MGB-EPS drawing tool for UML (Group 6)

by

Bowen Jiang, Matthew Smith, Gleb Myshkin

Stevens.edu

January 2, 2026

Version 1.12.16

MGB-EPS drawing tool for UML (Group 6)

Bowen Jiang, Matthew Smith, Gleb Myshkin
Stevens.edu

This document provides the requirements and design details of the PROJECT. The following table (Table 1) should be updated by authors whenever major changes are made to the architecture design or new components are added. Add updates to the top of the table. Most recent changes to the document should be seen first and the oldest last.

Table 1: Document Update History

Date	**Updates**
12/18/2025	BJ: • Fixed the large font issue
12/16/2025	MS and GM and BJ: • Created and worked on chapter 10.
12/05/2025	BJ and GM: • Write the content for week 12 report (Chapter A).
11/18/2025	GM: • Added Chapter 9. • Added a full Section 9.1 in Chapter 9.
11/13/2025	BJ: • Write the content for Chapter 8. • Add required diagrams into Chapter 8.
11/13/2025	MS: • Write the content for week 11 report (Chapter A). • Write version of doc and created prelimonary UI design.
11/06/2025	MS: • Write the content for week 10 report (Chapter A).
11/04/2025	MS: • Updated document to be dynamically updated
10/31/2025	BJ: • Write the content for week 9 report (Chapter A).

Table 1: Document Update History

Date	Updates
10/24/2025	BJ: • Write the content for Chapter (Chapter 6). • Created Use case and activity Diagrams • Created Use case specification
10/24/2025	MS: • Write the content for week 8 report (Chapter A). • Created all new requirements, and created links for them and the use cases and user stories
10/17/2025	GM: • Write the content for week 7 report (Chapter A).
10/16/2025	BJ and MS: • Worked on the Activity Diagram in chapter 6.
10/09/2025	BJ and MS: • Write the content for week 6 report (Chapter A).
10/07/2025	MS: • Added onto the list of stakeholders and made it complete.
10/02/2025	BJ and GM: • Added Roles and Responsibilities in Introduction • Added chapter (Chapter 4) and put in the information for timeline, development plan, methods use, and the irb information. • Added chapter (Chapter 5) to show which open source tools we'll be using (so far TikZiT and TinyTex have been added)
10/02/2025	MS: • Write the content for week 5 report (Chapter A).
09/25/2025	MS: • Write the content for week 4 report (Chapter A).
09/23/2025	BJ: • Added to bibliography and glossary.
09/18/2025	BJ: • Edit and fix the Overleaf issue with the index and glossary
09/18/2025	MS: • Write the content for week 3 report (Chapter A). • Added the team declaration chapter 3.
09/16/2025	GM: • Updated use case and requirements table to connect them to each other and link them
09/16/2025	BJ: • Updated Glossary and added new terms

Table 1: Document Update History

Date	Updates
09/16/2025	Ms: • Created chapter 3 and added the mission statement, key drivers, and key constraints
09/12/2025	BJ: • Modify and edit the overleaf to complete the week 2 report(Chapter A). • Created Use Case Chapter 2 can start working on the tables for user requirement and use case coverage table
09/11/2025	MSGM: • Write the content for the User Requirement and Use Case Coverage table
09/05/2025	BJ: • Modify and edit the overleaf to complete the week 1 report(Chapter A).
09/04/2025	MSGM: • Write the content for week 1 report (Chapter A).

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Our group is a three-people group and three of us are software engineering majors. Bowen, Gleb and Matthew are all in their 4th year. We are all homebodies who play lots of games, with Matthew's favorite being Pokemon Unite, Gleb's being Minecraft and Bowen's being Yugioh . Matthews goal is to hopefully work in the medical field while still in software, Bowen's goal is to hopefully find a job after graduate working on software field with decent salaries, while Gleb's goal is to be able to create his own startup soon after graduation that could eventually lead to financial stability.

1.1 Roles and Responsibilities

Bowen Jiang
- Development Lead Documentation Editor
- Ensures requirements and use cases are implemented
- Maintains Overleaf documentation, glossary, and bibliography

Matthew Smith
- Architect Test Lead
- Designs system structure and ensures EPS/LaTeX export functionality
- Coordinates testing strategies and validates requirement coverage

Gleb Myshkin
- Buildmeister Risk Management
- Sets up GitHub repo and build pipeline
- Manages scope risks, resource allocation, and coding efficiency

Shared Responsibilities
- Developers, Testers, and Documentation contributors
- Customer representation (acting as target student/programmer users)

Chapter 2

Use Cases

2.1 User Story Definitions

US - 01 *Whenever using Visual Paradigm, I often found myself spending more time wrestling with the interface than creating content. Copying classes could unintentionally duplicate instances, and the tool never clearly explained when to use certain diagram types or blocks. I want a system that is simpler, faster, and provides built-in explanations for its components.*

US - 02 *Using Draw.io makes exporting diagrams inconvenient since downloads are stored in* `.xml` *format. When I need to insert diagrams into documents or slides, I'm forced to take screenshots, which are not editable or scalable. I want an export format like EPS that maintains quality and compatibility with publishing tools.*

US - 03 *As a collaborator, I want to work on the same UML diagram in real time with my teammates so that we can build, modify, and review diagrams together without having to send files back and forth. Collaboration should be smooth and synchronized, reducing confusion and version mismatches.*

US - 04 *When I work on large UML projects, I often make accidental edits and lose previous versions. I want the system to keep track of past versions or changes so that I can easily revert to a previous state without rebuilding the entire diagram from scratch.*

2.2 Use Case Definitions

UC - 01 *A user wants to create a UML diagram and export it as an EPS file. This use case covers the creation, editing, and exporting workflow, including verifying that exported diagrams maintain visual fidelity and correct structure when imported into documents or tools.*

UC - 02 *A new or inexperienced user wants to create a UML diagram while receiving immediate help. The system provides contextual guidance, tooltips, or an interactive tutorial explaining different block types and their relationships as the user constructs a diagram.*

UC - 03 *An experienced user wants to make fine-grained changes to a UML diagram, including modifying or creating custom blocks, adjusting connections, and refining layout. This use case emphasizes control, customization, and efficiency for advanced users who understand UML standards.*

2.3 Requirement Definitions

2.3.1 User Requirements

UR - 01 *The user shall be able to create a sharable file that they can distribute to group mates or collaborators. This requirement ensures collaboration by allowing easy sharing of editable diagram files without exporting or manually sending large data blobs.*

UR - 02 *The user shall be able to export diagrams as Encapsulated PostScript (EPS) files. This allows integration with academic papers, technical reports, and vector-based graphics workflows, ensuring high-quality exports for printing and publishing.*

UR - 03 *The user shall be able to create their own custom UML blocks for reuse in different diagrams. This supports flexibility and modularity in design by letting users define personalized visual elements tailored to their use cases.*

UR - 04 *The system shall provide a dedicated guide or help tab that describes all available blocks, their purposes, and example usage. This improves usability for new users by giving accessible in-application guidance on when and how to use each diagram element.*

2.3.2 System Requirements

SR - 01 ***Main UML Diagram Types Requirement***

The system shall be able to create a display of a UML diagram based on the six main types (Sequence, Object, Class, Use Case, Activity, and State).

SR - 02 ***Drag-and-Drop Requirement***

The system shall have the tools necessary to create UML diagrams using a drag-and-drop interface.

SR - 03 ***New Instance Copying Requirement***
The system shall allow users to copy and modify blocks independently without affecting the original.

SR - 04 ***Direct Block Editing Requirement***
The system shall allow text editing directly on a block rather than through a separate input field.

SR - 05 ***GitHub Synchronization Requirement***
The system shall synchronize with GitHub to allow collaborative editing and version management.

SR - 06 ***Multi-User Collaboration Requirement***
The system shall allow multiple users to collaborate on a single project simultaneously.

SR - 07 ***Version History Requirement***
The system shall maintain a version history to track and retrieve edits made to diagrams.

2.3.3 Non-Functional Requirements

NFR - 01 ***3-Version Storing Requirement***
The system shall be able to store the latest three versions of edits for each project.

NFR - 02 ***Total Storage Footprint Requirement***
The system shall not exceed a total storage footprint of 10 GB, including all user data and cached diagrams.

NFR - 03 ***System Speed Requirement***
The system shall take no longer than five seconds to start and no longer than five seconds to load any saved diagram.

2.3.4 Domain Requirements

DR - 01 ***Prebuilt UML Standards Requirement***
The system shall only provide prebuilt options that comply with UML standards.

DR - 02 ***Licensing Requirement***
The system shall remain under the GPL 3.0 open-source license.

DR - 03 ***Standard UML Notation Requirement***
The system shall follow standard UML notation.

