

# Teaching the Development of Software Agents in Higher Education: First Insights from a European Initiative

Lukas Sebastian Hofmann  
Facultad de Informática  
Universidad Complutense de Madrid  
Madrid, Spain  
Mathematics and Computer Science  
Philipps-Universität Marburg  
Marburg, Germany  
Lukhofma@ucm.es

José Ignacio Requeno Jarabo  
Facultad de Informática  
Universidad Complutense de Madrid  
Madrid, Spain  
Jrequeno@ucm.es

Elena Gómez-Martínez  
Facultad de Informática  
Universidad Complutense de Madrid  
Madrid, Spain  
Mariaelena.Gomez@ucm.es

Daniel Braun  
Mathematics and Computer Science  
Philipps-Universität Marburg  
Marburg, Germany  
Daniel.Braun@uni-marburg.de

Gabriele Taentzer  
Mathematics and Computer Science  
Philipps-Universität Marburg  
Marburg, Germany  
Taentzer@mathematik.uni-marburg.de

## ABSTRACT

Artificial Intelligence (AI) is changing software engineering by enabling autonomous software agents. This shift requires new educational approaches that prepare students to design and realise such systems. We present first experiences from an exploratory seminar in which students developed software agents using three open-source frameworks: *n8n*, *OpenClaw*, and *LangChain/LangGraph*. Through project-based learning, students explored agent development, prompt engineering, and the use of software models as exchange artefacts. Our observations indicate that the different frameworks offer distinct educational benefits while highlighting common challenges, including ensuring reliable software model outputs and maintaining human oversight. These preliminary results provide insights into integrating model-driven agent development into higher education and identify directions for future curricula.

## CCS Concepts

• **Software and its engineering** → **Software creation and management**; • **Computing methodologies** → **Artificial intelligence**; • **Applied computing** → **Education**.

## KEYWORDS

Software Development, Model-Driven Engineering, Vibe Coding, Agentic Engineering, Software Agents, Computing Education

### ACM Reference Format:

Lukas Sebastian Hofmann, José Ignacio Requeno Jarabo, Elena Gómez-Martínez, Daniel Braun, and Gabriele Taentzer. 2026. Teaching the Development of Software Agents in Higher Education: First Insights from a European Initiative. In *ACM/IEEE 29th International Conference on Model*

*Driven Engineering Languages and Systems (MODELS Companion 2026)*, October 04–09, 2026, Málaga, Spain. ACM, New York, NY, USA, 5 pages.  
<https://doi.org/10.1145/3837062.3838913>

## 1 Introduction

Artificial Intelligence (AI), particularly generative and agentic AI, is fundamentally transforming software engineering [2, 11]. Beyond supporting code generation, AI-based approaches increasingly influence the architecture, design, and modelling of software systems [4, 6, 23]. Hence, the role of engineers is evolving from manually constructing artefacts to orchestrating, supervising, and developing software agents [11].

In a future in which software development becomes increasingly agentic, curricula have to adapt [1, 20, 23]. Rather than directly implementing software systems, developers will create and coordinate agents that are generating, maintaining, and evolving software. Such agents are expected to collaborate across different tasks and levels of abstraction as they are used for independent software system development with minimal to no human interaction.

(Software) Models can serve as first-order artefacts that facilitate exchange between tasks. For instance, one agent can generate use cases or user stories from informal user requirements, and another agent can subsequently generate architectural models, such as package or class diagrams [6, 12, 21]. In this context, models shift from being static documentation to evolving blueprints. Hereby, agents interpret, manipulate, and execute models, thereby introducing agentic Model-Driven Engineering (MDE), where the roles of, e.g., model transformation are augmented or orchestrated by agents; models serve as semantic boundaries enforcing domain constraints.

