Generating and Correcting Design Diagrams with LLMs

Caio V. M. Penayo Matr. 350183

Tiago G. Felice Matr. 350158

Hossein Nadali Matr. 343389

Thomas K/BIDI Matr. 350263

Abstract

The objective of this paper is to investigate the use of multi-agent Large Language Model (LLM) architectures for generating Unified Modeling Language (UML) class diagrams from natural language requirements. While single-pass LLMs can produce syntactically valid models, they often observe a performance drop regarding semantic correctness, specifically in complex scenarios like inheritance or association classes. To address this, we explore multi-agent systems that decompose the process into generation, critique, and revision. However, this approach presents a major drawback, regression, where misguided critiques degrade initially correct solutions. Hence, we will evaluate an enhanced architecture with a selection mechanism to mitigate regressions, in order to affirm its effectiveness in improving the robustness of the generated models.

1 Introduction

Our evaluation combines structural metrics, renderability checks, and LLM-based qualitative assessments on a set of representative modeling exercises. The results show that while naive multi-agent approaches may correct some semantic errors, they can also introduce regressions. In contrast, the selection-based architecture consistently improves or preserves solution quality, eliminating fixer-induced degradations.

The main contributions of this paper are:

- a reproducible single-agent baseline for UML class diagram generation from textual requirements;
- an empirical comparison of two multi-agent architectures highlighting the risks of unconditional revision;
- a selection-based multi-agent strategy that stabilizes the generation pipeline and improves semantic correctness;
- an evaluation combining automatic metrics and qualitative analysis to assess modeling quality.

2 Background and Related Work

UML class diagrams are a well-established artifact in Software Engineering, widely used to represent the static structure of software systems during analysis and design. Their effectiveness depends on correctly identifying domain entities, relationships, and constraints, making the modeling task highly reliant on both domain knowledge and abstraction skills. Prior research shows that even when syntactic correctness is achieved, semantic modeling errors remain common, particularly among novice modelers (Bolloju and Leung, 2006).

Recent advances in Large Language Models (LLMs) have motivated their application to a broad range of Software Engineering tasks, including requirements analysis, code generation, documentation, and testing. More recently, researchers have explored the use of LLMs to generate design artifacts such as UML diagrams from natural language descriptions (De Bari et al., 2024). Existing approaches rely on single-pass generation, where a model directly produces a diagram from textual input (De Bari et al., 2024).

Although single-agent LLM approaches often generate syntactically valid UML diagrams, recurring semantic limitations have been reported (De Bari et al., 2024). Common issues include missing domain concepts, incorrect relationship types, misuse of aggregation or composition, and the absence of association classes when relationships carry attributes. These limitations indicate that deeper modeling decisions remain challenging to capture in a single reasoning step.

Multi-agent LLM systems have emerged as a promising strategy for addressing complex reasoning tasks by decomposing them into specialized roles such as generation, critique, and revision. While critique-and-revise patterns have shown benefits in improving output quality, prior work also highlights the risk of instability in iterative LLM-based pipelines, motivating explicit control and validation mechanisms (Feldt et al., 2023).

Despite increasing interest in multi-agent LLM systems, empirical evidence of their effectiveness for software modeling tasks—particularly UML class diagram generation—remains limited. Existing studies rarely analyze regression risks introduced by revision steps or compare unconditional fixing with selection-based strategies. In this work, we address this gap by empirically comparing single-agent and multi-agent pipelines and analyzing how architectural choices affect both correctness and stability.

3 Experimental Design

This study follows an empirical evaluation approach to assess the impact of single-agent and multi-agent LLM architectures on the quality and correctness of UML class diagrams generated from natural language requirements. The experimental design is structured around clearly defined research questions, a controlled dataset of modeling exercises, and a combination of automatic and qualitative evaluation metrics.

3.1 Research Questions

The experiment is guided by the following research questions:

- **RQ1**: Does the use of multi-agent LLM architectures impact the quality and correctness of UML class diagrams generated from natural language requirements?
- **RQ2**: Which architectural aspects of multi-agent systems most strongly influence output quality and stability?

RQ1 focuses on comparing single-agent generation with two multi-agent variants, while RQ2 investigates the role of critique, revision, and selection mechanisms in preventing regressions and improving semantic correctness.

3.2 Dataset and Modeling Exercises

The evaluation is conducted on a dataset of five UML modeling exercises designed to represent common and diverse modeling patterns. Each exercise consists of a short natural language description and a corresponding ground-truth UML class diagram expressed in PlantUML.