DR - 04 ***EPS Exportation Requirement***
The system shall use EPS format to allow for exportation.

2.4 Constraint Definitions

CR - 01 ***Project Service***
Must be either an application or web based.

CR - 02 ***Hardware Profile Constraint***
Must be able to be processed on basic laptops for macOS and Windows.

CR - 03 ***Export Priority Constraint***
Must allow for EPS conversion above all else.

2.5 User Stories Table

Table 2.1: User Stories Table

User Story ID	Story	Linked Requirement(s)
US-01	**Visual Paradigm Pain Points**	UR-03, UR-04, SR-04, SR-03, DR-03
US-02	**Draw.io Export Friction**	UR-02, UR-04, DR-04
US-03	**Real-Time Collaboration**	UR-01, SR-05, SR-06
US-04	**Version History**	UR-01, SR-07, NFR-01

2.6 Use Case Coverage Table

Table 2.2: Use Case Coverage Table

Use Case	Description	Covered Requirements
UC-01	**Create & Export** A user wants to create a UML diagram and export to EPS.	SR-01 SR-02 SR-04 SR-07 UR-01 UR-02 DR-03 DR-04 NFR-01 NFR-03
UC-02	**Guided Creation** A user is new to creating UML diagrams and wants to create one while getting immediate help.	SR-01 SR-02 SR-04 UR-04 DR-01 DR-03
UC-03	**Advanced Editing** An experienced user wants fine-grained control over blocks and other advanced features.	SR-01 SR-03 SR-05 SR-06 SR-07 UR-01 UR-03 DR-01 DR-02 NFR-01 NFR-02 NFR-03

2.7 User Requirements Table

Table 2.3: User Requirements Table

ID	User Requirement Definition	Priority	Use Case / Story
UR-01	**Shareable File Requirement** *The user shall be able to create a sharable file that they can share with group mates.*	Must	UC-01 UC-03 US-03

Table 2.3: User Requirements Table

ID	User Requirement Definition	Priority	Use Case / Story
UR-02	**EPS Exportation Requirement** *The user shall be able to export diagrams as EPS.*	Must	UC-01 US-02
UR-03	**UML Block Creation Requirement** *The user shall be able to create their own blocks to be used in UML diagrams.*	Should	UC-03 US-01
UR-04	**Info Guide Requirement** *The system shall have a guide tab that describes the different "blocks" of the drag-and-drop system, including when to use them and example usage.*	Should	UC-02 US-01

2.8 System Requirements

Table 2.4: System Requirements Table

ID	System Requirement Definition	Priority	Use Case / Story
SR-01	**Main UML Diagram Types Requirement** *The system shall be able to create a display of a UML diagram based on the six main types (Sequence, Object, Class, Use Case, Activity, and State).*	Must	UC-01 UC-02 UC-03
SR-02	**Drag-and-Drop Requirement** *The system shall have the tools necessary to create UML diagrams using a drag-and-drop interface.*	Must	UC-01 UC-02
SR-03	**New Instance Copying Requirement** *The system shall allow users to copy and modify blocks independently without affecting the original.*	Should	UC-01 UC-03 US-01
SR-04	**Direct Block Editing Requirement** *The system shall allow text editing directly on a block rather than through a separate input field.*	Should	UC-01 UC-02
SR-05	**GitHub Synchronization Requirement** *The system shall synchronize with GitHub to allow collaborative editing and version management.*	Should	UC-03 US-03
SR-06	**Multi-User Collaboration Requirement** *The system shall allow multiple users to collaborate on a single project simultaneously.*	Would	UC-03 US-03
SR-07	**Version History Requirement** *The system shall maintain a version history to track and retrieve edits made to diagrams.*	Would	UC-01 UC-03 US-04

2.9 Non-Functional Requirements

Table 2.5: Non-Functional Requirements Table

ID	Non-Functional Requirement Definition	Priority	Use Case / Story
NFR-01	**3-Version Storing Requirement** *The system shall be able to store the latest three versions of edits for each project.*	Should	UC-01 UC-03 US-04
NFR-02	**Total Storage Footprint Requirement** *The system shall not exceed a total storage footprint of 10 GB, including all user data and cached diagrams.*	Would	UC-03
NFR-03	**System Speed Requirement** *The system shall take no longer than five seconds to start and no longer than five seconds to load any saved diagram.*	Must	UC-01 UC-03

2.10 Domain Requirements

Table 2.6: Domain Requirements Table

ID	Domain Requirement Definition	Priority	Use Case / Story
DR-01	**Prebuilt UML Standards Requirement** *The system shall only provide prebuilt options that comply with UML standards.*	Must	UC-02 UC-03 US-01
DR-02	**Licensing Requirement** *The system shall remain under the GPL 3.0 open-source license.*	Must	UC-03 US-03
DR-03	**Standard UML Notation Requirement** *The system shall follow standard UML notation.*	Must	UC-01 UC-02 US-01
DR-04	**EPS Exportation Requirement** *The system shall use EPS format to allow for exportation.*	Must	UC-01 US-02

2.11 Constraints

Table 2.7: Constraint Table

Constraint	Description
CR-01	**Project Service** Must be an application or web based.
CR-02	**Hardware Profile Constraint** Must be able to be processed on basic laptops for macOS and Windows.
CR-03	**Export Priority Constraint** Must allow for EPS conversion above all else.

Chapter 3

Team Declaration

3.1 Mission Statement

Our project aims to allow students and programmers to easily create uml style Diagrams, and export them as an eps file ready to be used however the user sees fit. This will be done by creating an app that allows the user to drag on specific "Blocks", connect, and edit them to display whatever the developer needs them to display for their project. The main focus of this project is to tackle the lackluster options that handle uml Diagram creation, creating an easier to use system that handles dynamic needs, ensuring that users have a comfortable time creating uml Diagrams, being adaptable for those in the know, and allowing those who aren't knowledgeable to have a guide to help them practice this vital skill.

3.2 Key Drivers

For the key drivers, these are our motivation for making this project, and are the basis for many of our requirements. The key drivers are a hard to navigate system for most others, limited options, lack of eps conversion, and a desire to make it useful for those new to uml.

- Hard navigation systems. For most uml based applications they have features or mechanics that are not standard among other applications, and as such make them more difficult or annoying to use then is needed. For example, when using visual paradigm, when you copy something (such as a class), you are making a second literal copy, meaning changing one changes the other. This is antithetical to copying in most instances, since usually when one wants to copy, they want something closely the same bu not exactly. Basically, we feel these alternatives are not as intuitive or comfortable to use as can be, and how system should be able to adress this.

- Limited options. While draw.io does allow for custom Blocks, most other systems don't have an easy way to create your own Blocks, let alone save them for later use. This means that those creating new systems entirely, or those who have specific things they want to adress, cannot do so as easily. Having the ability to create their own Blocks allows for those with these specific needs to have them met and even reuse them.

- Lack of eps conversion. This is rather straightforward, most of these systems don't allow you to create an eps file directly, and instead require you to convert from jpeg or png to eps, which is a hassle and leads to messy conversion issues. Our system will allow for a direct eps conversion, which will be useful for those using latex in particular.

- Useful for those new to uml. When we were starting out, uml seemed a lot more complicated, and while we had the tools, we didnt know how to use them. Having a built in guide, with an example of what it does and how to show it, would have made our lives a lot easier, especially if it had code-to-uml examples. We want to include a guide in order to ensure new people are encouraged to use our system, and gives them a starting-off point.

3.3 Key Constraints

For the key constraints, these are the main things we have to take into account when creating our system. Essentially, they are the limitations, including our manpower, processing power, and our key focus.

- Manpower constraints. Our group is made up of three people. This is not necessarily a small amount, but its not a lot either. Most other groups are 4 or even 5, which means we have about 75-60 percent of the man-hours as other groups. This limits the scope we can attain with this project, and means we must be very careful with scope creep.

- Processing power. Given that we are three students, we don't have access to the cutting edge tech that might allow for the app to have sloppy, inefficient code. Our code must be made more efficient than most, in order to ensure our application works for all of us. Doubly so given our group has both windows and mac, our application will have to be able to work with most operating systems.

- Key focus. Since this is an assigned project, specifically about uml and eps, our system priority will always be focused on that. This means that any decision, of it comes down to those two factors or something else, will always prioritize making it work in eps and have the correct uml Diagrams. Essentially, our focus is constrained to starting with that as the baseline, even if it goes against the natural course of the project, or the potential consumers wishes.

