Automated Generation of Activity Diagrams Using Prompt-Based Learning

Minor Project (PC842)

Presented By:
Vrajkumar Patel

Mentor:
Prof. Saurabh Tiwari

Outline

- Introduction
  - What is UML?
  - What is an Activity Diagram
- Objectives
- Methodology
  - Chain-Of-thought Prompting
  - Clarify-AD Refinement
- Experiment Design
- Evaluation Metrics
- Data Analysis & Results
- Conclusion
- Future Work
- References

What is UML?

- UML (Unified Modeling Language) is a standardized modeling language in software engineering.
- Used to visualize, specify, construct, and document software system artifacts.
- Facilitates communication among stakeholders in the development process.
- Supports both object-oriented design and model-driven development (MDD).
- Maintained by the Object Management Group (OMG) as an industry standard.

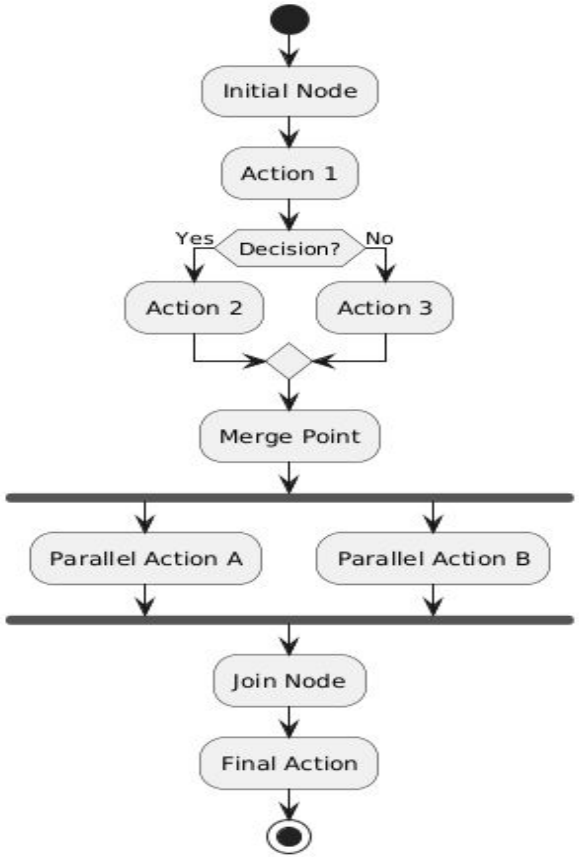
https://www.omg.org/spec/UML/

What is an Activity Diagram?

- A type of UML behavioral diagram used to model dynamic aspects of systems.
- Represents step-by-step workflows of activities and actions, showing how control flows from one activity to another.
- Captures choices (decisions), iterations (loops), and parallel behavior (fork/join) in a single visual language.
- Useful for visualizing use-case logic, **business processes**, **system operations**, and **workflow management**.
- Especially effective for complex, conditional, or concurrent logic that's hard to express in text alone.

Key Elements of an Activity Diagram

- Initial Node
- Activity/Action Node
- Decision/Merge Node
- Fork/Join Node
- Final Node



https://plantuml.com/activity-diagram-beta

Challenges in AD creation before (Manually)

- Time-consuming and repetitive process
- Requires deep understanding of UML semantics
- Prone to human error and inconsistency
- Difficult to maintain and update with evolving requirements
- Ambiguities in natural language can lead to misinterpretation
- Lacks automation and scalability for large systems
- Requires expertise in both domain knowledge and modeling tools

Activity Diagram Generation using LLMs

Advantages:

- Automates extraction of actions, decisions, and flows from natural language.
- Significantly reduces time and manual effort.
- Easy to scale across multiple use cases or systems.

Disadvantages:

