From Sketches to Specs: AI-Assisted Lightweight Metamodeling for Spec-Driven Development

Norbert Seyff
Institute of Interactive Technologies
University of Applied Sciences and Arts Northwestern Switzerland
Windisch, Switzerland
norbert.seyff@fhnw.ch

Martin Glinz
Department of Informatics
University of Zurich
Zurich, Switzerland
glinz@ifi.uzh.ch

***Abstract*—Sketching is a natural medium for early-phase software activities. However, preserving the resulting information beyond just taking photos is difficult and laborious. A decade ago, the FlexiSketch approach addressed this problem by combining free-form sketching with lightweight metamodeling capabilities, thus enabling a smooth evolution from sketches to semi-formal artifacts. The advent of large language models (LLMs) and specification-driven development (SDD) make it worthwhile to revisit the FlexiSketch approach. LLMs can support tasks such as shape recognition or metamodel inference that were previously out of reach. SDD uses a GenAI pipeline that generates code from structured specifications. The structured models produced by an AI-enhanced FlexiSketch approach could be a natural fit for such specifications.**

We envisage an AI-assisted sketch-based workflow which lets stakeholders engage in early-phase modeling with creative freedom, while producing structured artifacts that are suitable for feeding an SDD pipeline. In this position paper, we explore our vision: we identify points in the sketch-to-code pipeline where AI assistance for a FlexiSketch style approach is beneficial, propose design principles for that assistance, and outline a research agenda.

Keywords—requirements engineering, model-driven engineering, sketch-based modeling, lightweight metamodeling, spec-driven development, large language models, FlexiSketch

I. Introduction

Sketching is a natural medium for the early phases of software projects. When requirements are still vague and the problem is not yet well understood, stakeholders and requirements engineers reach for whiteboards and paper rather than modeling tools, because the freedom of an informal medium supports the creativity that early-phase activities demand. Formal modeling tools, by contrast, impose a predefined modeling language before the participants know what they want to express, and that constraint tends to suppress exploration rather than support it.

This freedom comes with a cost: informal sketches are hard to carry forward. A photo of a whiteboard neither captures syntax nor semantics that a tool could process. So, content to be used in later phases must be re-created by hand.

A decade ago, our FlexiSketch research prototype showed that this trade-off was not inevitable [1], [2]. FlexiSketch combined a freeform sketching surface with the ability of assigning types to sketched elements, thus growing a metamodel incrementally in the background. Later in the workflow, users could export the model sketches along with their incrementally grown metamodel. This way, FlexiSketch enabled sketch-based, creative modeling with structured, processable results – powered by *lightweight metamodeling*, as we call it, i.e., growing a metamodel along with sketch-based modeling.

We argue that two developments since then make this direction worth revisiting. First, the tasks that were hardest to automate in the original work are now within reach. Tasks such as recognizing hand-drawn shapes, suggesting types for newly drawn elements or inferring metamodel rules from growing sketches required manually created heuristics or training data that was unavailable at the time. Large language models (LLMs) now appear capable of supporting these tasks directly.

Second, a downstream use for structured artifacts has emerged. Spec(ification)-driven development (SDD) is a recent practice in which structured specifications act as prompts for AI coding agents, which then generate and iterate on executable code [3], [4]. SDD is largely silent on where those specifications come from. Current practice typically assumes that somebody writes them. Using the semi-formal artifacts that result from a sketch-based, lightweight metamodeling approach might be a better way to produce SDD specifications.

Taken together, these developments raise a possibility that did not exist when FlexiSketch was first built: an AI-assisted, sketch-based workflow in which non-expert stakeholders take part in early-phase modeling with the creative freedom of informal media, while the resulting artifacts are sufficiently formal for feeding an SDD pipeline. Whether such a workflow is achievable and efficiently produces good enough results are open questions that we frame rather than answer.

Instead, this paper sets out the design space and argues for a position within it. We make three contributions.

First, we revisit the sketch-to-code pipeline, identifying the points at which AI assistance is plausibly beneficial and showing how a synthesized domain-specific language and a generated, stakeholder-validated natural-language requirements document could together feed an SDD pipeline (Section III).

Second, we propose a set of design principles that constrain how AI assistance should be introduced, so that it strengthens lightweight metamodeling rather than undermining what made it valuable (Section IV).

Third, we outline a research agenda at the intersection of sketch-based requirements engineering, model-driven requirements engineering, and AI-assisted software engineering (Section V).