Chapter 4

Development Plan

4.1 Timeline

Phase 1 – Documentation Design
- Defined user requirements, use cases, and constraints
- Created glossary and bibliography

Due October 5th
Goal: Completed document

Phase 2 – Research Prototyping
- Studied existing tools (Umlet, Draw.io, Tikzit) – C++,Javascript,etc
- Recompiled Tikzit and integrated TinyTeX for EPS/LaTeX support
- Began user story collection

Due October 16th
Goal: Software prerequisites up and running

Phase 3 – Implementation
- Build core drag-and-drop UML editor
- Enable EPS/PNG/JPEG export
- Add GitHub sync and collaboration features

Due May 1st
Goal: Working prototype (with all musts implemented)

Phase 4 – Testing Refinement
- Validate requirements coverage with use cases
- Optimize performance for cross-platform use

Due Date To Be Determined
Goal: Finished, working code

The timeline is subject to change due to outside factors and is tentative

4.2 IRB Protocol

There is no IRB protocol necessary for this project, as this project is strictly based on software.

4.3 Agile Method

Iterative Process - Work divided into weekly reports, each producing tangible progress
- We plan on having weekly meeting and iterative workloads

User Centered - Requirements captured as user stories to guide development
- Requirements are based around user stories

Collaboration - Team of 3 shares tasks via Overleaf and GitHub, ensuring transparency
- Weekly meetings will go over everyone's tasks

Flexibility – Adjust scope and priorities based on weekly findings and challenges
- Some things will be adjusted as time goes on, with some things being decided on later to avoid ballooning costs

Continuous Improvement - Each sprint includes documentation updates, and bug fixes,before moving forward
- Our weekly reports will emphasize what was done and needs to be done to encourage continuous improvement

4.4 Method

The information below will be about our project.

4.4.1 Software

- C++ with help of TinyTex being R or Powershell. Python for creating automation scripts.
- Mac OS and Windows as operating systems (possibly Linux but not fully looked into)
- TikZiT and Tinytex for GUI and TikZ as the package (more added in the future)
- Laptop/PC (Software Only)

4.4.2 Review Process

Review Process to be determined throughout the project.

4.4.3 Build Plan

Currently using code from two tools listed above. Compiled separately from their GitHub pages, but will work on combining into one compilation to get both the GUI and compiler.

4.4.4 Workspace

- Laptops or Desktops used for working on the code (we have both a Windows and Mac operating system to test both operating systems)
- Software engineering lab for work space, but some work will be virtual.
- GitHub for all the code. Jira for Kanban Boards and task tracking. VS Code mainly used for actually modifying the code

4.4.5 Modification Request Process

Will be using Jira and GitHub for requesting and showing any problems that we find and add priorities to the issues. Along the project, we'll be adding any new ideas that we want in our project using Jira and GitHub.

- GitHub: used for code issues
- Jira: Anything else that we need to work on in the project

4.4.6 Stakeholders

- UML Creators
- Software Engineering Students/Professors
- Software Businesses
- Original creators of the Tikzit
- Original creators of the Tiny Text
- Open source software people

4.4.7 Communication Plan

- Communication between team members with the use of SMS or emails
- Use of Email for stakeholders.
- No current regime of communications between stakeholders (no planned dates or time for communication)

Chapter 5

Software Tools

5.1 TikZiT

TikZiT is an open source GUI tool that allows a user to create their own diagrams that will be saves as a .tikz file and then either exported to Ovearleaf or the code is taken and pasted into Overleaf. This is the main tool that we'll be looking into incorporating into our project as it allows us to already draw some basic diagrams with .tikz functionality. What we'll have to do is modify it so that it fits out requirements.

The following is a list of some of its features:

- Custom Nodes and Lines
- Modify Code if Needed
- Simple Diagrams
- Latex Code in Diagrams
- Preview of Diagram (Latex Compiler Required)
- Potential automation with OverLeaf (file saves)

Example uses of TikZiT will be added later.

5.2 TinyTex

TinyTex is a latex compiler that is based on a bigger latex compiler TeX Live. The reason why we went with TinyTex is its small storage installation size, compared to the full TeX Live or other compilers. The reason why we need a Latex compiler is so that we can preview our diagrams while we are working on them in TikZiT. It might also be helpful in converting our diagrams into the EPS format.

The following is a list of some of its features:

- No Real GUI

- Small Installation for a compiler (350 MB)
- Helps with Latex compiling, especially for TikZiT
- Has built in Command Prompt
- Might have some issues with installation for TikZiT compiler to work, easy to fix, but not yet automated
- Might integrate directly into TikZiT for combined download