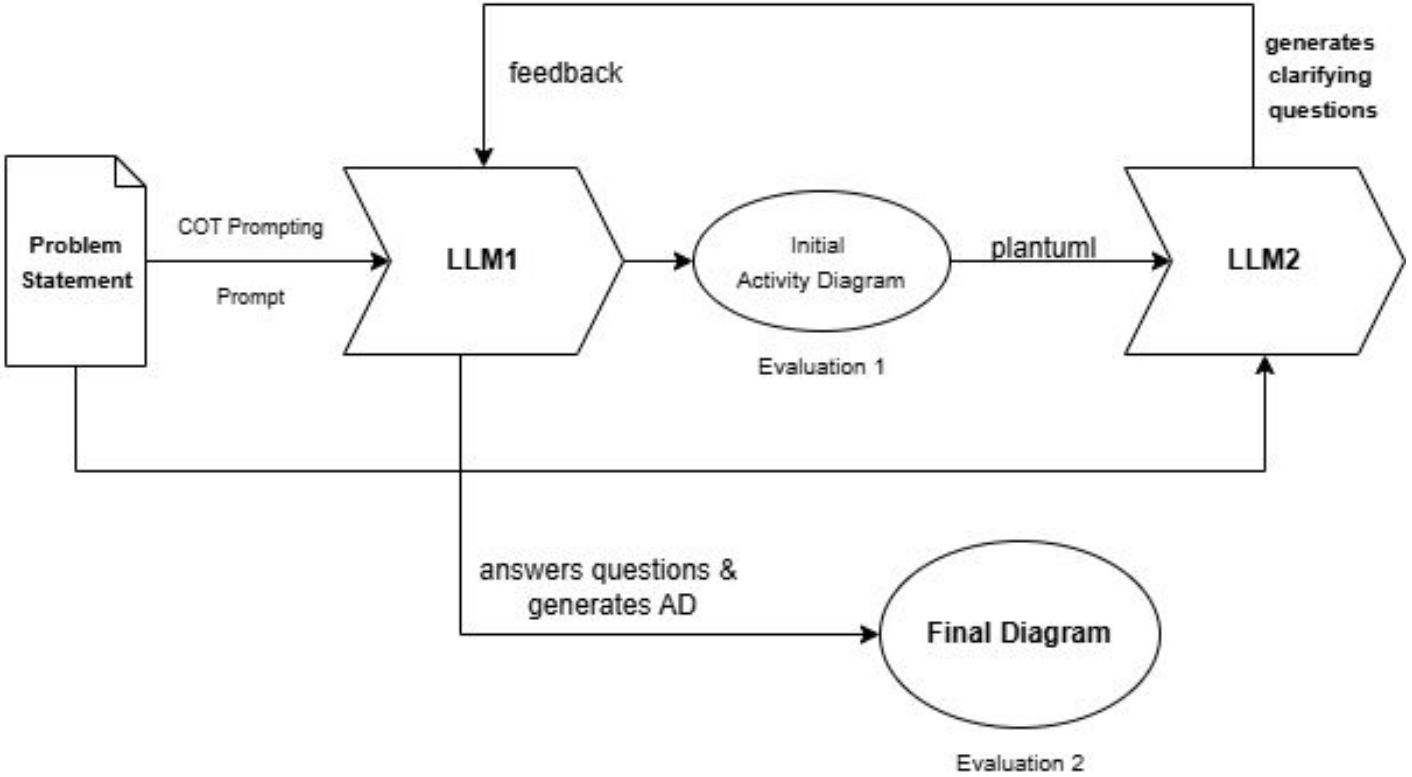
- May misinterpret vague or ambiguous language.
- Generated diagrams might lack **completeness** or **correctness** without validation.
- Difficult to handle complex or domain-specific logic without fine-tuning.
- Requires post-processing or validation in real-world scenarios.

Objectives

- Automatically generate UML Activity Diagrams (ADs) from natural language (NL) problem statements.
- Use Chain-of-Thought (CoT) prompting to extract and represent key elements such as actions, control flows, decisions, concurrency, and final nodes.
- Implement a ClarifyGPT-based feedback loop for iterative refinement and correction of diagrams.
- Ensure **completeness** and **correctness** of generated diagrams through evaluation metrics.

https://ieeexplore.ieee.org/document/10633327 https://dl.acm.org/doi/abs/10.1145/3660810

Methodology



Our Approach

1. Action Identification
2. Control Node Detection
3. Control Flow Construction
4. PlantUML Script Generation

COT

5. Clarifying Question Generation (Evaluator LLM)
6. Diagram Regeneration
7. Evaluation

Clarify-AD

Chain-Of-thought Prompting

- Used to systematically generate UML Activity Diagrams by,
  - identifying actions
  - control nodes (initial nodes, decision, fork/join) and
  - Establishing control flows
- This mimics human reasoning and improve reliability of generated diagrams.
- Enables generation of systematically valid and semantically meaningful PlantUML scripts from NL descriptions.
- Improves clarity, accuracy and practical usability of LLMs in model-driven development.

Clarify-AD refinement

- Evaluator LLM checks for completeness and correctness.
- Generates clarifying questions to address ambiguous, incomplete, or incorrect parts of the diagram.
- Primary LLM regenerates improved diagrams based on user/LLM responses to the clarification questions.
- Enables an interactive and iterative feedback loop, reducing manual debugging or rework.
- Ensures the final diagram aligns closely with the intended system behavior and requirements.

https://dl.acm.org/doi/abs/10.1145/3660810

Experiment Design

Dataset: 10 Problem statements (e-commerce, healthcare, banking, library)

Models: Gemini, LLaMA 3.2

Tools : Python, VS Code, R Studio, Ollama, PlantUml

Metrics: Completeness, Correctness, Standards, Understandability, Terminology

Research Questions:

RQ1 : What is the quality of diagrams generated by LLMs?

RQ2: What common issues arise in generated diagrams?