These developments will likely have important implications for software engineering education. To investigate this educational shift, we developed and conducted an exploratory seminar on AI in software engineering. The students were required to develop software agents and apply concepts of MDE. In doing so, we conducted an initial study to investigate how students develop agents with different technologies. Hence, our main contributions are:



This work is licensed under a Creative Commons Attribution 4.0 International License.  
*MODELS Companion 2026, Málaga, Spain*  
© 2026 Copyright held by the owner/author(s).  
ACM ISBN 979-8-4007-2903-4/26/10  
<https://doi.org/10.1145/3837062.3838913>

- (1) an initial project-based approach for teaching students how to develop model-based software agents;
- (2) preliminary evaluation from a comparison of three technology stacks to develop software agents.

This paper is structured as follows. Section 2 introduces the background and key concepts. Section 3 to Section 5 describe the course setting, teaching design, and agent requirements. Section 6 presents the preliminary evaluation. Finally, Section 7 reviews related work, and Section 8 concludes the paper and indicates further directions.

## 2 Background and Key Concepts

Software development including modelling is undergoing a transition from predominantly manual activities towards agentic ones, giving rise to emerging paradigms such as *vibe coding/modelling*, in which engineers iteratively communicate intentions through natural language interactions, and *agentic engineering/modelling*, where engineers purposefully employ agents [6, 9, 11].

Software models can serve as first-order artefacts that facilitate information exchange and task coordination among agents. Such agents may support activities including requirements-to-model model synthesis, model transformation, code generation, model validations and model-based trace checking [14, 15, 17, 24].

The combination of agentic AI and MDE offers the potential to establish processes in which agents collaborate through shared models [6, 12, 13, 21]. Nevertheless, an investigation from an educational perspective has yet to be conducted.

### 2.1 Frameworks for Agent Development

Several frameworks have emerged to develop agents. They differ in accessibility and abstraction as they range from proprietary low-code approaches to open-source frameworks. In our selection of approaches, we focus on the use of open-source projects as they provide transparency, reproducibility and enable students to inspect, adapt, and extend existing solutions rather than proprietary technology stacks, e.g., Claude Code<sup>1</sup> or Codex<sup>2</sup>.

**n8n**<sup>3</sup> is a low-code workflow automation platform that enables the visual composition of neuro-symbolic workflows. Hence, the behaviour of a *software agent* is represented as a sequence of connected actions and branches.

**OpenClaw**<sup>4</sup> is a prebuilt AI-agent. It provides reusable infrastructure for tool integration, execution management, and interaction with various AI-models. In contrast to the previous approaches, it abstracts from many low-level details of an agent.

**LangChain**<sup>5</sup> is a code-centric framework that provides abstractions for prompts, memory, tool usage, and chains of reasoning. It can be extended with *LangGraph* which enables the construction of stateful, graph-based *software agents*.

In addition to the above-mentioned frameworks, we used *Ollama*<sup>6</sup>. *Ollama* provides an execution environment for Large Languages Model (LLM) (AI) and exposes a unified interface. By decoupling the underlying AI-model from the surrounding logic, it promotes modularity and separation of concerns.

In summary, the considered frameworks provide complementary approaches for the development of agents. Yet, they differ in abstraction and complexity, ranging from low-code to code-centric.

## 3 Course Setting and Participants

The educational initiative was conducted as an exploratory seminar between instructors of the *Philipps-Universität Marburg* and the *Universidad Complutense de Madrid* during the summer term of 2026. An overview of the course settings is shown in Table 1.

Table 1: Overview of the course settings.

| Characteristic  | Description                                                     |
|-----------------|-----------------------------------------------------------------|
| Course type     | Exploratory seminar                                             |
| Credits         | 3 ECTS                                                          |
| Duration        | 12 weeks                                                        |
| Delivery mode   | Hybrid (online and in-person)                                   |
| Target audience | B.Sc. and M.Sc. students in Computer Science and related fields |
| Group size      | Up to three students per team                                   |
| Assessment      | Scientific report and tool demonstration                        |
| Main topic      | Development of software agents                                  |

Out of 121 applicants who had completed all required foundational modules in Computer Science or a related field, nine participants were selected at random. The selected cohort consisted of six undergraduate and three graduate students, including four Computer Science students, three Data Science students, and two Business Informatics students; they had prior instruction in software engineering and completed at least 70% of their degree program.