The selected exercises cover a range of modeling scenarios, including:

- simple binary associations,
- association classes with relationship attributes,
- inheritance hierarchies,
- composition relationships with lifecycle constraints,
- multi-entity interaction with cardinality restrictions.

This diversity allows the evaluation to expose limitations related not only to syntactic correctness but also to higher-level semantic modeling decisions. All ground-truth diagrams were manually defined and used exclusively for evaluation purposes.

3.3 Evaluation Metrics

Generated diagrams are evaluated using a combination of automatic structural metrics, renderability checks, and qualitative assessments.

Structural Correctness. We compute precision, recall, and F1-score for both classes and relationships by comparing generated diagrams against the ground truth. Relationships are matched based on connected entities and relationship type. In addition, we measure *multiplicity accuracy*, defined as the proportion of correctly generated multiplicities over those present in the ground truth.

Renderability. Each generated PlantUML diagram is rendered using the PlantUML toolchain. The rendering success is recorded as a binary indicator, ensuring that the reported results correspond to syntactically valid and renderable diagrams.

Qualitative Assessment. To capture semantic and pragmatic aspects that are not fully reflected by structural metrics, we employ an LLM-based judge that assigns a quality score on a scale of 0–100. The judge evaluates the diagram with respect to the requirement coverage, conceptual correctness, and overall clarity. However, judge scores are used as a tool to conduct a relative comparison between alternative models for the same exercise, rather than as absolute measures.

Comparative Analysis. For multi-agent architectures, metrics are computed on the final selected diagram. We additionally record score deltas between single-agent, multi-agent v1, and multi-agent v2 outputs to analyze improvements and regressions introduced by architectural changes.

4 Single-Agent Baseline

As a baseline, we implement a single-agent pipeline in which a single LLM generates a UML class diagram directly from the natural language requirements in a single step. This approach represents the most common usage pattern of LLMs for design artifact generation (De Bari et al., 2024) and serves as a reference point for evaluating the benefits and limitations of multi-agent architectures.

Given an exercise description, the agent is prompted to produce a UML class diagram expressed in PlantUML. The prompt explicitly enforces output constraints, requiring valid PlantUML syntax and prohibiting any additional formatting. The generated diagram is then evaluated without any iterative refinement or external feedback, reflecting a single-pass generation strategy.

The baseline is assessed using the evaluation framework described in Section **??**. The metrics show that the single-agent approach performs well in terms of basic diagram completeness. In most exercises, the generated diagrams achieve perfect or near-perfect precision and recall for classes and relationships, indicating that the model can correctly identify core domain entities and their connections.

However, despite strong structural scores, qualitative analysis reveals recurring semantic and conceptual limitations. In exercises where requirements imply association classes or relationship attributes, the single-agent often produces structurally correct but conceptually incomplete diagrams. For example, relationships that require dedicated modeling constructs are frequently represented as simple associations, omitting intermediary classes needed to capture domain semantics. Similar issues arise with life-cycle constraints, where compositions are replaced by weaker association types.

These limitations are reflected in the qualitative judge scores, which vary significantly between exercises even when structural metrics remain high. This discrepancy highlights a key weakness of the single-agent approach: surface-level correctness does not guaranty semantic adequacy. Even if the agent is effective at reproducing common UML patterns, it struggles to consistently apply higher-level modeling decisions that depend on interpreting implicit constraints in the requirements.

Overall, the single-agent baseline establishes a strong reference in terms of syntactic validity and basic structural alignment, but it also exposes the challenges of relying on a single reasoning step for complex modeling tasks. These observations motivate the exploration of multi-agent architectures that introduce explicit critique, revision, and control mechanisms to improve semantic correctness and modeling robustness.

5 Multi-Agent Architectures

To address the limitations observed in the single-agent baseline, we design and evaluate two multi-agent architectures that decompose the UML diagram generation task into specialized roles. Both architectures follow a sequential pipeline, but differ in how revisions are handled.

5.1 Multi-Agent v1: Generator–Critic–Fixer

The first multi-agent architecture (MA1) extends the single-agent baseline by introducing an explicit critique-and-revision loop. The goal of this design is to determine whether separating generation and evaluation responsibilities can improve semantic correctness without modifying the underlying generation model.