Evaluation Metrics

- **Completeness** : All required actions, control flows, decisions, and outcomes are included in the diagram.
- **Correctness** : Assesses if the generated elements and their sequencing align with the intended system behavior.
- **Terminology Alignment** : Measures how well the terms used in the diagram match the domain-specific language of the input description.
- **Adherence to UML Standards** : Assesses whether the diagram conforms to formal UML modeling guidelines and best practices.
- **Understandability** : Determines if the diagram can be easily interpreted by stakeholders with varying levels of technical expertise.

https://ieeexplore.ieee.org/abstract/document/10628665

Experiment Design

Null Hypothesis (H_0) :

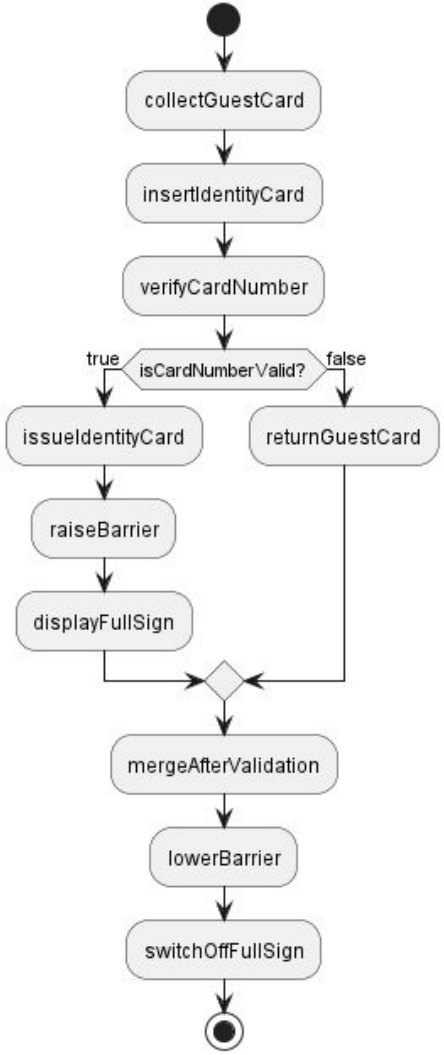
- $Comp_{Before} = Comp_{After}$
- $Corr_{Before} = Corr_{After}$
- $Stand_{Before} = Stand_{After}$
- $Termi_{Before} = Termi_{After}$
- $Under_{Before} = Under_{After}$

Alternative Hypothesis (H_1) :

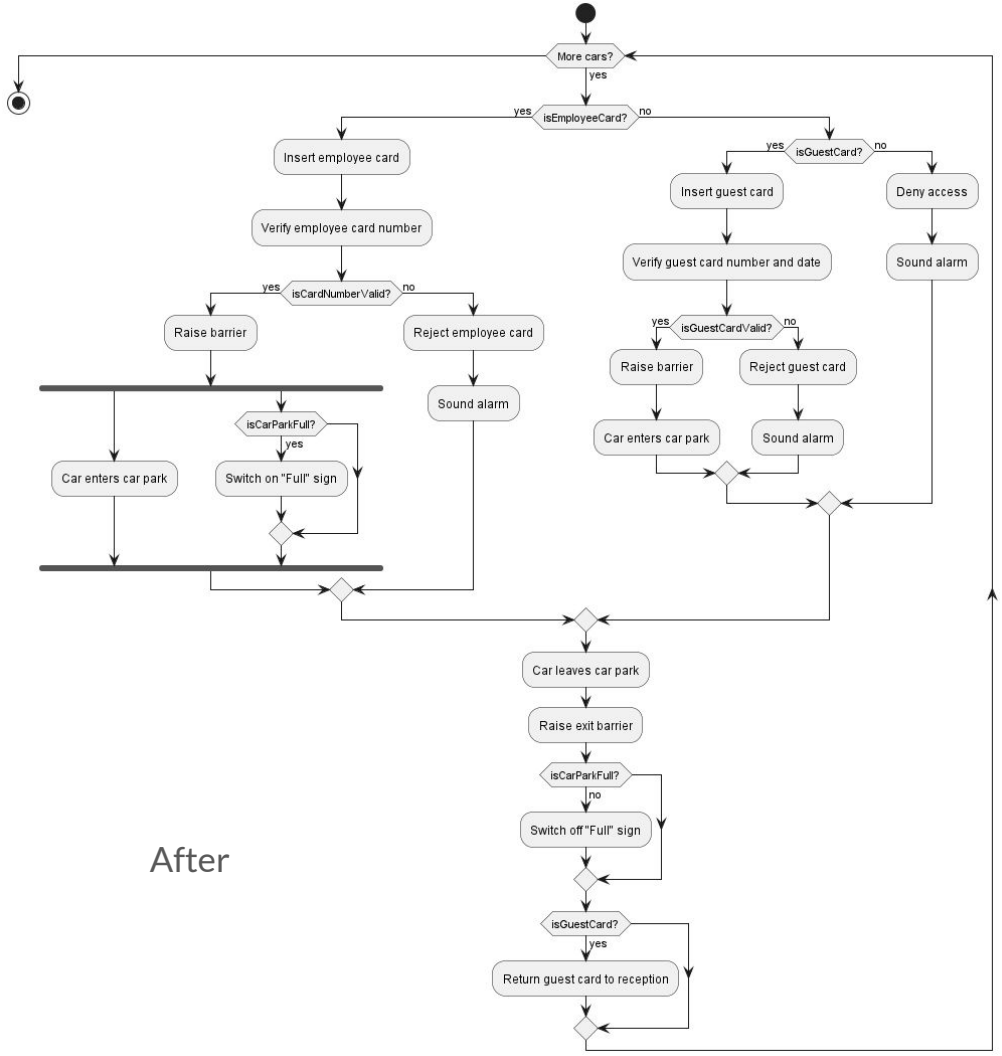
- $Comp_{Before} < Comp_{After}$
- $Corr_{Before} < Corr_{After}$
- $Stand_{Before} < Stand_{After}$
- $Termi_{Before} < Termi_{After}$
- $Under_{Before} < Under_{After}$

