugins could improve accuracy and integration.

Our approach holds promise for industries like e-commerce and transportation, where rapid prototyping and iterative refinement are key to meeting tight deadlines. By automating modeling, it speeds up development cycles, as shown in our ticket reservation system case study. However, its reliance on clear User Stories means vague inputs can lead to incomplete models, and LLMs struggle with complex domain-specific constraints, such as regulatory requirements in transportation (e.g., ensuring compliance with safety standards). To address this, combining LLMs with formal verification methods could ensure model accuracy while keeping automation benefits. Scaling the approach to larger, more complex projects or regulated domains like healthcare, where intricate constraints like patient data security are critical, remains a challenge and requires further testing. Moving forward, we plan to validate the approach in real-world industry settings, fine-tune LLMs for specific domains, and explore online modeling tools to make it more accessible. This work offers a practical way to blend MDE's precision with Agile's responsiveness, with room to grow for broader applications.

Declaration on Generative AI

The authors used Grok-3 for text editing and refinement. All outputs were reviewed and validated by the authors, who take full responsibility. No proprietary or third-party confidential data were provided to these tools.

References

[1] L. Burgueño, D. Di Ruscio, H. Sahraoui, M. Wimmer, Automation in model-driven engineering: A look back, and ahead, ACM Transactions on Software Engineering and Methodology (2025).

[2] C. Verbruggen, M. Snoeck, Practitioners' experiences with model-driven engineering: a meta-review, Software and Systems Modeling 22 (2023) 111–129.

[3] A. Alami, O. Krancher, How scrum adds value to achieving software quality?, Empirical Software Engineering 27 (2022) 165.

[4] K. Lano, S. Kolahdouz-Rahimi, J. Troya, H. Alfraihi, Introduction to the theme section on agile model-driven engineering, Software and Systems Modeling 21 (2022) 1465–1467.

[5] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang, Large language models for software engineering: A systematic literature review, ACM Transactions on Software Engineering and Methodology 33 (2024) 1–79.

[6] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, R. Rubei, On the use of large language models in model-driven engineering, Software and Systems Modeling (2025) 1–26.

[7] Eclipse Foundation, Papyrus: Open source model-based engineering tool, https://www.eclipse.org/papyrus/, 2023. Accessed: March 31, 2025.

[8] D. C. Schmidt, Model-driven engineering, IEEE Computer 39 (2006) 25–31. doi:10.1109/MC.2006.58.

[9] J. Whittle, J. Hutchinson, M. Rouncefield, The state of practice in model-driven engineering, IEEE Software 31 (2013) 79–85. doi:10.1109/MS.2013.65.

[10] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2004.

[11] Y. Zhang, S. Patel, Agile model-driven development in practice, in: Proceedings of the Agile 2007 Conference, IEEE, 2007, pp. 13–18. doi:10.1109/AGILE.2007.10.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, Advances in Neural Information Processing Systems 30 (2017) 5998–6008.

[13] G. Liebel, N. Marko, M. Tichy, Model-based engineering in safety-critical systems: A systematic literature review, Software and Systems Modeling 17 (2018) 343–365. doi:10.1007/s10270-016-0556-8.