The small cohort engendered close interaction between instructors and students, as well as facilitated regular feedback sessions.

## 4 Teaching Design

The seminar was grounded in project-based learning in which multiple software agents had to be developed based on a common software specification like in [3]. In doing so, the students had to learn about their regarding stack and search for scientific articles which might help them. We have just provided the students with an introduction session on agent development and one on modelling.

To expose students to different development styles, each team independently investigated one of three distinct technology stacks for the development of software agents, i.e., workflow-centric, code-centric, and prebuild-agent approaches. The stacks are: (i) *n8n*, *Ollama*, and *VS Code Plugin infrastructure*, (ii) *LangChain*–*Graph*, *Ollama*, and *VS Code Plugin infrastructure*, and (iii) *OpenClaw*, *Ollama*, and *VS Code Plugin infrastructure*.

Throughout the semester, intermediate results were presented and discussed in regular plenum sessions, enabling students between teams to exchange experiences and reflect on their challenges. The assessment consisted of a scientific report and a presentation.

<sup>1</sup><https://code.claude.com/docs/en/overview>

<sup>2</sup><https://openai.com/codex/>

<sup>3</sup><https://n8n.io/>

<sup>4</sup><https://openclaw.ai/>

<sup>5</sup><https://www.langchain.com/>

<sup>6</sup><https://ollama.com/>

**Feedback Collection.** Feedback was collected through two semi-structured interviews with the teams and an analysis of their reports. The interview guide, as well as the questions addressed in the reports, were developed following the framework of Kallio et al. [19] and informed by experiences from Hove and Anda [18]. The feedback was combined as sources of evidence, enabling the findings to be corroborated across multiple sources for credibility.

## 5 Requirements of a Software Agent

The students were required to develop multiple specialised software agents that would communicate with one another via generated models and a shared context e.g., for namespaces. The following agents were required to be developed: (i) A requirements agent, which is tasked with producing requirement models. (ii) An architecture agent, which consumes requirement models and provides feedback or produces design models. (iii) An implementation agent, which consumes design models and provides feedback or produces fitting software systems. (iv) A testing agent, which tests the produced code based on the models created previously and reports back to the implementation agent with bug reports, cf. Fig. 1.

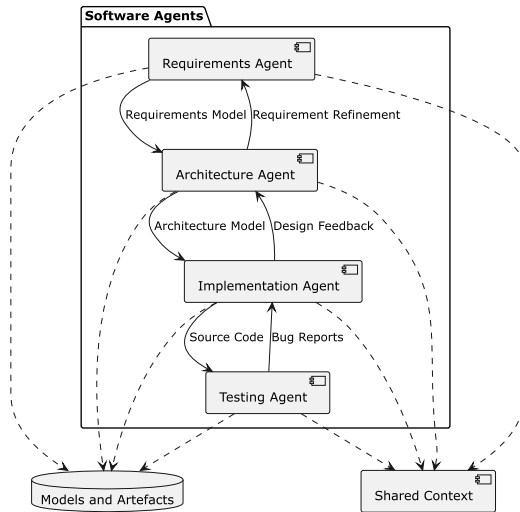


Figure 1: Architecture of the software agent to build.

Additionally, *traceability links* shall be computed between the software models generated by the agents. Furthermore, it is posited that the models shall enhance *explainability* by capturing intermediate results. By doing so, they can elucidate the rationale behind architectural decisions and artefacts, rendering the underlying reasoning of the agents more accessible. All generated models and artefacts are stored in a shared database. Finally, the software agents should provide mechanisms for *verification*, *consistency checking*, and other *quality assurances* for their generated models.

The students were required to develop and document unit-, integration-, and system-tests. Furthermore, each developed agent underwent acceptance tests conducted by teaching assistants (TAs) during review meetings to ensure the functionality.

## 6 Preliminary Evaluation

In this chapter, we present initial insights into teaching the development of software agents using a project-based approach, based on our *qualitative student-based feedback collection*, cf. Section 4.