H_0 = There is no significant difference between the scores before and after clarification.
H_1 = The scores after clarification are significantly higher than before.

Before



After



Generated Clarification Questions

Clarifying Question - 1
The diagram shows **collectGuestCard** as the first action. Should this be an alternative path, perhaps triggered by a condition like **isGuestCard**?, rather than the primary flow?

Answer
Yes, **collectGuestCard** should be an alternative path, triggered by a condition like **isGuestCard**? or a separate starting point for guest access. The primary flow should focus on employee access.

Clarifying Question - 2
The diagram doesn't differentiate between employee and guest card processing after initial card insertion. How are the different card types handled after verification? Should there be separate paths for employee and guest card processing?

Answer
Yes, separate paths are needed after verification. Employee cards grant ongoing access, while guest cards have a limited validity period. The system needs to check the guest card's expiry date.

Common issues identified

Completeness : Summarization issue, Inconsistent requirements,

Correctness : Incorrect structure, Incorrect interaction

Adherence to Standard : Syntax Errors, Semantic Errors

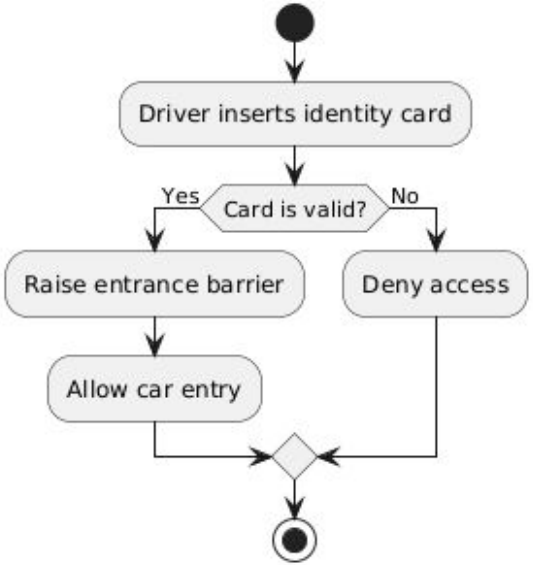
Terminology : Additional Term, Inconsistent Terminology

Understandability : Presence of Redundancy, Actor overrepresentation

https://ieeexplore.ieee.org/abstract/document/10628665

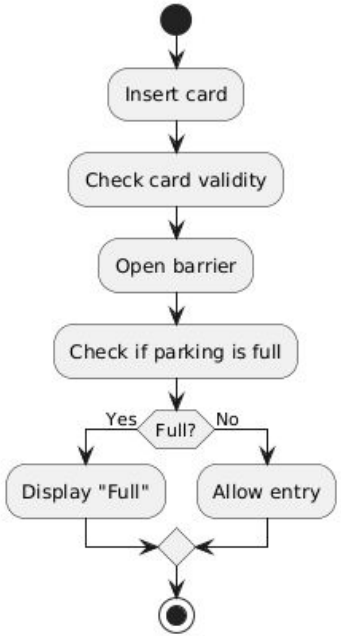
Common issues identified

1. Summarization issue :



Critical Details Lost in High-Level Abstractions

2. Incorrect Structure



"Display 'Full'" happens after access – structurally incorrect (should happen before insertion).

Data Analysis & Results

Problem	Version	Completeness	Correctness	Adherence to Standards	Understandability	Terminology
Problem 1	Before	3	3	5	4	5
	After	4	4	5	4	5
Problem 2	Before	2	3	4	3	5
	After	3	4	5	4	5
Problem 3	Before	2	3	4	3	4
	After	4	4	5	4	5
Problem 4	Before	2	3	5	4	5
	After	4	4	5	5	5
Problem 5	Before	2	3	5	4	5
	After	3	4	5	4	5