Chapter 6

Activity Diagrams And Use Case Diagrams

6.1 Use Case Diagram for Use Case UC-01

Linked to the use case: UC-01

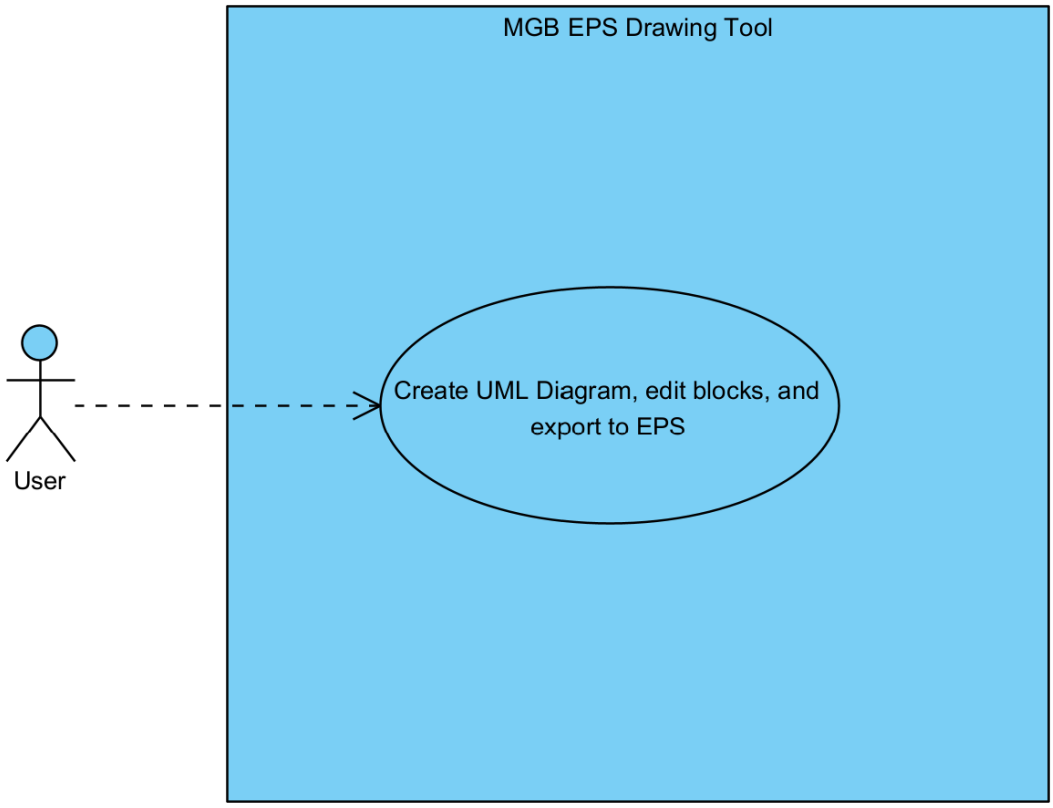

Figure 6.1: Use Case Diagram for UC-01

6.2 Use Case Diagram for Use Case UC-02

Linked to the use case: UC-02

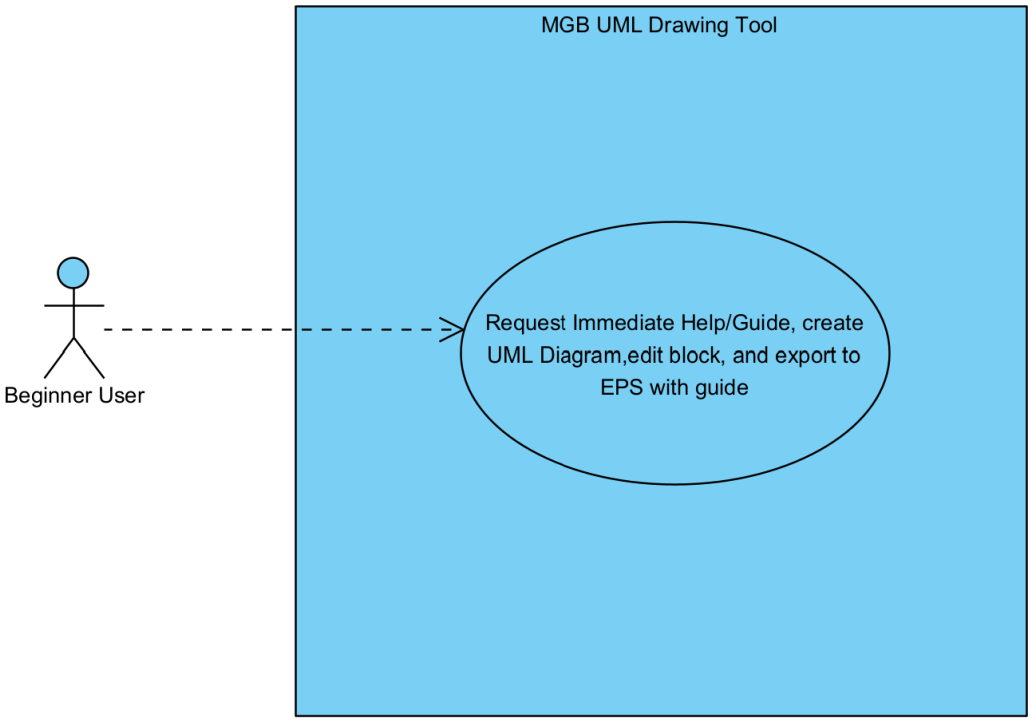

Figure 6.2: Use Case Diagram for UC-02

6.3 Use Case Diagram for Use Case UC-03

Linked to the use case: UC-03

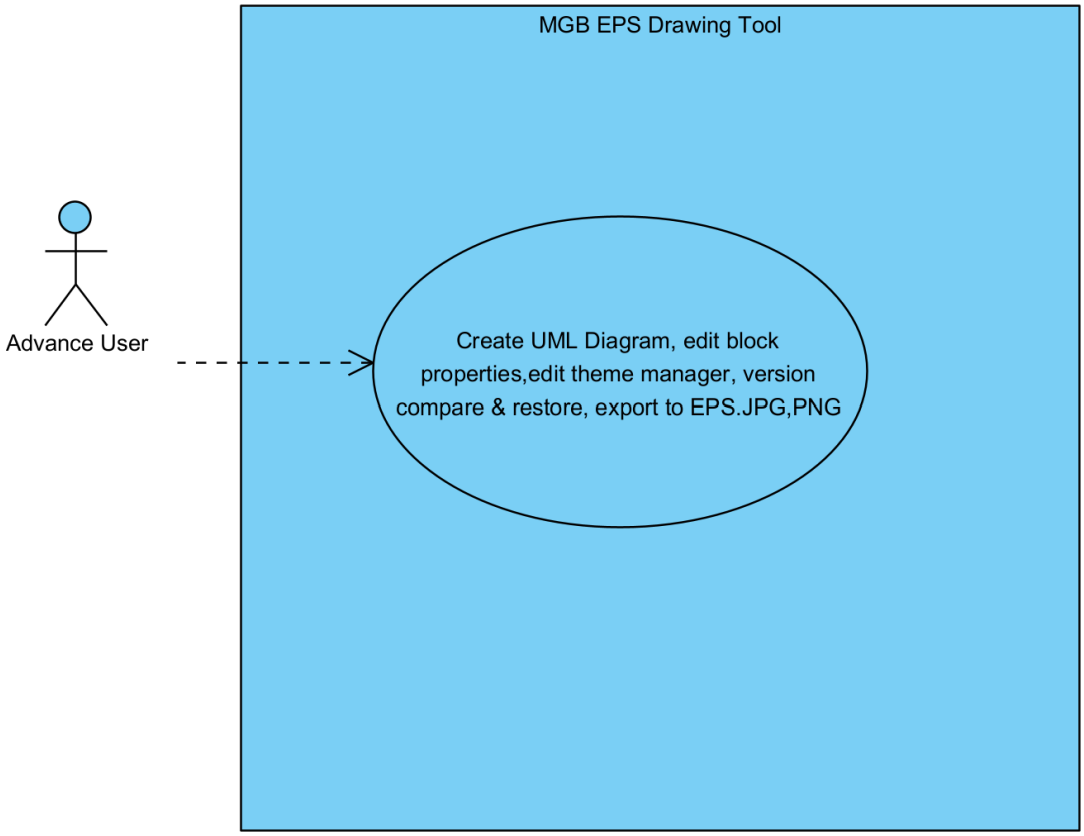

Figure 6.3: Use Case Diagram for UC-03

6.4 Activity Diagram For Use Case UC-01

Linked to the use case: UC-01

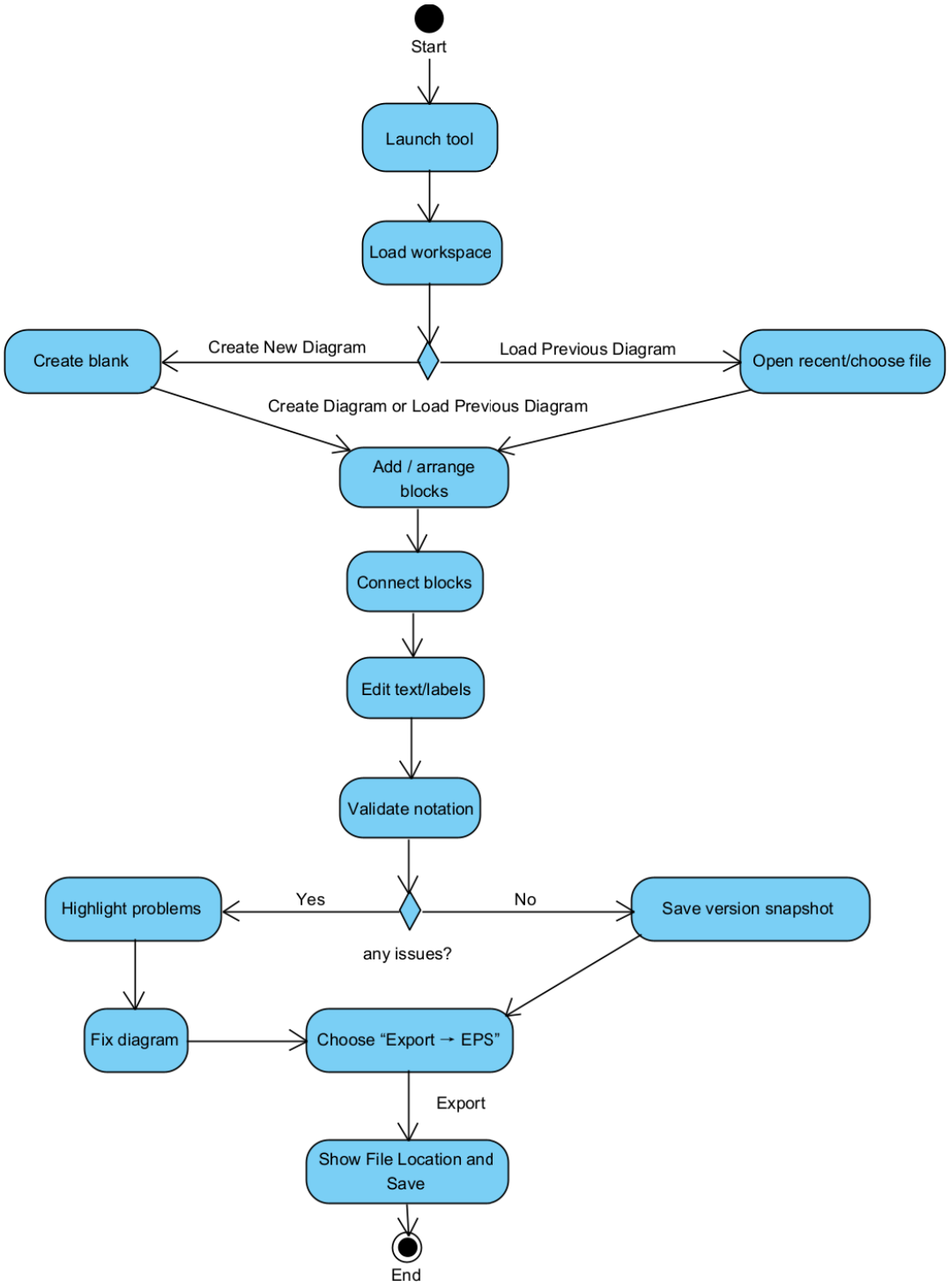

Figure 6.4: Activity Diagram for UC-01

6.5 Activity Diagram For Use Case UC-02

Linked to the use case: UC-02

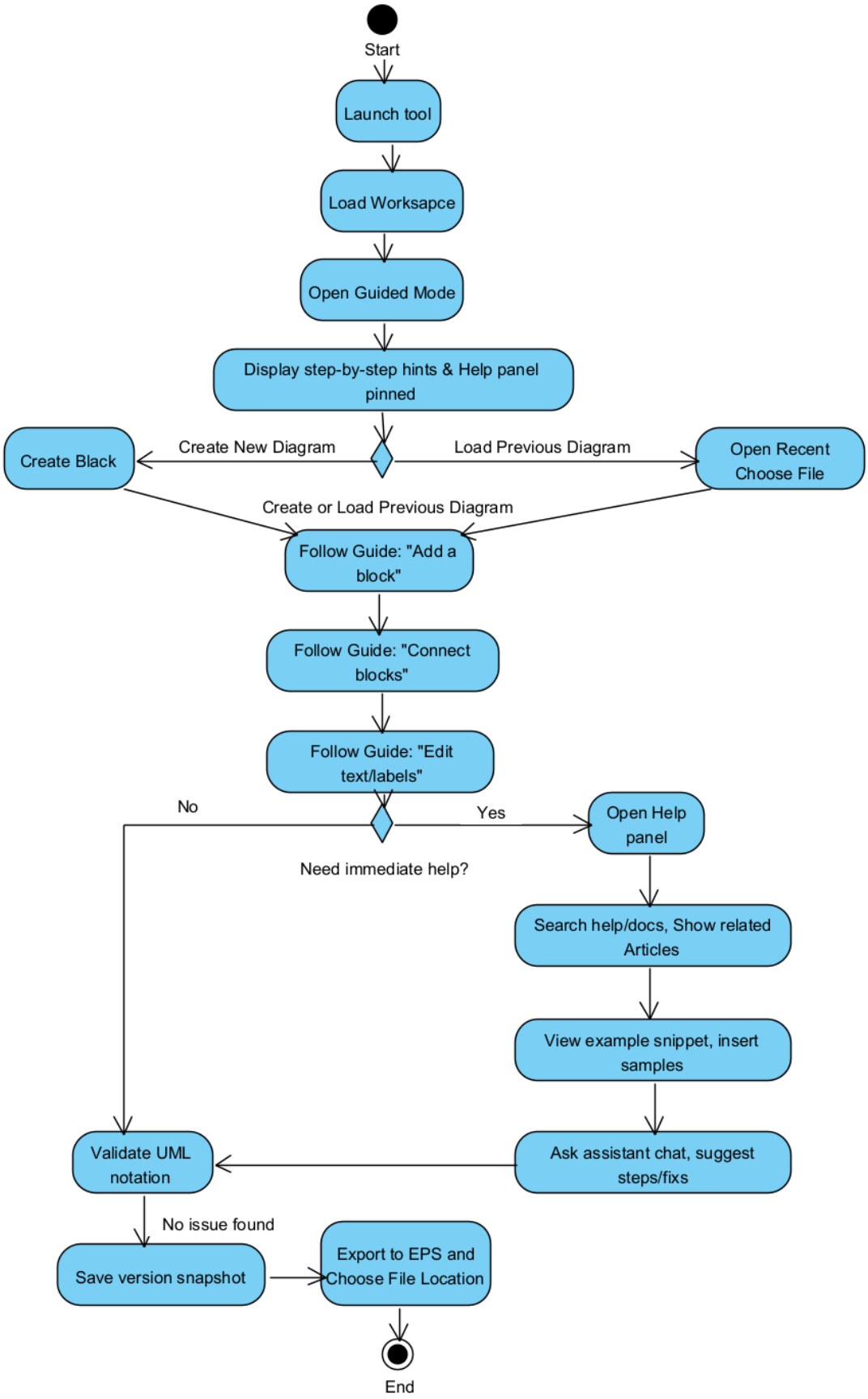

Figure 6.5: Activity Diagram for UC-02

6.6 Activity Diagram For Use Case UC-03

Linked to the use case: UC-03

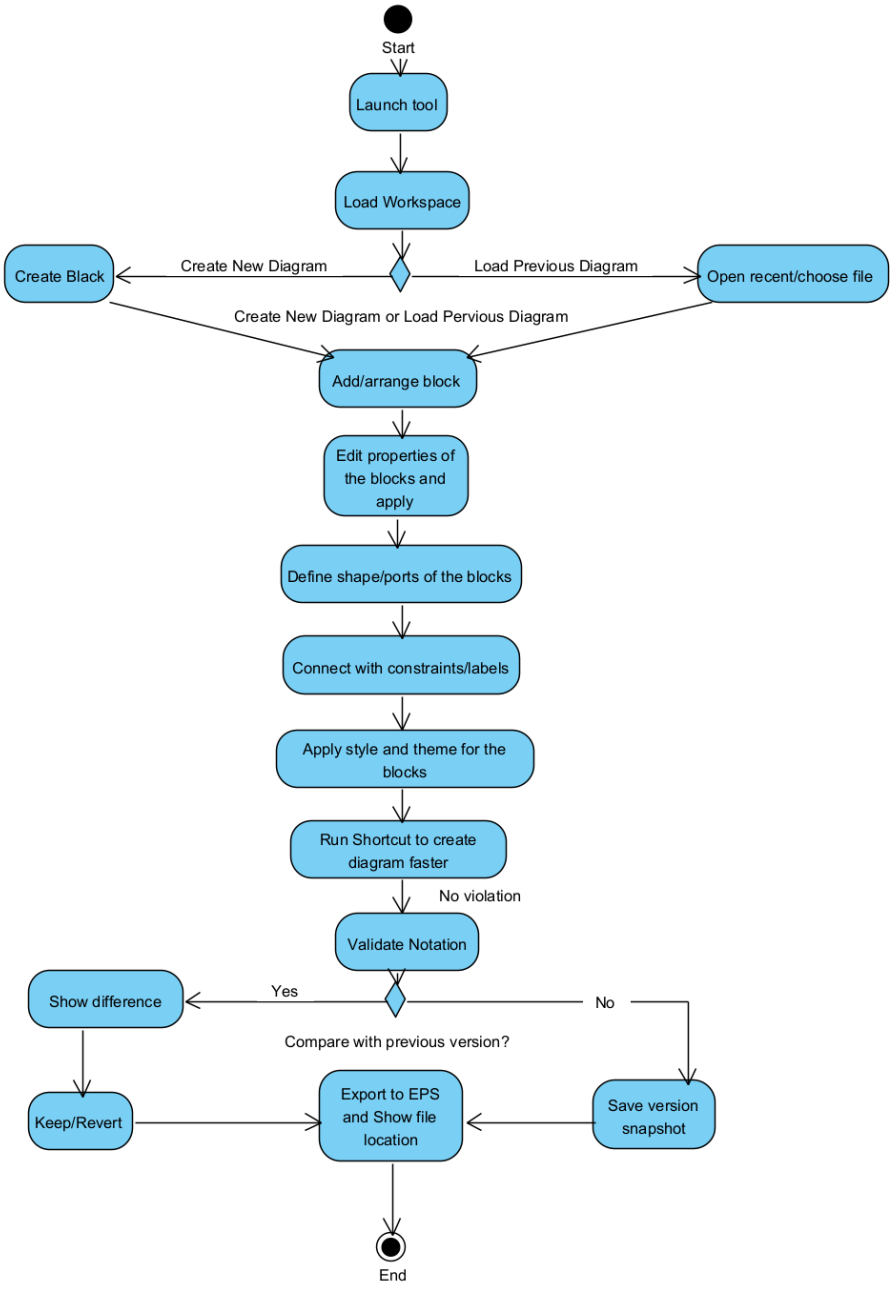

Figure 6.6: Activity Diagram for UC-03

6.7 Use Case Specifications For Use Case

Use case: Create UML Diagram and Export to EPS	
Use case identifier	UC-01
Brief description	A user launches the UML tool, creates (or opens) a diagram, and exports the result to an EPS file.
Primary actors	User
Secondary actors	File System
Preconditions	1. Application is available and starts successfully. 2. User has permission to create and save files. 3. Workspace is initialized.
Main flow	1. User launches the tool. 2. System loads the workspace and shows *New* or *Open*. 3. User chooses *New* diagram or load diagram. 4. User adds/arranges blocks, connects them, and edits text. 5. System validates UML notation. 6. System creates a version snapshot. 7. User chooses *Export* → *EPS*. 8. System confirms success and shows the file location.
Postconditions	1. EPS file exists at the chosen location. 2. Latest diagram state is saved (version snapshot updated).
Alternative flows	**Open existing** — In Step 3 the user selects *Open*; the system loads a prior diagram, then continue at Step 4. **Validation issues** — In Step 5 the system highlights problems; the user fixes the diagram and re-validates, then proceed to Step 6. **Export failure** — In Step 8 rendering or write fails; the system shows an error and the user retries or cancels.
Related requirements	SR-01, SR-02, SR-03, SR-04, SR-07; UR-01, UR-02; DR-03, DR-04; NFR-01, NFR-03