II. Background and Related Work

A. Lightweight Metamodeling and Sketch-Based Tools

The idea of modeling with a not yet fixed modeling language has been around for quite a while. Example-driven and by-demonstration metamodeling approaches let users introduce example instances first, with the metamodel inferred or refined afterwards [5], [6]. Bottom-up metamodel inference works from existing artifacts to recover a metamodel that fits them [7]. Flexible and partial-modeling approaches relax conformance constraints during creation and re-tighten them later [8].

Sketch-based tools take this idea into the freeform medium of pen-and-paper diagramming. Several prototypes have explored this space: Calico, Knight, SUMLOW, and others [9], [10]. They typically target UML or other predefined notations recognized from informal input. Our own earlier work on the FlexiSketch research prototype [1], [2] differs in that the notation itself is treated as user-defined. As users sketch, they can declare that a particular shape denotes an element type, or that a particular line denotes a relation type. The metamodel grows incrementally from these declarations rather than being imposed up front. Wüest et al. [11] framed metamodel inference from sketch declarations as a semi-automatic task; later work [12] extended the approach to collaborative settings with multiple tablets and a shared electronic whiteboard.

Two properties of FlexiSketch matter for the position we develop here. First, the user – not the tool vendor – defines the notation. Second, the resulting artifact admits multiple levels of formality at the same time: parts of the sketch can be fully typed against the emergent metamodel while others remain informal. Both properties distinguish FlexiSketch from sketch-based tools that recognize predefined notations, and from conventional modeling tools that require conformance to a predefined modeling language.

B. Spec(ification)-Driven Development

SDD is a recent practice in software engineering in which structured specifications act as prompts for AI coding agents, which then generate and iterate on executable code [3], [4]. Implementations include open-source toolkits such as GitHub's Spec Kit and proprietary tools such as Amazon's Kiro. The general claim is that explicit, structured specifications produce more reliable AI-generated code than unstructured prompting, and that the specification itself becomes a durable artifact of the development process.

Two features of SDD are relevant here. First, the specification is treated as the source of truth: code is generated against it, regenerated when it changes, and validated against it. Second, the practice is largely silent on how the specification is produced. Industry discussions and toolkits typically assume that a developer or requirements engineer will write the specification, often through a conversational session with an AI assistant. Other upstream practices, such as converting existing requirements documents, generating from user stories, or eliciting from stakeholders, could in principle feed the same pipeline, but have received little systematic attention.

Academic engagement with SDD is still early. Vogelsang [13] frames the broader question of how requirements engineering should adapt when specifications increasingly function as prompts, arguing that prompt engineering and human evaluation become central RE concerns. A growing body of work investigates LLMs for individual RE tasks – for example, requirements elicitation through conversational interaction [14], completeness checking of natural-language requirements [15], and broader surveys of LLM applicability across RE tasks [16] – but the question of how specifications are produced collaboratively with stakeholders in the LLM era remains largely open.

C. Positioning

Sketch-based requirements engineering has historically produced semi-formal artifacts grounded in stakeholder dialogue. FlexiSketch and similar tools could export these artifacts into standard modeling formats, but the path from such an export to executable software was not direct: it required substantial manual modeling work in the conventional sense. SDD, conversely, has a downstream pipeline that turns specifications into executable code with the help of AI agents. However, SDD draws those specifications from a narrow set of authoring practices, typically a developer or requirements engineer working alone with an AI assistant, that exclude most early-phase stakeholder activity. The two lines of work address different ends of what could be a single pipeline, and the gap between them has not been systematically explored.

It is reasonable to ask whether SDD is simply a re-emergence of model-driven engineering (MDE) under new labels. There is overlap, but we see two substantial differences. First, classic MDE typically assumes a metamodel designed up front by modeling experts, while the lightweight metamodeling approach we revisit here treats the metamodel as something that emerges from stakeholder activity. This is not just a technical difference, since it changes who controls the vocabulary the system will eventually encode. Second, where MDE pipelines have historically required tooling built around a specific metamodel (transformation rules, generators, validators), SDD generation runs through general-purpose AI agents that consume a specification as context rather than as input to a compiled toolchain. This makes lightweight, project-specific metamodels deployment-feasible in a way that they have not been before. We argue these differences are substantive enough to treat the integration as a research direction in its own right, though empirical work is needed to confirm this.

