Detailed Project Plan

UML Class Diagram Drawing Using or Assisted by Natural Language Prompts as a Web Based Application

By KWOK Chak Lai (Lefinno), LIU Hao Wei (Roger), WONG Sum Yi (Jenny)

Group fyp24050

1 – Project Background

1.1 Context and Motivation

Effective communication is crucial for any software project, whether it involves clients or internal teams, and diagrams are an excellent tool to facilitate this. Specifically, systems diagrams are a useful means of communicating ideas, proposals, plans, among others. With the use of standardized unified modeling language (UML) rules, these diagrams can be well-understood with little ambiguity (Booch et al., 1999).

Class diagrams, a type of system diagram, are used to model the static structure of a system; they can be used to model the objects within the system, the relationships between these objects, the role of these objects and what services they provide. They are typically drawn with boxes representing classes, which contain the methods and values these classes may contain, along with lines linking classes together and representing their relationships (IBM, 2021). An example of such a class diagram is shown below in Figure 1. The reasons for picking class diagrams specifically for our project will be elaborated on later in Section 2.1.

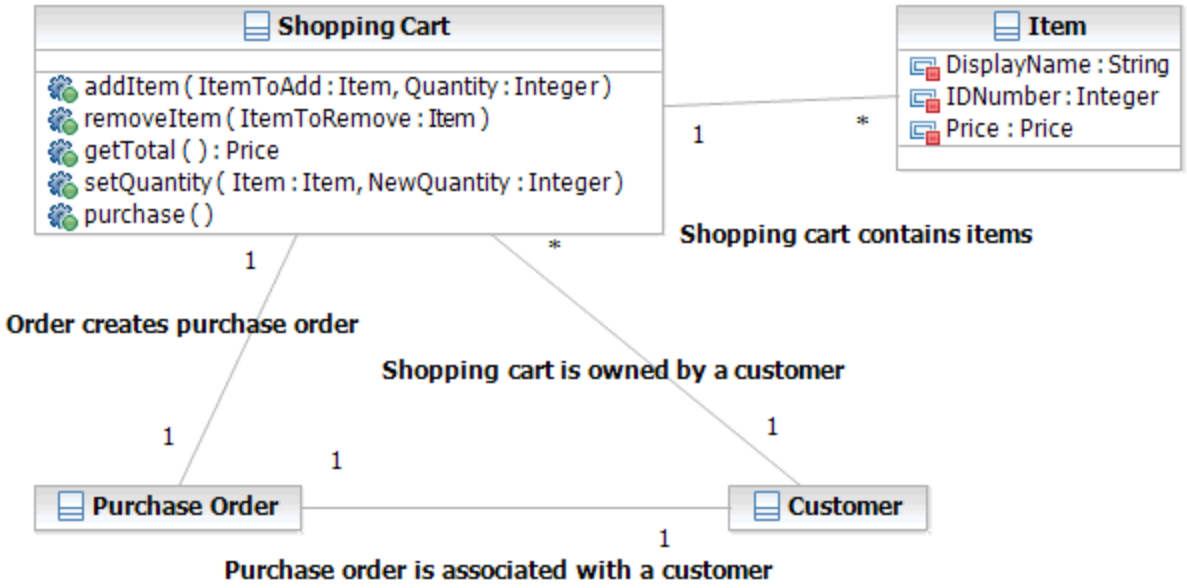

Figure 1: Sample Class Diagram (IBM, 2021)

With current methods, creating such diagrams is usually tedious and time consuming, as the primary means of creating such diagrams is by hand-drawing them, which may lead to these diagrams being drawn as informal images, reducing their usefulness (Conrardy & Cabot, 2024).

1.2 Existing Technology and Studies

The rise in popularity and progression in performance of Large Language Models (LLMs) have been quite prominent in recent years, which has led to their use in more than just text generation and machine translation. In particular, they have shown some promise in image generation, however directly generating a diagram using image generation algorithms is unwise, as they are generally unable to generate graphs directly (Cámara et al., 2023).

However, that is not to say that LLMs cannot be used to create these system diagrams, as LLMs can reliably be used to generate diagrams via Planttext and using PlantUML, which is a text-based diagramming tool. As an example, this is the technique used by Conrardy and Cabot (2024), where the authors used LLMs to turn human hand-drawn picture inputs into Planttext, which was then displayed as a diagram with PlantUML. Their investigation primarily focused on generating relatively simple diagrams in PlantUML that were correct in terms of syntax and determining which of their prompts were the most successful.