## 6.1 Technology Comparison

*n8n and Ollama.* It can be indicated that *n8n* and *Ollama* technology stack provide a low-code environment that enabled students to quickly start developing workflow-driven AI-agents. The graphical representation simplified the initial design process and made individual workflow steps easy to inspect. Moreover, the clear separation of workflow activities made the validation of artefacts simpler, as each processing step could be developed, and tested independently. Furthermore, integrating separate sub-workflows through webhooks allowed different phases to be connected with relatively little effort. However, as workflows became larger and incorporated more AI-models and processing stages, maintaining the visual workflows became increasingly complex and execution times grew noticeably. Students also observed that much of the development effort shifted towards designing and refining prompts, as *n8n* made it easy to experiment with prompting strategies while placing less emphasis on implementing orchestration logic in code.

For instance, Team 1 designed an agentic workflow that generates user stories from natural-language requirements based on a predefined conceptual model [14]. To evaluate the workflow, they sent an API request containing the following JSON payload: “I want to create a small calculator which can do the basic arithmetic.”. They then verified whether the backend returned user stories that were consistent with the predefined conceptual model and the provided requirements. The generated user stories are then further used to derive component and class diagrams, which are used to design the system architecture and support code and test generation.

*OpenClaw and Ollama.* Students feedback suggests that *OpenClaw* (prebuild agent) provided a great working solution with little implementation effort. Most development focused on defining SKILL.md files, with one file created for each software agent (cf. Fig. 1), making prompt and context engineering the primary development activity. Nonetheless, the students reported that the prebuild agent offered limited extensibility except with SKILL.md files and required a separate Python launcher for orchestration. Furthermore, the limited context windows of locally hosted AI-models constrained the amount of context that would have been needed to operate *OpenClaw* efficiently. Finally, the agent’s broad system access introduced security risks. These were mitigated by restricting the agent’s available capabilities through the SKILL.md files and deploying it exclusively in sandboxed environments.

*LangChain/LangGraph and Ollama.* Our data suggest that the LangChain/-Graph and Ollama technology stack provide students with an easy-to-use, code-centric framework. In contrast to low-code approaches, the framework exposed the orchestration logic through source code, allowing students to directly control agent behaviour, state management, validation, and communication. An implementation can be structured into dedicated agents, prompts, schemas, and utility modules, resulting in a clear separation of concerns. While prompts remained essential for defining agent behaviour, much of the system was implemented through code, requiring students to carefully coordinate prompt design with the orchestration logic and validation mechanisms.

## 6.2 Models as First-Order Artefacts

Across all technology stacks, the students reported that they were able to beneficially use software models as the exchange artefacts between agents. Different development phases require different types of models, ranging from requirements models and architectural models, such as component, class, and sequence diagrams, to implementation models and model-based testing artifacts derived from the previously specified models, typically represented in formats such as JSON or PlantUML. While these structured representations improved the transparency and understandability of results, they also introduced additional maintenance effort, as changes in one phase required synchronising software models consumed by subsequent software agents. Yet, the linked software models enabled traceability via multiple agents.

Students further observed that software models played an important role in prompt and context engineering. Requirements of software models had to be parsed into prompts and context windows, while prompts themselves became tightly coupled to the underlying conceptual model structures. Consequently, prompts required versioning throughout development to maintain compatibility between software agents.

## 6.3 Lessons Learned for Educational Settings

The exploratory seminar yielded several transferable first insights that emerged consistently across all three technology stacks

First, students learned through direct experience that locally hosted AI-models require external assurance mechanisms, such as compilers, parsers, and schema validators, to produce structurally reliable outputs, e.g., class diagrams or component diagrams. Instruction following alone proved insufficient to guarantee usable software models and other artefacts. This lesson became deeply internalised by the students when they observed their own software agents fail and were required to design mitigation strategies.