We take the position that AI-assisted lightweight metamodeling is a candidate worth investigating for closing the gap between sketch-based RE and SDD, and we devote the remainder of the paper to examining what such an integration would require.

III. Where AI Assistance Could Help

We revisit the FlexiSketch pipeline – sketch capture, declaration of element and relation types, metamodel inference, and lifting into downstream artifacts – and identify a set of points at which AI assistance appears worth investigating. The points are not equally promising, and not all of them are equally low-risk to implement; Section IV develops design principles intended to constrain how each is approached. Figure 1 summarizes the sketch-to-code pipeline and locates the integration points discussed below.

A. Type Proposals for Sketched Elements and Relations

When a user sketches a new shape that resembles shapes already typed in the same session, a vision-capable LLM could

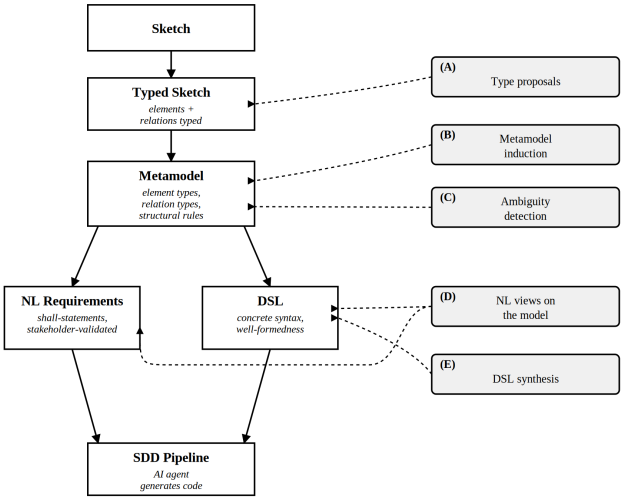

Fig. 1. The sketch-to-code pipeline with AI-assistance integration points (A–E) mapped to the pipeline stages discussed in Section III. The metamodel emerges incrementally from the typed sketch; the DSL is synthesized from the stabilized metamodel; the natural-language requirements are generated from the same model. Together, the DSL and the validated requirements feed the SDD pipeline.

propose a likely element type, leaving the user to confirm, modify, or reject the suggestion. The same applies to relation types between sketched elements. This addresses one of the recurring frictions in the original FlexiSketch tool: the cognitive cost of repeatedly typing similar elements during a fast-moving elicitation session. Wüest et al. [11] framed type assignment as a partially automated task. The capabilities of today's LLMs suggest that meaningful assistance at this point in the pipeline is now within reach. How well this will actually work in practice remains to be studied.

B. Metamodel Inference from a Growing Sketch

As element and relation types accumulate, an LLM can be asked to infer the metamodel implied by the sketch so far, that is, identifying candidate generalizations across element types, recurring structural patterns, and relations that appear to follow consistent rules. This kind of abstraction is difficult for humans to perform during an active sketching session, when their attention is committed to the domain content rather than to the structure of the emerging language. Two design considerations are important here. First, the tool should offer the inferred metamodel as a proposal that the requirements engineer can act on after the session, rather than silently imposing it. Second, the inference engine should not be permitted to add elements or relations that do not already appear in the sketch; its role is to surface structure that is implicit, not to invent structure that is not there. In practice this constraint can be enforced by requiring every proposed type or rule to reference at least one sketch element by identifier, and by validating the model's output against the sketch's element inventory. That said, the inference engine also could infer potentially missing metamodel concepts and generate corresponding hints. Existing work on AI-assisted metamodel completion, e.g., [17], needs to be considered here.

C. Ambiguity and Consistency Detection

Large language models are well suited to surfacing the kinds of inconsistencies that are easy to miss during live sketching. For example, an element labeled "Order" might be used once as an entity and once as an activity; a relation labeled "manages" might carry different cardinalities in different parts of the sketch; or naming collisions might occur across collaboratively sketched diagrams. The AI-assistant should flag such ambiguities rather than resolve them. Sketch-based elicitation deliberately tolerates partial formality, and silent resolution risks committing the users of the tool to interpretations they never validated. The point is not to remove ambiguity automatically, but give the requirements engineer and the stakeholders a list of issues, maybe with hints for resolving them.

D. Natural-Language Views on the Model

Two uses of natural-language generation from the model are worth considering, with different audiences and timing.