Many other research papers, such as Ferrari et al. (2024) and Fill et al. (2023) focus on creating system diagrams with LLMs use a similar method (using Planttext). In addition, this project can streamline a tedious process and provide substantial benefit in that

regard. It is important to note, however, that the studies we have read have all mentioned limitations about the performance of LLMs in this task, with some struggling with correctness and completeness.

1.3 Implementation Philosophy

The goal of this project, as mentioned previously, is to make the creation of systems diagrams a more intuitive and user-friendly process, and better streamline the process of communication between managerial and developmental groups. Outsourcing jobs in development has become more prominent in recent times, and not everything related to a project will be necessarily done in-house, so effective communication via diagrams is vital. As such, it is important to make diagramming accessible to those that may have little background in software engineering with a user-friendly design, though this requires the model itself to be very accurate and precise in drawing a diagram in accordance with the user's requirements, as the potential for miscommunication with a mis-drawn diagram is quite high. Perhaps for the time being, this tool may be used to make diagram drawing less of a hassle for software engineers, as recent research has shown LLMs struggle with accuracy when drawing such diagrams (Cámara et al., 2023).

2 – Project Objective

2.1 Overall Objective

This project's overall objective is to develop a web application that assists users in drawing UML class diagrams by leveraging natural language prompts. Our goal is to streamline the process of software design and development by increasing the efficiency and speed of UML diagramming, meanwhile also improving the communication between all stakeholders involved in the development process, regardless of whether they come from a technical background or not. We also aim to increase the accessibility of UML diagramming by providing a more intuitive way to go about diagram creation and allowing users to create complex class diagrams without needing extensive knowledge of UML syntax or diagramming tools.

As there is a large variety of UML diagrams involved in the software engineering process, it will be hard for us to design and train an LLM that can cover the entire scope of UML diagrams. Therefore, for this project we will be focusing on UML class diagrams which are used to visualize the components within a system and the relationships between them.

The preliminary design of the core framework can be represented in the flow chart below, in Figure 2. This core framework can also be expanded upon to create more natural language interfaces with other purposes.

By the end of this project, we aim to deliver a working prototype of our web application and a comprehensive report on its efficacy, outlining the abilities of our application and its limitations. At the most minimal, we aim to deliver a web application that is able to take in natural language inputs from the user, have a LLM process the prompts and output a UML class diagram that accurately represents the user's input.

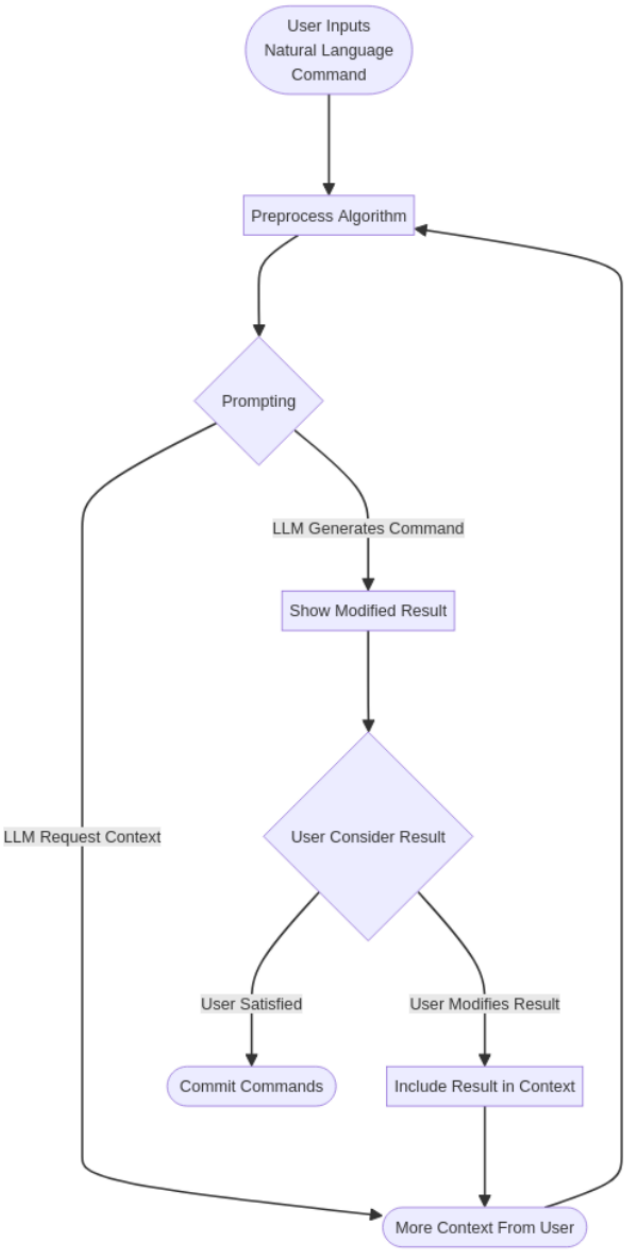

Figure 2: Sample Flowchart of Core Framework

2.2 Incremental Objectives

Our project can be more definitively broken down into the following objectives:

- **Developing a web-based application**: It is first necessary to establish a server framework and create a user-friendly interface that allows users to enter prompts and view their results.
- **Designing and training an LLM for our needs**: Further research needs to be done on LLMs in order to find one that best fits our needs and requirements. After deciding on an algorithm, we may need to develop and train the model, making sure to validate the result of the model and make improvements if necessary.
- **Natural language processing integration**: An algorithm needs to be developed in order to interpret and parse user prompts, process them along with its context, and output corresponding UML class diagram elements such as classes, attributes and relationships.
- **Diagram visualisation**: In order to enable users to visualise the output of the model, we need to create an algorithm that can take in natural language outputs from the LLM and display them on the web application in the form of a UML class diagram.
- **Testing and refinement**: We need to make sure that our code is robust and that it generates responses accurately and in enough detail. Both subjective and objective validation should be done to evaluate the effectiveness of our model such that it can be improved upon.
- **Explore future enhancements**: Once we have established a working prototype, we can use the remaining time to explore and implement additional quality of life features to our application.

By the end of this project, apart from completing the above objectives, we also hope to gain invaluable knowledge and insight into the software development process and the inner workings of large language models.

3 – Project Methodology

3.1 Project Management Style

For this project, considering our non-standard schedules and individual specialties, we adopt an Agile-inspired approach centered around Kanban methodology. This approach balances individual autonomy with team coordination while maintaining a flexible timetable. Key aspects of our project management style include:

1. Flexible task management: Utilizing a Kanban board to visualize workflow and manage tasks asynchronously.
2. Regular check-ins: Brief weekly meetings to align progress, address challenges, and adjust priorities as needed.
3. Collaborative tools: Leveraging version control systems and project management software to facilitate seamless coordination.
4. Incremental deliveries: Focusing on completing smaller, functional components rather than large chunks of work.
5. Adaptive planning: Regularly reassessing project priorities and adjusting our approach based on progress and emerging requirements.

This management style allows us to maintain productivity despite our varying schedules, while ensuring consistent progress towards our project goals. It also promotes clear communication and efficient resource allocation among team members.

3.2 Collaboration Platform and Collaboration Guidelines

For our project, we will utilize GitHub as our primary collaboration platform. This choice aligns with industry best practices and leverages GitHub's robust features for version control, project management, and team collaboration. Our GitHub workflow will be inspired by successful open-source projects and will include the following key elements:

1. Branch Naming Convention:
    a. feature/[feature-name] for new features
    b. bugfix/[bug-description] for bug fixes
    c. hotfix/[issue-number] for critical fixes
    d. release/[version-number] for release preparations
2. Commit Message Format:
    a. Start with a capitalized imperative verb (e.g., "Add", "Fix", "Update")
    b. Keep the first line under 50 characters
    c. Provide a detailed description in subsequent lines if necessary
3. Pull Request (PR) Process:
    a. Create a PR with a clear title and description
    b. Link relevant issues in the PR description
    c. Assign at least one reviewer
    d. Address all comments before merging
4. Code Review Checklist:
    a. Check for code style consistency
    b. Verify functionality and edge cases

    c. Ensure adequate test coverage
    d. Look for potential security issues
5. Release Process:
    a. Create a release branch
    b. Update version numbers and changelog
    c. Conduct final testing
    d. Merge to main and tag the release

3.3 Technology Stack

Go for backend, React for frontend, having client-side rendering, client-side caching and storage, with no login needed. Display logic handled in the front end, user input processing, prompt querying, etc., will be on the backend. We will use WebSocket for our API protocol for now since it is a full duplex session and it's easier for us to do interactive stuff which we need.