Table 6.1: Use Case Specifications For Use Case UC-01

Use case: Create UML Diagram with Guided Help (UC-02)	
Use case identifier	UR-02
Brief description	A novice user creates a UML diagram using *Guided Mode* with contextual hints and immediate help (docs/examples/assistant).
Primary actors	New User
Secondary actors	Help/Docs Service, Assistant Chat, File System
Preconditions	1. Application launches successfully; workspace is initialized. 2. Help content is available (local cache or network).
Main flow	1. User launches the tool; system offers *Guided Mode*. 2. System enables step-by-step hints and pins the Help panel. 3. User chooses to start from a *Template* or load diagram 4. Guided hint: add and arrange blocks. 5. Guided hint: connect blocks. 6. Guided hint: edit text/labels. 7. System validates UML notation (non-blocking issues shown inline). 8. System saves a version snapshot. 9. User can finish or continue editing (export optional, see UC-01).
Postconditions	1. A valid diagram exists and is saved. 2. The user has received contextual guidance for key steps.
Alternative flows	**Switch to Standard** — At Step 1 the user selects Standard mode; proceed with UC-01 main flow. **Immediate Help** — At any step the user opens Help: search docs, view example snippet, or ask assistant; resume at the current step. **Validation Issues** — At Step 7 the system highlights problems; user applies fixes; re-validate, then continue to Step 8. **Help Unavailable** — If online help is unreachable, system falls back to local tips; user may retry later.
Related requirements	SR-01, SR-02, SR-04; UR-04; DR-01, DR-03.

Table 6.2: Use Case UC-02: Guided creation with immediate help

Use case: Advanced Create & Modify with Pro Features (UC-03)	
Use case identifier	UC-03
Brief description	An experienced user tailors a UML diagram with advanced controls (custom blocks, precise connections, styles, auto-layout, macros), validates it, and optionally exports (EPS/JPG/PNG).
Primary actors	Experienced User
Secondary actors	File System; Layout Engine; Style/Theme Manager; Library Manager (Custom Blocks); Versioning Service
Preconditions	1. Application launches; workspace is initialized. 2. User preferences (grid/snap, shortcuts) and libraries are available. 3. User has permission to read/write project files.
Main flow	1. User launches the tool; system loads workspace and preferences. 2. User chooses *New* or *Open* diagram. 3. **Advanced editing loop:** select block → edit properties / define custom block & save to library / connect with constraints & labels / align & auto-layout / apply style or theme. 4. System validates UML. 5. System saves a version snapshot. 6. (Optional) User compares with previous version and keeps/reverts. 7. User chooses *Export* → EPS (optionally JPG/PNG or batch). 8. System renders and writes file(s), then confirms location.
Postconditions	1. Diagram with advanced edits is saved. 2. Version history updated; any custom blocks/styles saved to library. 3. Exported file(s) exist at the chosen location.

Table 6.3: Use Case UC-03: Advanced editing and pro features

Use case: Advanced Create & Modify with Pro Features (UC-03)	
Alternative flows	**Missing assets** — When opening, referenced fonts/images not found; system prompts to locate/replace, then continue. **Validation violations** — Lint flags issues; system highlights and suggests fixes; user corrects and re-validates. **Export failure** — Render/write error; system shows message; user retries or cancels. **Revert changes** — User uses version compare to restore an earlier snapshot.
Related requirements	SR-01, SR-02, SR-03, SR-04, SR-07; UR-01, UR-02; DR-03, DR-04;NFR-01, NFR-03.

Table 6.4: Use Case UC-03: Advanced editing and pro features

6.8 Class Diagrams

xxxx

Chapter 7

Stakeholders

Table 7.1: Stakeholder Table

Stakeholder	Description	Relation
UML Creators	Those who need to create a uml diagram	They are going to be a large user base, and are the biggest scope of users for this project (essentially, a catch-all for everyone who might use the product)
Software Engineering students/Professors	Those who are learning how to make uml or teaching it	These people are mostly affected by the intuitive design and the functionality, with less emphasis on cost
Software Businesses	Those who are in charge of making large, complex programs that are used commercially and require specific and demanding code.	These people are mostly affected by cost and functionality, with less emphasis on having an intuitive design.
Original creators of Tikzit	The group who made the original Tikzit, the basis for this project	These people are affected by what we do with their code, and must be credited.
Original creators of Tiny Text	The group who made the original Tiny text, a useful tool for this project	These people are affected by what we do with their code, and must be credited.
Open source software people	Those who rely almost solely on open source code, and who might be relying on this product by itself.	These people care about intuitive design and functionality, similar to students and professors.
Competitor UML diagram makers	Those who work on or represent other UML making software (visual paradigm, draw.io, and others)	These will influence us, as well as potentially be influenced BY us, and affect how many users we will have directly.