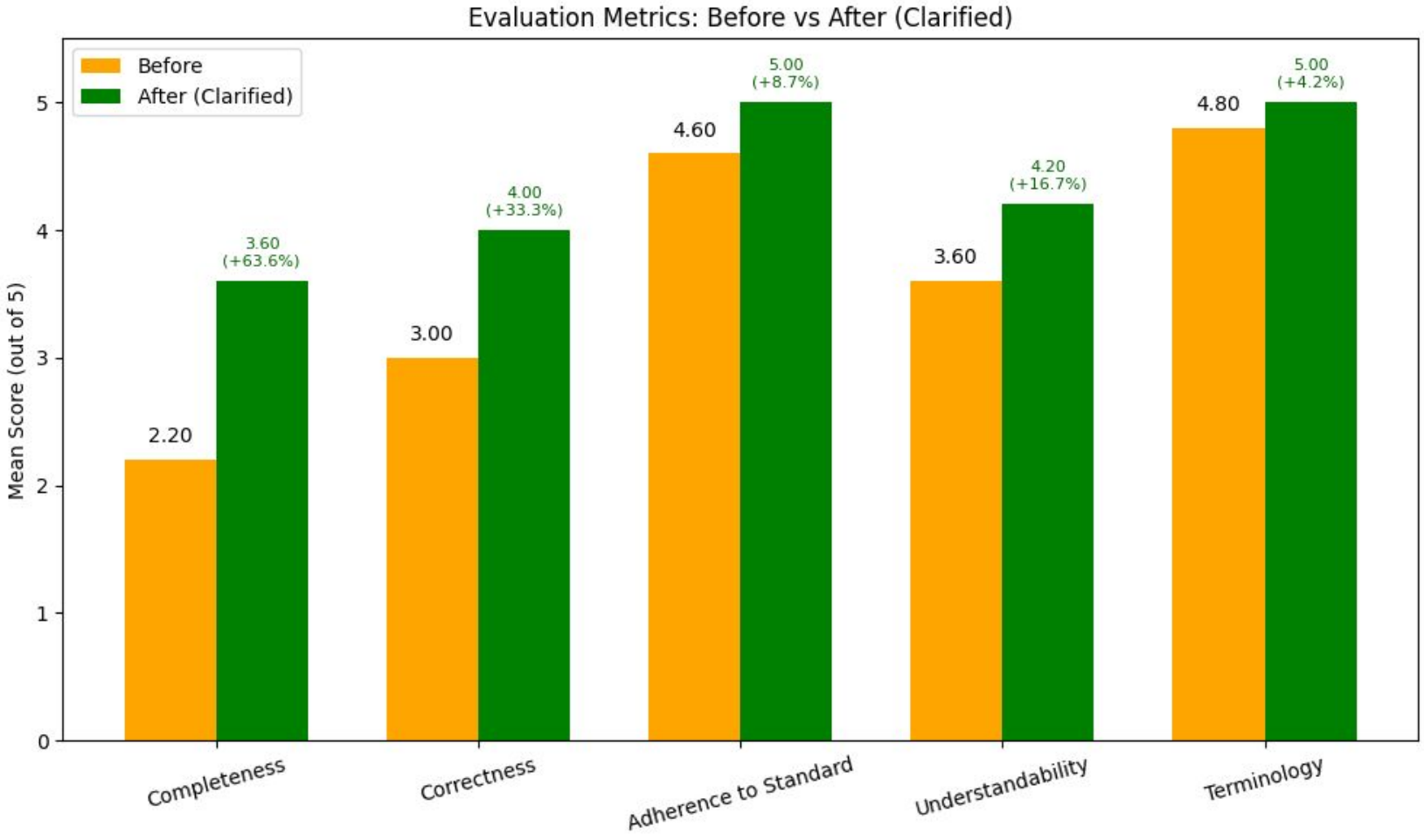
https://ieeexplore.ieee.org/abstract/document/10628665

Data Analysis & Results

% Improvement = (After - Before)/ Before * 100

Metric	Before Mean	After Mean	% Improve-ment	T-Statistic	P-Value	Significance (p < 0.05)
Completeness	2.2	3.6	63.34	5.7155	0.0046	Significant
Correctness	3.0	4.0	33.33	inf	0.0000	Significant
Adherence to Standard	4.6	5.0	8.70	1.6330	0.1778	Not Significant
Understandability	3.6	4.2	16.67	2.4495	0.0705	Not Significant
Terminology	4.8	5.0	4.17	1.0000	0.3739	Not Significant

https://ieeexplore.ieee.org/abstract/document/10628665


Evaluation Metrics: Before vs After (Clarified)
Before
After (Clarified)
Mean Score (out of 5)
0
1
2
3
4
5
2.20
3.60
(+63.6%)
3.00
4.00
(+33.3%)
4.60
5.00
(+8.7%)
3.60
4.20
(+16.7%)
4.80
5.00
(+4.2%)
Completeness
Correctness
Adherence to Standard
Understandability
Terminology

Results

Key Takeaways:

- Completeness and Correctness showed the highest gains, reflecting better alignment with problem requirements.
- Moderate improvements in Understandability and UML adherence indicate better clarity and conformance, though not as dramatic.

These gains highlight the effectiveness of iterative clarification using LLMs in refining diagram structure and content.