The pipeline consists of three agents:

- **Generator**, which produces an initial UML class diagram from the natural language requirements, using the same prompt and configuration as the single-agent baseline;
- **Critic**, which analyzes the generated diagram and identifies semantic issues, missing elements, or conceptual inconsistencies, without modifying the diagram itself;
- **Fixer**, which revises the diagram based solely on the critic's feedback.

In this architecture, the output produced by the fixer is always accepted as the final solution, regardless of whether the revision improves or degrades the original diagram. This unconditional acceptance policy allows us to isolate the effect of critique and revision, but it also introduces the risk of regressions.

Empirically, MA1 demonstrates that the critique–fix loop can successfully correct certain semantic modeling errors that the single-agent baseline fails to address, such as missing association classes or incomplete domain abstractions. However, the absence of a control mechanism means that the fixer may also introduce unnecessary elements or inappropriate relationships, leading to degraded solutions in cases where the original diagram was already acceptable.

5.2 Multi-Agent v2: Generator–Critic–Fixer–Selector

The second architecture (MA2) refines the previous pipeline by introducing an additional **selector** agent designed to prevent fixer-induced regressions. Rather than assuming that every revision leads to an improvement, MA2 explicitly evaluates whether a fixed diagram should replace the original one.

The MA2 workflow proceeds as follows: the generator produces an initial diagram, the critic analyzes it, the fixer optionally generates a revised version, and the selector then compares the original and revised diagrams. The selector evaluates both alternatives using renderability checks and qualitative judge scores. The revised diagram is accepted only if it renders correctly and achieves a higher quality score than the original; otherwise, the original diagram is preserved.

This selection-based decision policy introduces a conservative control mechanism that stabilizes the multi-agent pipeline. By allowing improvements to be adopted while rejecting harmful revisions, MA2 combines the corrective potential of critique–fix loops with robustness against over-modification.

As a result, MA2 preserves correct solutions produced by the generator while selectively incorporating beneficial fixes. This architectural refinement directly addresses the instability observed in MA1 and enables a more reliable application of multi-agent LLM systems to UML class diagram generation.

6 Results

This section reports the results of the empirical evaluation, comparing the single-agent baseline (SA) with the two multi-agent architectures (MA1 and MA2). We first present quantitative results based on automatic metrics and judge scores, followed by a qualitative analysis highlighting how architectural choices influence modeling outcomes.

Table 1: Quantitative Comparison (Mean $\pm$ STD)

Metric	**SA**	**MA1**	**MA2**
Judge Score (%)	67.0 ± 29.1	71.0 ± 20.1	82.0 ± 20.5
Class F1	0.95 ± 0.11	0.97 ± 0.06	0.95 ± 0.11
Relation F1	0.67 ± 0.47	0.68 ± 0.46	0.67 ± 0.47
Multiplicity Acc.	0.40 ± 0.55	0.20 ± 0.45	0.20 ± 0.45

6.1 Quantitative Results

Table 1 summarizes the quantitative comparison between the three approaches across the five exercises. We report judge scores, structural F1-scores for classes and relationships, multiplicity accuracy, and renderability.

Overall, all approaches consistently produce renderable diagrams, indicating that syntactic correctness is not a discriminating factor. Similarly, class and relationship F1-scores are high for most exercises across all configurations, often reaching perfect scores. This confirms that LLM-based approaches are effective at identifying core entities and basic relationships.

However, significant differences emerge in the qualitative judge scores. Averaged across all exercises, the single-agent baseline achieves a mean score of 67.0, while MA1 improves slightly to 71.0. In contrast, MA2 achieves a substantially higher average score of 82.0, corresponding to an improvement of +15 points over the single-agent baseline.

At the individual exercise level, MA2 improves or matches the single-agent score in all cases. Notably, improvements are most pronounced in exercises involving more complex modeling decisions. For instance, MA2 outperforms SA by +20 points in Exercise 2 and by +30 points in Exercise 3, while preserving equivalent performance in simpler scenarios. Importantly, MA2 never degrades solution quality relative to SA.

In contrast, MA1 exhibits mixed behavior. While it improves upon SA in some exercises, it also introduces regressions, as reflected by unchanged or lower judge scores in certain cases. These results suggest that critique-and-fix alone is insufficient to guarantee stable improvements.

Structural metrics further support this interpretation. While class and relationship F1-scores remain largely unchanged across architectures, multiplicity accuracy proves more challenging. Neither MA1 nor MA2 consistently improves multiplicity correctness, indicating that fine-grained cardinality modeling remains difficult even with multi-agent reasoning. Nevertheless, MA2 avoids the multiplicity regressions observed in MA1 by preserving original solutions when fixes are detrimental.