1. Generating natural language paraphrases. During the session, a running paraphrase – produced by an LLM and visible to participants – gives stakeholders who cannot fluently read semi-formal diagrams a representation they can engage with. As the sketch grows, the LLM generates short prose descriptions of what the typed elements and relations currently say: "A customer can place orders; each order belongs to one customer." Stakeholders can challenge the paraphrase ("that's not what I meant"), which lets the requirements engineer or the stakeholders themselves correct the underlying sketch. It might also be possible to let the LLM generate proposals for how to do the corrections. This is the integration point that may matter most for elicitation quality: it helps detect faults early, fosters stakeholder involvement and supports stakeholders who are not familiar with interpreting requirements models.

2. Generating natural language specifications. After the session, once the model has stabilized, the same generative capability can be turned to producing a structured natural-language requirements artifact, for example, "shall" statements, scenarios, or acceptance criteria, depending on the project's conventions. For example, such an artifact might contain statements such as "each customer shall be able to place one or more orders" (a natural-language rendering of structural content already captured in the metamodel) and "The system shall not confirm an order before the customer's payment method has been validated" (behavioral and temporal content that goes beyond what a structural metamodel typically captures). Such an artifact serves downstream uses that the model alone does not address: human review and sign-off, regulatory documentation, traceability records, and inputs to other requirements workflows. How requirements content that goes beyond the structural vocabulary of the metamodel should be elicited, represented, and traced is an open question we return to in Section V.

E. DSL Synthesis from the Stabilized Metamodel

Once a sketching session concludes and the emergent metamodel has stabilized, an LLM can be asked to lift it to a domain-specific language with an explicit abstract syntax, a concrete syntax aligned with the sketched notation, and a set of well-formedness constraints. The DSL is one part of the handoff to SDD. Traceability runs from sketch element to metamodel concept to DSL construct to generated code – a chain that, if maintained, makes the whole pipeline inspectable end-to-end. Whether current language models can synthesize DSLs of sufficient quality to play this role, and whether the resulting DSLs are durable across sessions, are open empirical questions.

F. Combining the NL Spec and the DSL Views

The combination of the generated natural language requirements (Section III.D.2) with the synthesized DSL (Section III.E), gives the AI coding agent both an explicit ontology and validated behavioral statements, with traceability between them. The DSL synthesized from the metamodel serves as the source of structural truth – the concepts that exist, the relations between them, the constraints that govern them. The natural-language requirements extend this with behavioral and temporal content that the DSL does not capture, while remaining anchored to its vocabulary.

G. Not All Points Are Equally Promising

The integration points above are not on equal footing. Type proposals (A) and natural-language views (D) appear the lowest-risk and the most immediately tractable. Metamodel inference (B) is potentially the most valuable but also the most exposed to the risks discussed in Section IV below: silent inference, bias toward standard notations, and drift away from the stakeholders' own emergent language. DSL synthesis (E) is the most ambitious and the most dependent on capabilities that current language models have not been systematically evaluated on for this purpose. Ambiguity detection (C) is intermediate: useful, generally low-risk, but challenging if one tries to resolve rather than detect.

IV. Design Principles for AI-Assisted Lightweight Metamodeling

The integration points in Section III are not equally safe to implement, and naive integration could undermine the property that made FlexiSketch distinctive in the first place: that the users control the notation. We propose the following design principles as constraints on how AI assistance should be introduced into a lightweight metamodeling workflow.

A. User Authority Over Notation

AI assistance shall propose, never impose. All type assignments, metamodel changes, DSL choices, etc. have to be suggestions that the user can accept, modify, or reject. If the model autonomously commits structural decisions, the tool collapses into a fixed-vocabulary modeling environment, which is exactly what lightweight metamodeling was built to avoid.

B. Resistance to Notation Homogenization

LLMs carry strong priors toward standard notations: UML class diagrams, BPMN flows, ER models, RDF triples. Without explicit countermeasures, projects assisted by an LLM are likely to drift toward the same handful of patterns, defeating the purpose of supporting domain-specific emergent notations. Concrete countermeasures include retrieval restricted to within-session context, prompt-level penalties for importing external vocabulary, and explicit user-controlled "stay inside our language" modes. A high LLM temperature might also help.

C. Visible and Reversible AI Inferences

Every AI-driven inference should be a first-class, visible, and reversible event in the model history. Silent reorganization of the metamodel makes it difficult for the requirements engineer to reconstruct why the model looks the way it does, which matters for downstream negotiation, sign-off, and audit. Therefore, reviewability is a requirement of the integration, not an optional feature.