Conclusion

- Our clarification-driven approach significantly improved the quality of activity diagrams, especially in Completeness (+63.34%) and Correctness (+33.33%).
- Statistical analysis confirmed these two improvements are highly significant ($p < 0.05$), validating the effectiveness of our method.
- Other metrics (Adherence, Understandability, Terminology) showed moderate gains, though not statistically significant.
- The approach promotes clearer structure, better logical flow, and alignment with the problem statement.
- This method can serve as a foundation for automated diagram refinement pipelines in model-driven development.

Future Work

- Extend the approach to other UML diagrams, such as sequence diagrams, class diagrams, and state machines.
- Fine-tune LLMs on UML and software engineering datasets for improved understanding of modeling conventions.
- Enable real-time user-in-the-loop clarification to guide LLMs during diagram generation.
- Integrate with modeling tools (e.g., StarUML, PlantUML IDEs) for seamless visualization and editing.

References

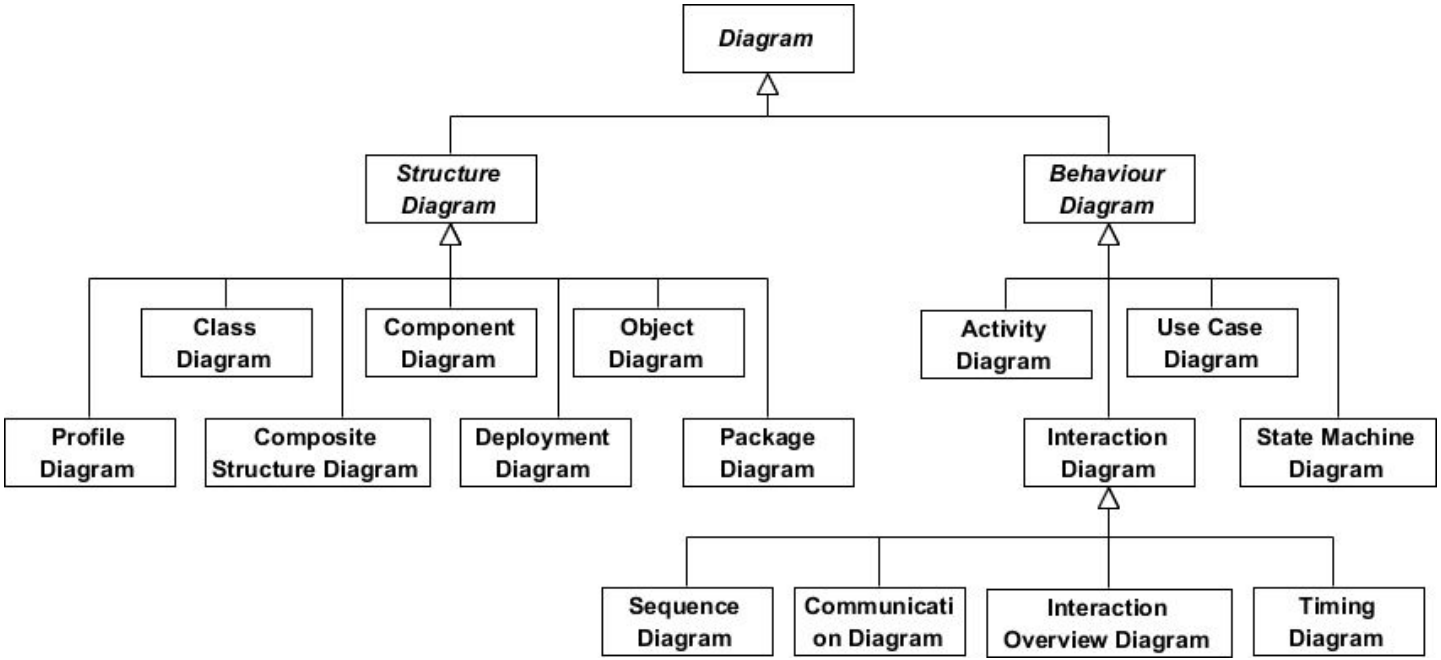
[1] A. Ferrari, S. Abualhaija, and C. Arora, “Model generation with LLMs: From requirements to UML sequence diagrams,” in Proc. of the 32nd IEEE Int. Requirements Engineering Conference (RE), 2024.

[2] Y. Li, J. Keung, X. Ma, C. Y. Chong, and J. Zhang, “LLM-based class diagram derivation from user stories with chain-of-thought promptings,” in Proc. of the 48th IEEE Annual Computers, Software and Applications Conference (COMPSAC), 2024.

[3] F. Mu, L. Shi, S. Wang, Z. Yu, B. Zhang, and C. X. Wang, “ClarifyGPT: A framework for enhancing LLM-based code generation via requirements clarification,” Proc. ACM Softw. Eng., 2024.

THANK YOU

Types Of UML Diagram.