Second, all teams recognized that prompt and context engineering is fundamentally an exercise in model specification for agentic AI. Developing software agents requires students to deeply understand abstract concepts of domain modelling, constraints (e.g., structural validity), and multi-level modelling. Without a rigorous background in MDE, students struggled to define the precise input/output interfaces for the agents, leading to cascading failures. Therefore, higher education curricula must position MDE not as a legacy design method, but as the foundational framework for structuring software agents.

Third, despite employing different technology stacks, all teams converged on the value of human-in-the-loop. They consistently indicated that inspecting and correcting artefacts was more efficient than restarting fully automated execution. This experience reinforced the seminar's broader perspective that the software engineer's role is shifting from directly authoring software towards supervising and guiding software agents.

Finally, the seminar deliberately relied on open-source AI-models, e.g., qwen3:1.7b<sup>7</sup>, gemma3:1b<sup>8</sup>, and glm-5.1<sup>9</sup> to avoid requiring students to purchase API tokens. All teams reported that hardware constraints influenced their architectural decisions. These

constraints manifested in AI-model switching strategies, reduced parallelism, and file-by-file processing rather than the generation of whole-project at once. While this provided valuable experience with resource aware system design, it also highlighted a practical challenge for teaching software agent development. Hence, educational settings should ensure that students have access to sufficient computational resources, so that hardware limitations do not become a barrier to learning and disproportionately influence outcomes.

## 6.4 An Initial Selection Guidance

Drawing on our initial insights from the student projects and collected feedback, we propose the following stack selection guidance.

We suggest using *n8n* with *Ollama* for students who have limited programming experience or a design-oriented focus. Its visual editor allowed Team 1 to become productive quickly without mastering coding, making it a good entry point for understanding what a workflow-based agent does. It is less suited to students who want to practice schema design, testing, or debugging at the code level, since its abstractions limit that exposure, and its low extensibility ceiling makes it a poor fit for highly customised architectures.

We suggest using *LangChain/LangGraph* with *Ollama* for students who have stronger programming backgrounds and an interest in formal, research-adjacent reasoning. Team 2's engagement with schema validation, static checks, and diagnosing the self-referential feedback loops reflect a depth of engineering reasoning this code-centric framework rewards. It asks more of students up front but yields the most transferable insight into how to build agents.

We observed that *OpenClaw* with *Ollama* occupies a middle ground, suiting students who should focus on prompt design and phase decomposition rather than building orchestration code from scratch as session handling and tooling are already provided. Thus, Team 3 focused on prompt and assurance design, yielding a working agent faster, at the cost of less insight into orchestration mechanics.

## 6.5 Threats to validity

The course instructor bias is a potential *internal threat*, given their role in delivering the seminar, facilitating feedback, and conducting interviews. Selection effects may also have shaped team dynamics, although the randomly selected participants came from applicants who had voluntarily enrolled in the seminar.

Regarding *external validity*, the small, heterogeneous cohort limits the direct generalisability of our conclusions to broader student populations. Whilst this interdisciplinary mix enriched qualitative discussions, the varying competencies in software engineering and modelling likely impacted the effectiveness of each team.

## 7 Related Work

We organize the related work into three areas: (i) the use of AI in software engineering education, (ii) the use of AI for agentic software engineering, and (iii) the use of AI for agentic modelling. To this end, we focus on contributions from leading venues in Software Engineering and Software Modelling.

**AI in Software Teaching.** The increasing adoption of AI has stimulated interest in its use within software engineering education. For example, [8] report widespread use of generative AI tools among software engineering students, highlighting their growing role in learning and development activities. Building on this trend, [25] explore the integration of AI tools into project-based software

<sup>7</sup><https://ollama.com/library/qwen3:1.7b>

<sup>8</sup><https://ollama.com/library/gemma3:1b>

<sup>9</sup><https://ollama.com/library/glm-5.1>

engineering courses, while [7] investigate the use of ChatGPT for learning formal specifications. Similarly, [10] study the integration of an LLM-based AI tutor into programming assessments and analyse students' experiences with AI-generated feedback. *While these studies demonstrate the potential of AI for teaching and learning activities, they focus on AI-assisted education rather than on teaching students how to engineer autonomous agents.*