6.2 Qualitative Analysis

To better understand the observed quantitative trends, we conducted a qualitative analysis of how each architecture affected individual exercises. The single-agent baseline performs well for straightforward modeling tasks but struggles with implicit semantic requirements, such as the need for association classes or stronger lifecycle constraints.

The MA1 architecture demonstrates that the critic–fixer loop can successfully correct such issues. In Exercise 2, for example, MA1 introduces an association class required to store relationship attributes, correcting a limitation of the single-agent output. However, because all fixes are unconditionally accepted, MA1 may also introduce unnecessary or incoherent modeling elements. This behavior is particularly evident in Exercise 5, where an initially acceptable diagram is degraded by the fixer.

The MA2 architecture resolves this instability through explicit selection. By comparing original and revised diagrams and accepting fixes only when they lead to a measurable improvement, MA2 preserves correct solutions while still benefiting from critique-driven revisions. In Exercise 5, MA2 correctly rejects a degraded revision and retains the original diagram. Conversely, in Exercise 4, MA2 successfully accepts a beneficial fix that MA1 failed to exploit.

These observations are summarized in a qualitative comparison indicating which agent corrected or preserved each solution. Across all exercises, MA2 consistently avoids regressions while incorporating genuine improvements when available.

6.3 Answering the Research Questions

RQ1: Does the use of multi-agent LLM architectures impact the quality and correctness of UML class diagrams? Yes. The results show that multi-agent architectures can significantly improve diagram quality, but only when combined with appropriate control mechanisms. While MA1 demonstrates the potential of critique-and-revision, MA2 achieves consistent and substantial improvements over the single-agent baseline.

RQ2: Which architectural aspects most strongly influence output quality and stability? The results indicate that the decision policy governing revision acceptance is critical. Unconditional fixing introduces regressions, whereas selection-based acceptance stabilizes the pipeline. Role separation alone is insufficient; explicit comparison and conservative selection are essential for reliable multi-agent UML generation.

7 Discussion and Conclusion

The results of this study highlight a clear distinction between structural correctness and semantic adequacy in LLM-generated UML class diagrams. While all evaluated approaches consistently achieve high structural scores, qualitative differences emerge when diagrams are assessed in terms of conceptual modeling decisions. This confirms that surface-level correctness is insufficient for capturing implicit requirements such as association classes, lifecycle constraints, and abstraction choices.

The comparison between the evaluated architectures shows that multi-agent reasoning can improve UML generation, but only under appropriate control policies. The generator–critic–fixer pipeline demonstrates that critique-driven revision can correct certain semantic errors; however, unconditional acceptance of revisions introduces instability and may degrade previously correct solutions. In contrast, the selection-based architecture consistently improves or preserves solution quality by explicitly controlling when revisions are applied. This suggests that architectural decision policies play a more critical role than the mere addition of reasoning agents.

From a practical perspective, these findings have direct implications for the design of LLM-based modeling assistants. Systems intended to support software modeling, particularly in educational or exploratory settings, must balance corrective feedback with robustness. Selection mechanisms provide a lightweight yet effective means of preventing over-modification while still benefiting from multi-agent reasoning.

This study is subject to certain limitations. The evaluation relies in part on an LLM-based judge to assess qualitative aspects of modeling quality, and the number of exercises considered is limited. While the dataset is small, it was intentionally designed to cover diverse UML modeling patterns, and the consistency of observed trends across exercises suggests that the results are not artifact-specific. Nevertheless, future work should validate the proposed architecture on larger datasets and with additional evaluation perspectives.

In conclusion, this paper presented an empirical comparison of single-agent and multi-agent LLM architectures for UML class diagram generation. The results show that a naive multi-agent approach is insufficient to guarantee improvements, whereas a selection-based multi-agent pipeline achieves stable and consistent gains in semantic modeling quality. Future work includes expanding the evaluation scope, incorporating human assessment, and exploring more specialized agent roles tailored to specific UML modeling patterns.

References

Narasimha Bolloju and Felix S. K. Leung. 2006. Assisting novice analysts in developing quality conceptual models with uml. *Communications of the ACM*, 49(7):108–112.

Daniele De Bari, Giacomo Garaccione, Riccardo Coppola, Luca Ardito, and Marco Torchiano. 2024. Evaluating large language models in exercises of uml class diagram modeling. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*.

Robert Feldt, Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Towards autonomous testing agents via conversational large language models. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*.