D. Preservation of Productive Ambiguity

Premature disambiguation can be a defect rather than a virtue. If an LLM resolves every underspecified element by guessing, it locks in interpretations that stakeholders never had a chance to validate. AI assistance should flag underspecification (e.g., "this element appears in three diagrams with different relations; is that intentional?") rather than silently resolve it. The tolerance of the approach for partial formality must extend to AI-assisted versions of the same workflow.

E. Traceability of AI Inferences to Sketch Origin

Every AI-produced suggestion, be it a proposed type, a metamodel rule, a synthesized DSL construct, or a generated requirements statement, must carry an explicit link to the sketch elements that it originates from. Without such links, the requirements engineer cannot tell which inferences are grounded in the stakeholders' work and which the model has introduced on its own. Traceability at the inference level also enables traceability across the broader pipeline: if every downstream artifact can be traced to its origins, the SDD pipeline becomes inspectable end-to-end.

V. Downstream Implications and Research Agenda

A. Implications for Spec(ification)-Driven Development

The distinguishing feature of our proposed integration path to producing SDD specifications is that they emerge from the stakeholders' needs and ideas, captured as sketches and result, with AI assistance, in both a project-specific DSL and a validated natural-language requirements document. The downstream coding agent thus operates on validated stakeholder needs and vocabulary. This addresses a key problem of SDD: confidently generating code from misencoded stakeholder needs.

B. Open Research Questions

In this position paper, we raise many questions that need to be answered by future research. In our view, the six questions listed below are the most important ones.

1) Empirical evaluation. How does AI-assisted sketching with lightweight metamodeling compare to unassisted sketching and to other ways of producing SDD specifications, measured in terms of completeness, stakeholder validation, and downstream code quality? This is the most basic question the proposal raises and the one against which any further claims must be measured.

2) The combination question. Does the pairing of a synthesized DSL and a stakeholder-validated natural-language requirements document – both derived from the same emergent metamodel – improve AI agent performance compared to other approaches to producing SDD specifications? The hypothesis is that the DSL provides structural grounding the agent can check against, while the natural-language requirements convey behavioral intent the DSL does not capture; neither alone may suffice, but together they may constrain generation more tightly than either. This is the central empirical question on which the value of our proposal rests.

3) Scope of the natural-language artifact. How should requirements content that goes beyond the structural vocabulary

of the metamodel, such as behavioral rules, value constraints, or temporal conditions, be elicited, represented, and traced back to artifacts of the sketching session? Without a principled answer, the natural-language layer risks either underrepresenting behavioral intent or drifting away from its anchor in the metamodel.

4) Resistance to homogenization. To what extent do AI-assisted metamodels drift toward LLM priors such as UML, BPMN, or ER, and which countermeasures effectively preserve domain-specific notations? The same models that make assistance possible also carry the strongest tendency to collapse emerging notations back to familiar patterns, so measuring this drift and testing countermeasures is a precondition for trusting the rest of the pipeline.

5) Cross-session pattern reuse. When AI assistance has access to prior sessions, does notation reuse become more productive or merely more uniform? The risk is that learned patterns increase efficiency for individual sessions while gradually homogenizing the notation space across projects, eroding the property the approach is designed to preserve.

6) Longitudinal metamodel evolution. How does a metamodel evolve over multiple sessions when AI assistance is present versus absent? Does AI assistance accelerate productive convergence, or does it accelerate premature commitment to structural decisions that stakeholders had not yet endorsed?

C. A Call to the MoDRE Community

These questions sit at the intersection of model-driven requirements engineering, sketch-based modeling, and AI-assisted software engineering. They cannot be answered by any one of these communities alone. We invite the MoDRE community to engage, particularly through empirical studies, comparative tool evaluation, and the development of shared benchmarks for AI-assisted lightweight modeling.

VI. Conclusion

A decade ago, FlexiSketch demonstrated that stakeholder sketches could remain creative while still producing processable models. Today, SDD needs structured specifications to feed AI coding agents but is largely silent on where those specifications come from. We have argued that closing the gap between these two lines of work is worthwhile and technically tractable.

The synergy runs in both directions. AI assistance makes sketch-based modeling with lightweight metamodeling more tractable by automating tasks such as type proposal, metamodel inference, ambiguity detection, and natural-language generation that previously required hand-written heuristics or unavailable training data. Sketch-based modeling, in turn, gives SDD a way to obtain specifications that have been authored and validated with stakeholders rather than improvised by a developer's prompt. The case for revisiting this research line rests on the combination, not on either contribution alone.