Chapter 8

PreliminaryDesign

8.1 UI Design

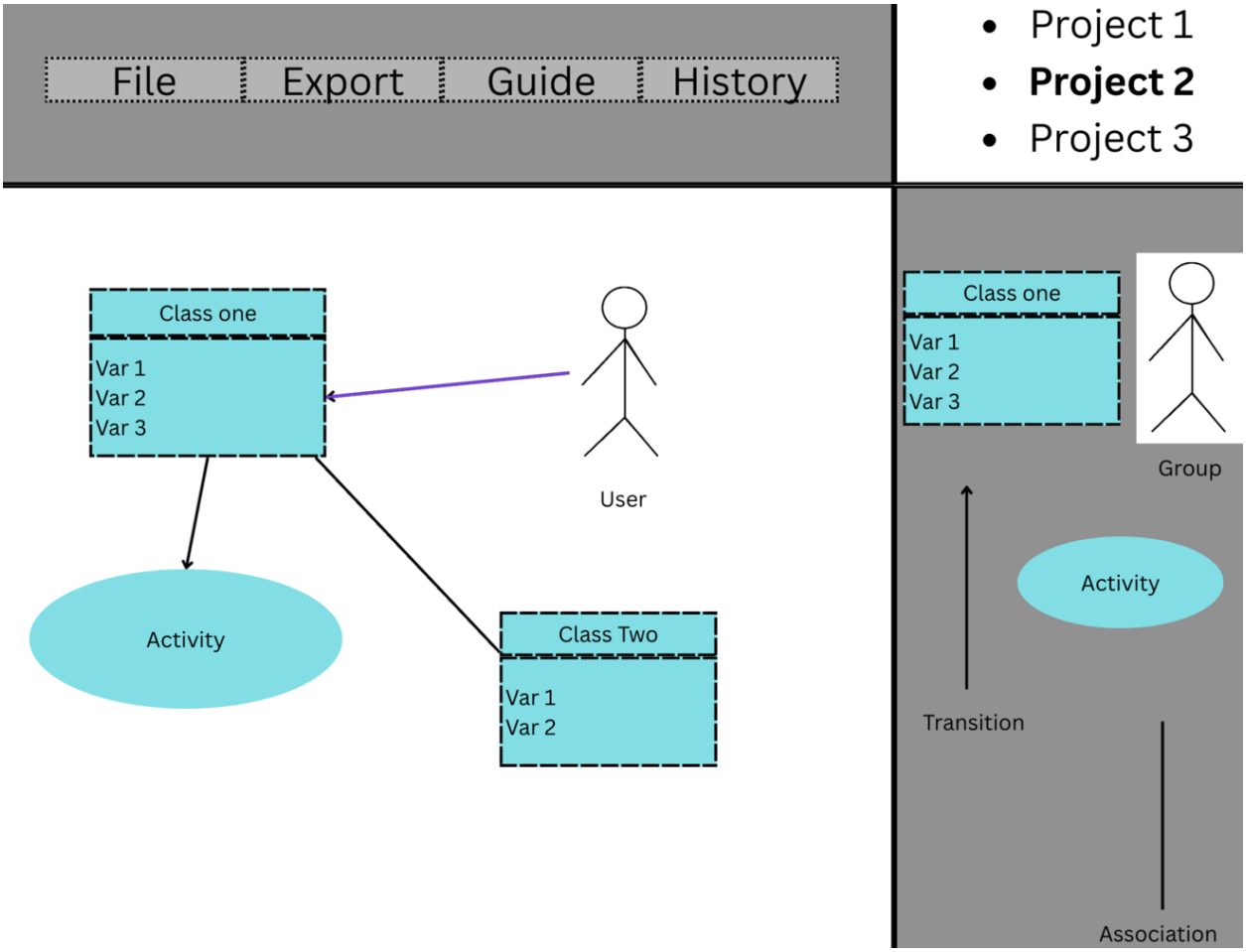

Figure 8.1: UI Design

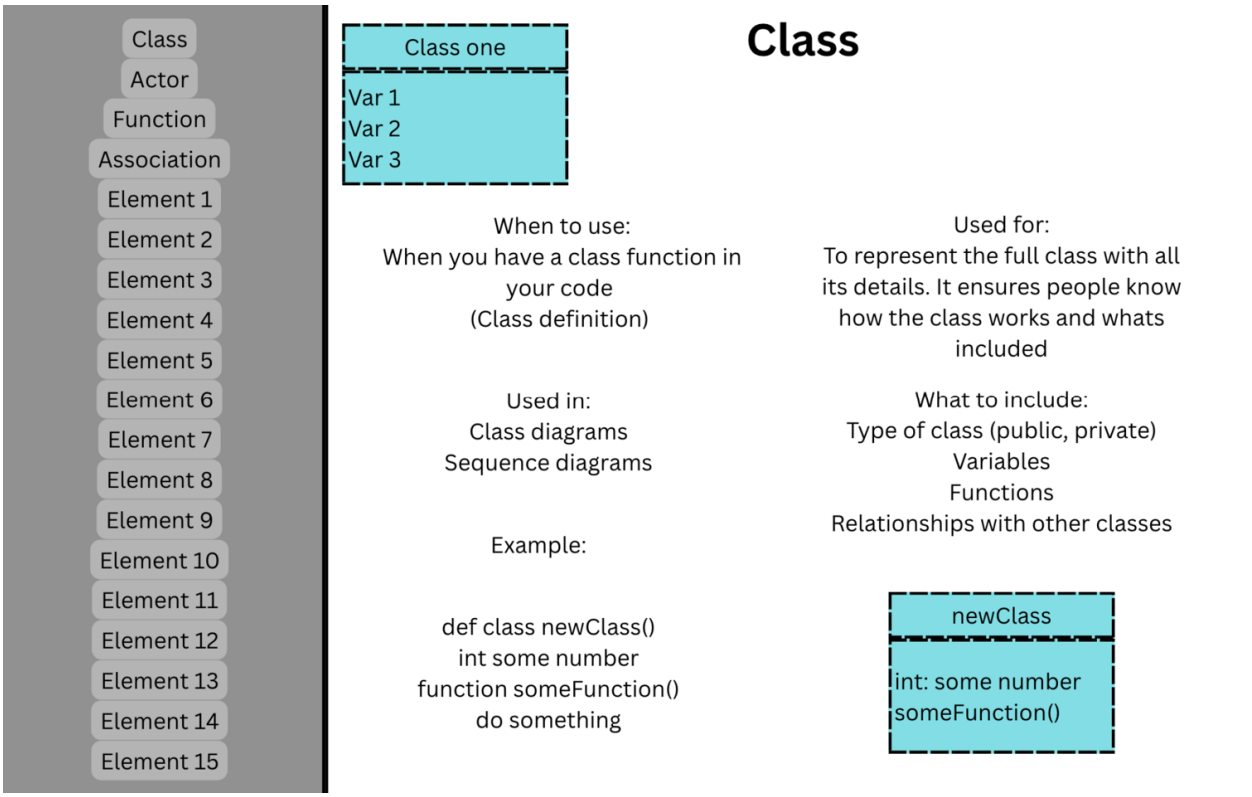

Figure 8.2: UI Design

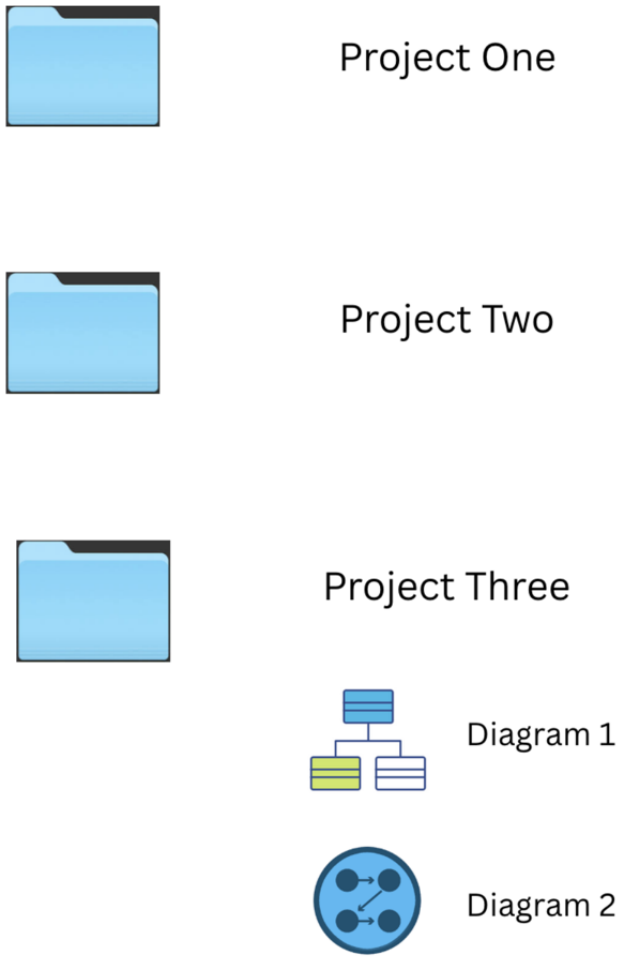

Figure 8.3: UI Design

8.2 User persona

Figure 8.4: User persona

8.3 Architecture Design

8.3.1 4+1 View

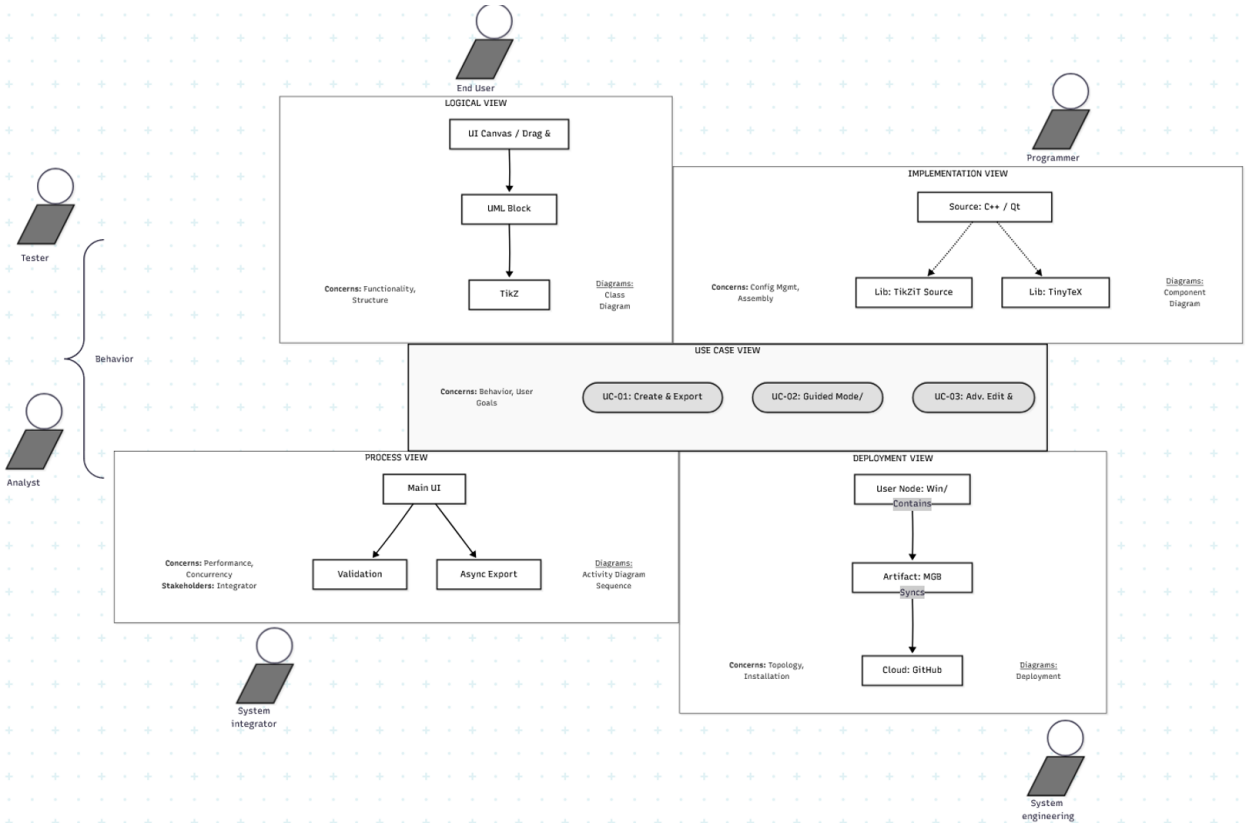