**AI in Agentic Software Engineering.** Recent research has increasingly explored software engineering agents, ranging from general-purpose AI software engineers [1] to studies of agent reasoning and human-agent collaboration [20]. Other work has focused on improving agent reliability through observability and alignment mechanisms [26, 27], while specialised agents have been proposed for tasks such as software refactoring and program repair [5, 23]. Together, these studies demonstrate the growing maturity of agentic software engineering. However, these studies focus on building and evaluating agents rather than teaching future software engineers how to develop them. *This highlights a gap in software engineering education regarding the teaching of agent-based software development and motivates our educational initiative.*

**AI in Agentic Modelling.** While agent-based software engineering has recently attracted considerable attention, the application of agents to MDE tasks is still emerging. Existing work has explored specialised modelling agents for tasks such as UML generation, where [12] introduced *NOMAD*, a multi-agent framework for generating UML class diagrams from natural language requirements. Similarly, [21] presented *CLERK*, an LLM-based companion that supports the modelling of regulatory knowledge, while [13] investigated how runtime models can be augmented into distributed reasoning agents. More broadly, [22] highlight the growing convergence between MDE and LLM-based agent frameworks through a quasi-systematic review and metamodel. However, these works primarily investigate agents as artefacts to be engineered, rather than as systems that students must learn to develop. As a result, there is little understanding of how agentic MDE can be taught in higher education. *This gap motivates the educational initiative presented in this paper and highlights the need for future empirical studies on teaching and learning for agentic MDE.*

## 8 Conclusion and Future Work

In this paper, we presented insights from an exploratory project-based seminar in which three teams developed their own software agents from a common set of requirements and using software models as exchange artefacts. Our initial findings suggest that *n8n* and *OpenClaw* are well suited for beginners, whereas *LangChain* provides greater flexibility for students with stronger programming backgrounds. In future work, we want to validate our preliminary findings through a larger and standardised quantitative study that mitigates the detected threads, investigate how software engineering education should adapt to increasingly agentic development, prepare complementary exercises to support learning, and release the developed artefacts as open educational resources.

**Acknowledgments.** This work has been supported by the Spanish MINECO/FEDER projects AwESome (PID2021-122215NB-C31), BioFramework (PID2024-159323NB-I00), the Region of Madrid project DESAFiO-CM (TEC-2024/COM-235), the European Union (08273-MOB-0364) and INNOVA UCM (183 and 191).

**Data Availability.** The lecture material, interviews, reports, and presentations used as data sources are accessible on Zenodo [16].