Component	Choice	Rationale
Backend Language	Go (Golang)	- Excellent performance for handling concurrent connections - Strong standard library support for HTTP and WebSockets - Efficient resource usage, ideal for scaling - Easy deployment with single binary output - Good support for writing clean, maintainable code
WebSocket Library (Backend)	Gorilla WebSocket	- Well-maintained and widely used in the Go ecosystem - Provides a robust implementation of the WebSocket protocol - Offers good performance and low overhead - Easy to integrate with standard Go HTTP servers
Frontend Framework	React	- Component-based architecture for reusable UI elements - Virtual DOM for efficient updates and rendering - Large ecosystem and community support - Good performance for interactive applications - Easy to integrate with WebSocket libraries
WebSocket Library (Frontend)	react-use-websocket	- React hooks-based library for easy WebSocket integration - Provides a clean API for managing WebSocket connections - Handles reconnection logic and connection status - Compatible with functional components and modern React practices

State Management	React Hooks (useState, useContext)	- Built-in React feature, no additional libraries needed - Simplifies state management for medium-sized applications - Reduces boilerplate compared to Redux for simpler use cases - Easy to use and understand, promoting cleaner code
Client-side Storage	LocalStorage	- Provides persistent storage across sessions - No private project data stored on server side
Communication Protocol	WebSocket	- Full-duplex, real-time communication - Lower latency compared to HTTP polling - Efficient for frequent, small messages - Ideal for interactive, real-time applications - Good support in both modern browsers and server environments
LLM API Integration	Flexible (local or cloud-based)	- Allows for integration with various LLM providers (e.g., OpenAI, Hugging Face) - Flexibility to switch between local and cloud-based solutions - Enables future-proofing as LLM technology evolves - Can be tailored to specific performance or cost requirements
Development Environment	Docker	- Ensures consistency across development environments - Easy to set up and reproduce the development environment - Simplifies deployment and scaling processes - Can include all necessary dependencies in a single container
Version Control	Git with GitHub	- Industry standard for version control - Facilitates collaboration and code review processes - Integrates well with CI/CD pipelines - Provides additional features like issue tracking and project management

3.4 Workflow Definition

In a typical development workflow, the process begins with feature planning and task creation, where new requirements are identified and broken down into manageable tasks. Developers then implement these features, adhering to coding standards and writing unit tests as they go. Once the initial implementation is complete, the code undergoes peer review, where feedback is provided, and improvements are made. Following this, a comprehensive testing phase occurs, including automated unit tests, integration testing, and manual testing to verify functionality and user experience. If bugs are discovered

during testing, they are logged, analyzed, and fixed in a debugging cycle, with fixes being verified through additional testing. Documentation is updated to reflect new features or changes, and the code is then integrated into the main development branch. After successful integration, the new features are deployed to a staging environment for final checks before moving to production. Post-deployment, feedback is gathered, and any necessary adjustments are identified. Throughout this process, there's a focus on continuous improvement, with regular reviews of development processes and implementation of refinements. This workflow provides a structured yet flexible approach to adding features, debugging, and testing, ensuring consistent quality and progress in software development.

3.5 Deployment and Testing Procedures

Development and Local Testing:

Developers work on features and bug fixes, writing and running unit tests as they go. They perform basic functional testing on their local machines to catch obvious issues. Once satisfied with their work, developers commit their code to the shared repository.

Scheduled Testing:

At regular intervals, the tester pulls the latest code from the repository. They set up a fresh Docker environment to ensure consistency and isolation. Within this environment, the tester runs a predefined set of tests, including integration tests, user interface tests, basic performance checks, and any specific tests relevant to recent changes. All issues found during this testing phase are carefully documented.

Bug Review and Planning:

After the testing phase, the entire team comes together to review the bug list compiled by the tester. They discuss each issue, prioritizing bugs based on their severity and potential impact on the project. Bugs are then assigned to developers to be addressed in the next development cycle.

Next Development Cycle:

With bug assignments in hand, developers begin the next cycle. They focus on fixing the identified issues alongside continuing work on new features. This cycle then repeats, starting again with the Development and Local Testing phase.

3.6 Performance Measurement Metrics

Ferrari et al. (2024) mentions that their assessment criteria for their LLM-generated systems diagrams involved "correctness, completeness, adherence to standard, understandability, terminological alignment", and we believe this is a good set of criteria for a system diagram. Wang et al. (2024) goes more in-depth about the criteria of 'correctness', which we will borrow some ideas from.