Our design principles are not final answers; rather they are constraints under which we believe such an integration would be worth pursuing. Whether the integration delivers on its promise is a question we have framed but not answered. We invite the MoDRE community, together with researchers in sketch-based RE and AI-assisted development, to take up the challenge.

Acknowledgments

We used an AI assistant for language editing and polishing of this manuscript. All motivation and arguments originate from the authors.

References

[1] D. Wüest, N. Seyff, and M. Glinz, "FlexiSketch: A mobile sketching tool for software modeling," in Proc. 4th Int. Conf. Mobile Computing, Applications and Services (MobiCASE 2012), Seattle, USA, 2013, pp. 225–244.

[2] D. Wüest, N. Seyff, and M. Glinz, "FlexiSketch: A lightweight sketching and metamodeling approach for end-users," Software and Systems Modeling, vol. 18, no. 2, pp. 1513–1541, 2019.

[3] B. Böckeler, "Understanding spec-driven development: Kiro, spec-kit, and Tessl," martinfowler.com, 2025. [Online]. Available: https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html

[4] D. Delimarsky, "Spec-driven development with AI: Get started with a new open source toolkit," The GitHub Blog, Sep. 2, 2025. [Online]. Available: https://github.blog/ai-and-ml/generative-ai/spec-driven-development-with-ai-get-started-with-a-new-open-source-toolkit/

[5] J. J. López-Fernández, J. Sánchez Cuadrado, E. Guerra, and J. de Lara, "Example-driven meta-model development," Software and Systems Modeling, vol. 14, no. 4, pp. 1323–1347, 2015.

[6] H. Cho, J. Gray, and E. Syriani, "Creating visual domain-specific modeling languages from end-user demonstration," in Proc. 4th Int. Workshop on Modeling in Software Engineering (MiSE), 2012, pp. 22–28.

[7] F. Javed, M. Mernik, J. Gray, and B. R. Bryant, "MARS: A metamodel recovery system using grammar inference," Information and Software Technology, vol. 50, no. 9–10, pp. 948–968, 2008.

[8] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty," in Proc. 34th Int. Conf. Software Engineering (ICSE 2012), 2012, pp. 573–583.

[9] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: How and why software developers use drawings," in Proc. SIGCHI Conf. Human Factors in Computing Systems, 2007, pp. 557–566.

[10] Q. Chen, J. Grundy, and J. Hosking, "SUMLOW: Early design-stage sketching of UML diagrams on an e-whiteboard," Software: Practice and Experience, vol. 38, no. 9, pp. 961–994, 2008.

[11] D. Wüest, N. Seyff, and M. Glinz, "Semi-automatic generation of metamodels from model sketches," in Proc. 28th IEEE/ACM Int. Conf. Automated Software Engineering (ASE 2013), Silicon Valley, USA, 2013, pp. 664–669.

[12] D. Wüest, N. Seyff, and M. Glinz, "Sketching and notation creation with FlexiSketch Team: Evaluating a new means for collaborative requirements elicitation," in Proc. 23rd IEEE Int. Requirements Engineering Conf. (RE'15), Ottawa, Canada, 2015, pp. 186–195.

[13] A. Vogelsang, "From specifications to prompts: On the future of generative large language models in requirements engineering," IEEE Software, vol. 41, no. 5, pp. 9–13, 2024.

[14] K. Ronanki, C. Berger, and J. Horkoff, "Investigating ChatGPT's potential to assist in requirements elicitation processes," in Proc. 49th Euromicro Conf. Software Engineering and Advanced Applications (SEAA), 2023, pp. 354–361.

[15] D. Luitel, S. Hassani, and M. Sabetzadeh, "Improving requirements completeness: Automated assistance through large language models," Requirements Engineering, vol. 29, no. 1, pp. 73–95, 2024.

[16] J. J. Norheim, E. Rebentisch, D. Xiao, L. Draeger, A. Kerbrat, and O. L. de Weck, "Challenges in applying large language models to requirements engineering tasks," Design Science, vol. 10, e16, 2024.

[17] M. Weyssow, H. A. Sahraoui, and E. Syriani, "Recommending metamodel concepts during modeling activities with pre-trained language models," Software and Systems Modeling, vol. 21, no. 3, pp. 1071–1089, 2022.