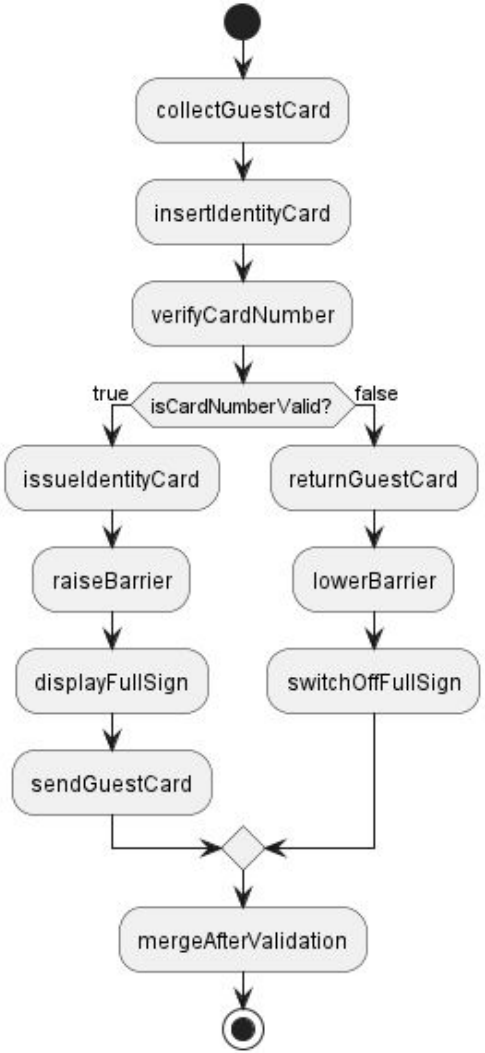


https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

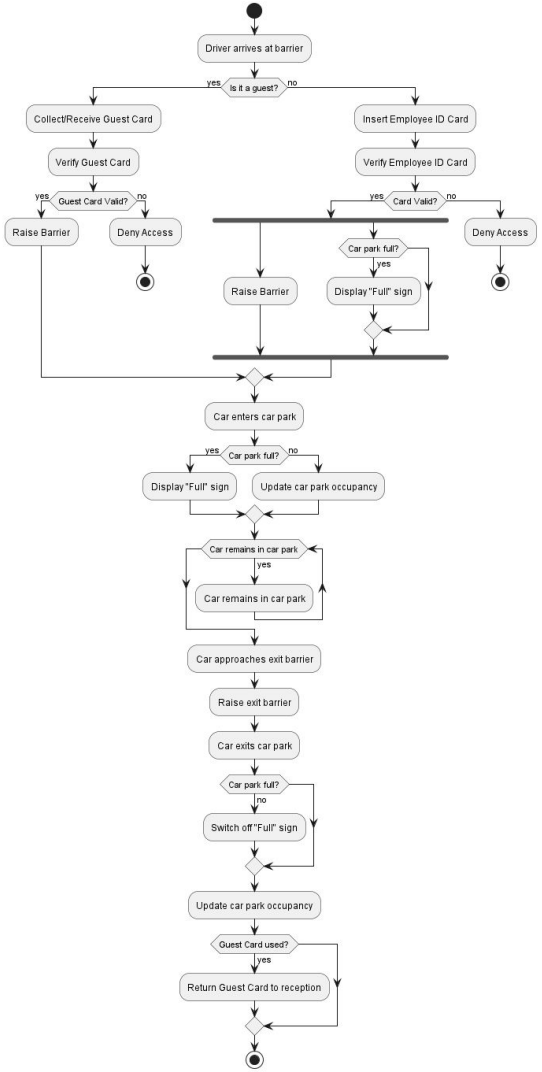
History of an Activity Diagram

-

Before



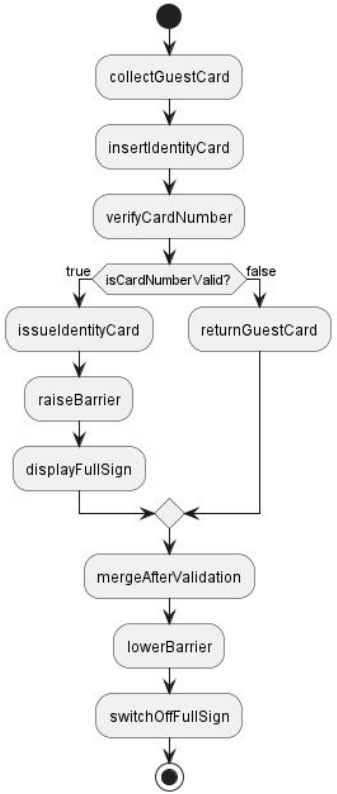
After

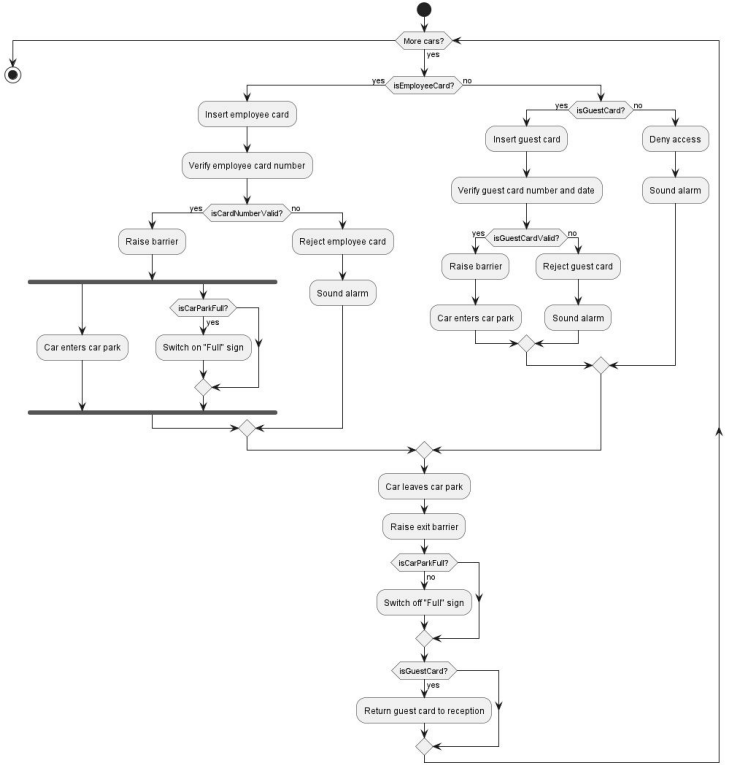


Generated Clarification Questions

Clarification Questions	Answer
The diagram shows **collectGuestCard** as the first action. Should this be an alternative path, perhaps triggered by a condition like **isGuestCard**?, rather than the primary flow?	Yes, **collectGuestCard** should be an alternative path, triggered by a condition like **isGuestCard**? or a separate starting point for guest access. The primary flow should focus on employee access.
The diagram lacks a path for handling invalid employee identity cards. What happens if **verifyCardNumber** fails for an employee card? Should there be an action like denyAccess or soundAlarm?	Yes, a path should be added for invalid employee cards. Actions like denyAccess, soundAlarm, and potentially logInvalidAttempt should be included.