When assessing the quality of a system diagram, there are several factors that must be accounted for, however the factors that we will use when assessing our model-generated diagrams are as follows:

- Correctness
    - The diagrams generated must adhere to UML modeling rules.
        - This requirement is objective and is thus relatively easy to work with.
- Completeness
    - The diagrams must adhere to the requirements set out by the user
        - It is important to note that this is subjective in nature; what is required by one person may not be the same as another person, as they may have different interpretations of the same requirements. In other words, the generated diagram must convey the same meaning as the intention of the user
        - One solution to the subjectivity issue used by Ferrari et al. (2024) and Wang et al. (2024) is to have the authors of the paper assess the models created and determine if it was successful or not.
- Understandability
    - The diagram must be sufficiently clear, readable and understandable, and should not contain repeated elements if at all possible.
        - Similarly to the previous point, this is also quite subjective in nature, as people will have differing opinions on what is and is not understandable. Because of this, our solution will be to have our group members assess the models created.

There is unfortunately no effective way to assess the more subjective aspects of a system diagram, as individuals will have differing opinions on these criteria, so our solution would be like those used by the authors mentioned in the above papers and have the members of our group provide assessments of our diagrams.

4 – Project Schedule and Milestone

4.1 Project Milestones and Milestone Definitions

To assist with our development, we have broken down our development process into clear phases and created corresponding milestones. We have also separated our milestones into two categories: core and auxiliary. Core milestones are milestones that outline the development of our application, while auxiliary milestones are those that focus on enhancing our application by implementing additional features.

Subsequent to our initial meeting, we have decided upon the following core milestones:

Milestone	Description
MS1	This phase focuses on establishing the framework for our application. This includes exploring hosting and deployment options and setting up our server infrastructure. Some basic API calls should be established to test that the server environment is working as expected.
MS2	This phase focuses on establishing the baseline functionality for our application. At this point, our application should be able to allow a user to input natural language prompts, process them and generate a basic UML class diagram.
MS3	This phase focuses on extending the functionality of our application and enhancing the ability of our LLM. The user should be able to input additional natural language prompts in order to modify, add, link and group classes. Meanwhile, our LLM should be able to utilise spatial context awareness and selective context inclusion in order to enable it to generate more complex UML class diagrams.

Furthermore, the following auxiliary milestones have been proposed as additional features that we can include in our application should time allow:

Milestone	Description
MS4	This milestone focuses on improving interface design such that our application is more intuitive and user-friendly, providing a better user experience overall.
MS5	This milestone focuses on implementing additional functionalities to allow user to edit generated UML class diagrams manually so that they can make small corrections to the diagram while also utilising the LLM to assist them in diagramming when necessary, further enhancing their overall workflow.
MS6	This milestone focuses on improving and refining the LLM for generating our UML class diagrams. Furthermore, additional functionality can be

	added to allow the user to change AI cores and test which model performs the best.

4.2 Project Schedule and Timeline

The following is our preliminary schedule for completing our core milestones:

Milestone	Scheduled Completion Date
MS1	Mid - Late Oct 2024
MS2	Early Jan 2025
MS3	Early - Mid March 2025

As development progresses, the content and completion date of our auxiliary milestones will be adjusted and decided upon.

5 – References

Booch, G., Rumbaugh, J., Jacobson, I. (1999). *Unified Modeling Language User Guide, The* (2nd Edition). Addison-Wesley Object Technology Series.

IBM. (2021). *Class Diagrams*. Retrieved from https://www.ibm.com/docs/en/rsm/7.5.0?topic=structure-class-diagrams.

Conrardy, A., & Cabot, J. (2024). From Image to UML: First Results of Image Based UML Diagram Generation Using LLMs. https://doi.org/10.48550/arXiv.2404.11376

Cámara, J., Troya, J., Burgueño, L., Vallecillo, A. (2023). *On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML*. *Software and Systems Modeling* **22**, 1-13. https://doi.org/10.1007/s10270-023-01105-5

Fill, H-G., Fettke, P., Köpke, J. (2023). *Conceptual Modeling and Large Language Models: Impressions From First Experiments With ChatGPT*. Enterprise Modelling and Information Systems Architecture **18**, 1–15. https://doi.org/10.18417/emisa.18.3

Ferrari, A., Abualhaija, S., Arora, C. (2024). *Model Generation with LLMs: From Requirements with to UML Sequence Diagrams*. https://doi.org/10.48550/arXiv.2404.06371

Wang, B., Wang, C., Liang, P., Li, B., & Zeng, C. (2024). *How LLMs Aid in UML Modeling: An Exploratory Study with Novice Analysts*. https://doi.org/10.48550/arXiv.2404.17739