## References

- [1] L. Applis et al. 2025. Unified software engineering agent as ai software engineer. *arXiv preprint arXiv:2506.14683* (2025).
- [2] L. Banh et al. 2025. Copiloting the future: How generative AI transforms Software Engineering. *Information and Software Technology* 183 (2025), 107751.
- [3] B.J.S. Barron et al. 2014. Doing with understanding: Lessons from research on problem- and project-based learning. In *Learning through problem solving*. Psychology Press, 271–311.
- [4] I. Bouzenia and M. Pradel. 2025. Understanding Software Engineering Agents: A Study of Thought-Action-Result Trajectories. In *2025 40th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*. IEEE Press, 2846–2857. doi:10.1109/ASE63991.2025.00234
- [5] I. Bouzenia et al. 2025. Repairagent: An autonomous, llm-based agent for program repair. In *IEEE/ACM 47th Int. Conf. on Software Engineering (ICSE)*. IEEE.
- [6] V. Campanello et al. 2025. On the use of GPT-4 in the reverse engineering of class diagrams. *Journal of Object Technology* 24, 2 (2025), 1–14.
- [7] A. Capozucca et al. 2025. Do AI assistants help students write formal specifications? a study with ChatGPT and the b-method. In *IEEE/ACM 37th Int. Conf. on Software Engineering Education and Training (CSE&T)*. IEEE, 19–29.
- [8] R. Choudhuri et al. 2025. Insights from the frontline: Genai utilization among software engineering students. In *IEEE/ACM 37th Int. Conf. on Software Engineering Education and Training (CSE&T)*. IEEE, 1–12.
- [9] J. Cito and D. Bork. 2025. Lost in Code Generation: Reimagining the Role of Software Models in AI-driven Software Engineering. *arXiv preprint arXiv:2511.02475* (2025).
- [10] E. Frankford et al. 2024. AI-Tutoring in Software Engineering Education. In *46th Int. Conf. on Software Engineering: Software Engineering Education and Training (ICSE-SEET '24)*. ACM, 309–319. doi:10.1145/3639474.3640061
- [11] A. Gadde. 2025. Democratizing software engineering through generative ai and vibe coding: The evolution of no-code development. *Journal of Computer Science and Technology Studies* 7, 4 (2025), 556–572.
- [12] P. Giannouris and S. Ananiadou. 2025. NOMAD: A Multi-Agent LLM System for UML Class Diagram Generation from Natural Language Requirements. *arXiv preprint arXiv:2511.22409* (2025).
- [13] D. Hernandez and J. Cleland-Huang. 2025. Ambient Advisory Models: Augmenting Runtime Models Into Distributed Reasoning Agents. In *ACM/IEEE 28th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS)*. IEEE.
- [14] L.S. Hofmann et al. 2025. Detecting redundancies between user stories with graphs and large language models. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 244–260.
- [15] L.S. Hofmann et al. 2025. Toward Safeguarding GenAI with Graph Transformation: Defining an Exchange Format in JSON. In *GCM 2025: Workshop on Graph Computation Models (co-located with STAF 2025)*. STAF Workshops.
- [16] L.S. Hofmann et al. 2026. *Teaching the Development of Software Agents in Higher Education: First Insights from a European Initiative*. doi:10.5281/zenodo.21279261
- [17] L.S. Hofmann et al. 2026. Towards Neuro-Symbolic Assurance for Artificial Intelligence via Graph Transformation. In *GCM 2026: Workshop on Graph Computation Models*. STAF Workshops. *CEUR Workshop Proceedings*.
- [18] S.E. Hove and B. Anda. 2005. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*. IEEE, 10–pp.
- [19] H. Kallio et al. 2016. Systematic methodological review: developing a framework for a qualitative semi-structured interview guide. *Journal of advanced nursing* 72, 12 (2016), 2954–2965.
- [20] A. Kumar et al. 2025. Why AI Agents Still Need You: Findings from Developer-Agent Collaborations in the Wild. *arXiv preprint arXiv:2506.12347* (2025).
- [21] J.S. Mercado et al. 2026. CLERK: A companion large language model expert for modeling regulatory knowledge. *Data & Knowledge Engineering* (2026), 102578.
- [22] J. W. P. Miranda et al. 2026. Bridging MDE and LLM-Based Agent Frameworks for Multi-Agent Systems: A Quasi-Systematic Review and Metamodel. *MODEL-SWARD* (2026), 33–44.
- [23] K. Oueslati et al. 2025. RefAgent: A Multi-agent LLM-based Framework for Automatic Software Refactoring. *arXiv preprint arXiv:2511.03153* (2025).
- [24] G. Quattrocchi et al. 2026. Can LLMs Generate User Stories and Assess Their Quality? *IEEE Transactions on Software Engineering* 52, 5 (2026), 1773–1790. doi:10.1109/TSE.2026.3670612
- [25] A. Rausch et al. 2026. Preparing Students for AI-Driven Agile Development: A Project-Based AI Engineering Curriculum. *arXiv preprint arXiv:2603.09599* (2026).
- [26] B. Rombaut et al. 2025. Watson: A cognitive observability framework for the reasoning of llm-powered agents. In *40th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*. IEEE, 739–751.
- [27] K. Zhang et al. 2025. Sealign: Alignment training for software engineering agent. *arXiv preprint arXiv:2503.18455* (2025).