The diagram doesn't account for the exit barrier. How is the exit barrier's behavior modeled? Is there a separate activity or a transition from **mergeAfterValidation**?	The exit barrier's behavior is missing. A separate activity or a transition from **mergeAfterValidation** (or even a separate activity diagram) is needed to model the automatic raising of the exit barrier.
The diagram doesn't differentiate between employee and guest card processing after initial card insertion. How are the different card types handled after verification? Should there be separate paths for employee and guest card processing?	Yes, separate paths are needed after verification. Employee cards grant ongoing access, while guest cards have a limited validity period. The system needs to check the guest card's expiry date.

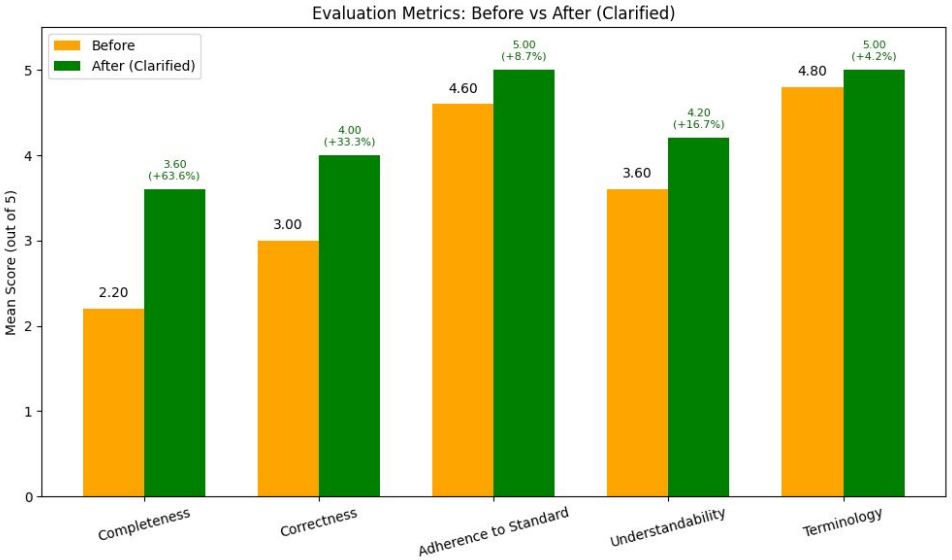
Before vs After – ClarifyGPT-Based Improvement





Data Analysis & Results

Metric	Before Mean	After Mean	% Improvement
Completeness	2.2	3.6	63.64
Correctness	3.0	4.0	33.33
Adherence to Standards	4.6	5.0	8.70
Understandability	3.6	4.2	16.67
Terminology Alignment	4.8	5.0	4.17



https://ieeexplore.ieee.org/abstract/document/10628665