Figure 8.5: 4+1 Views

Use case and activity diagrams can see Chapter6

8.3.2 Logical View

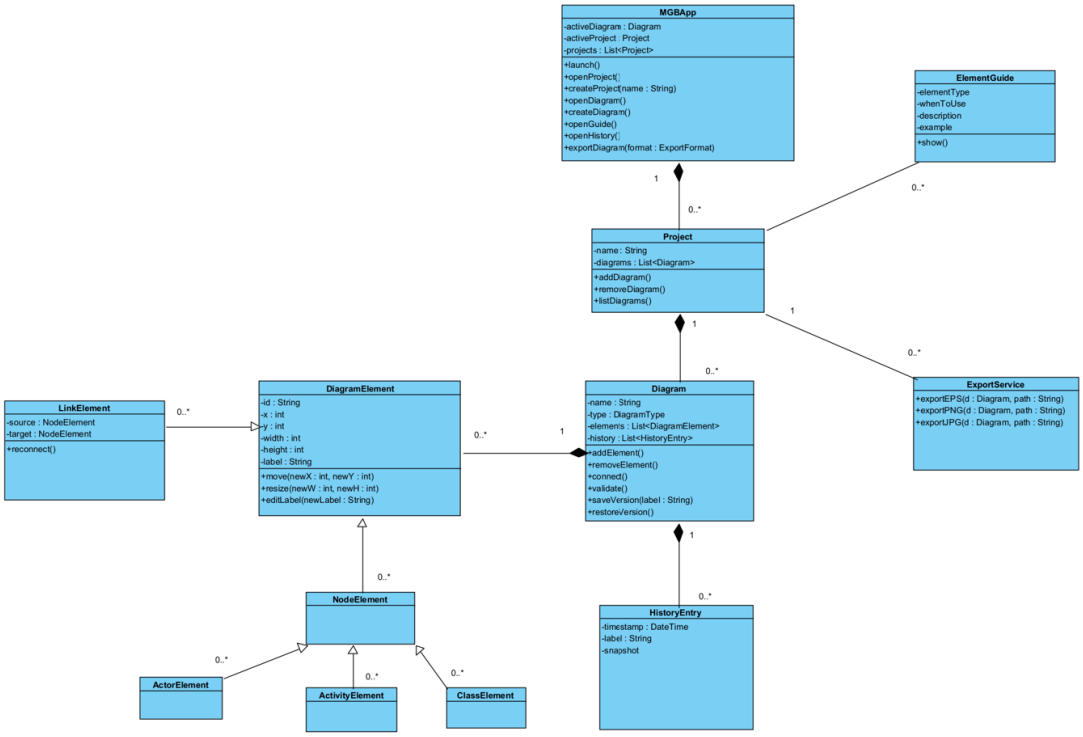

Figure 8.6: Logical View – Class Diagram

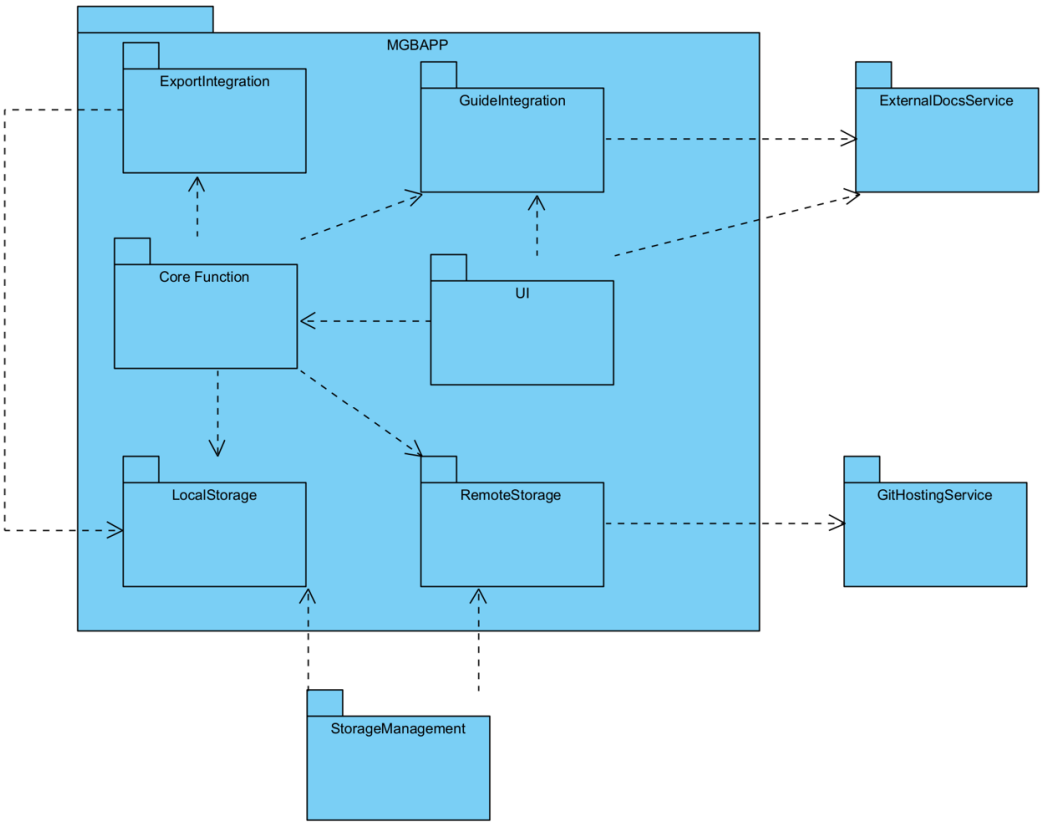

Figure 8.7: Logical View – Package Diagram

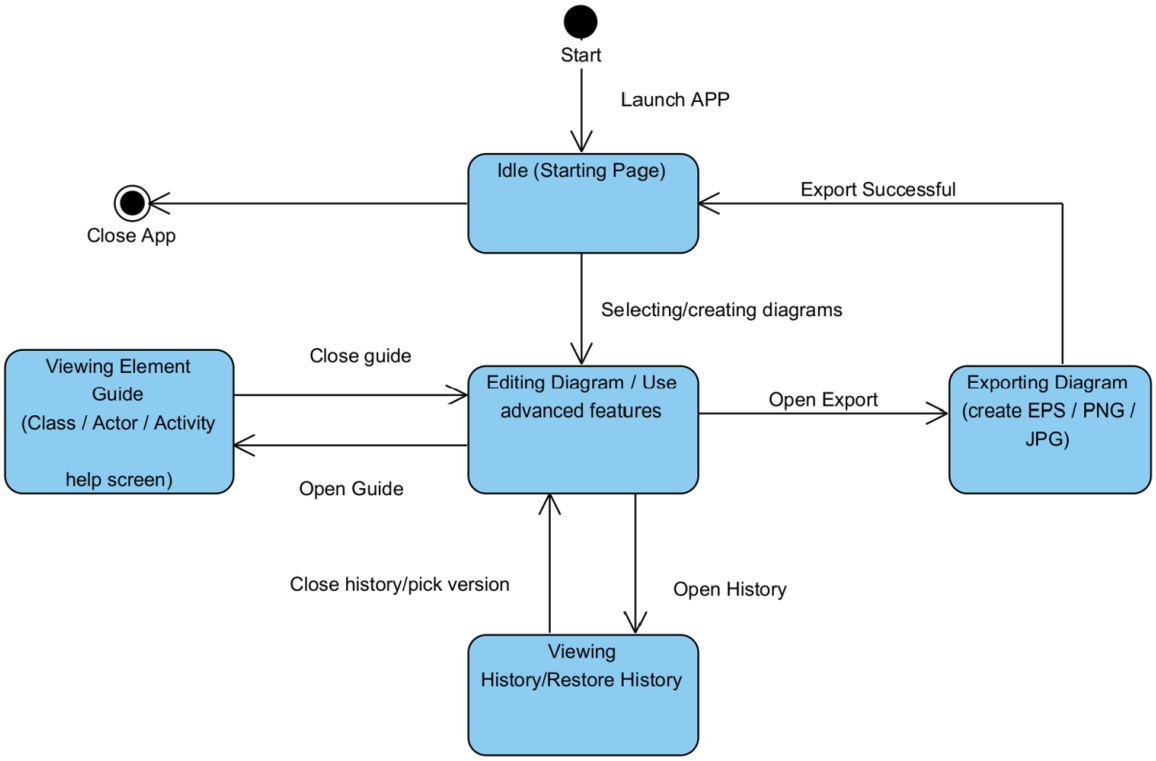

Figure 8.8: Logical View-State Machine Diagram

8.3.3 Process View

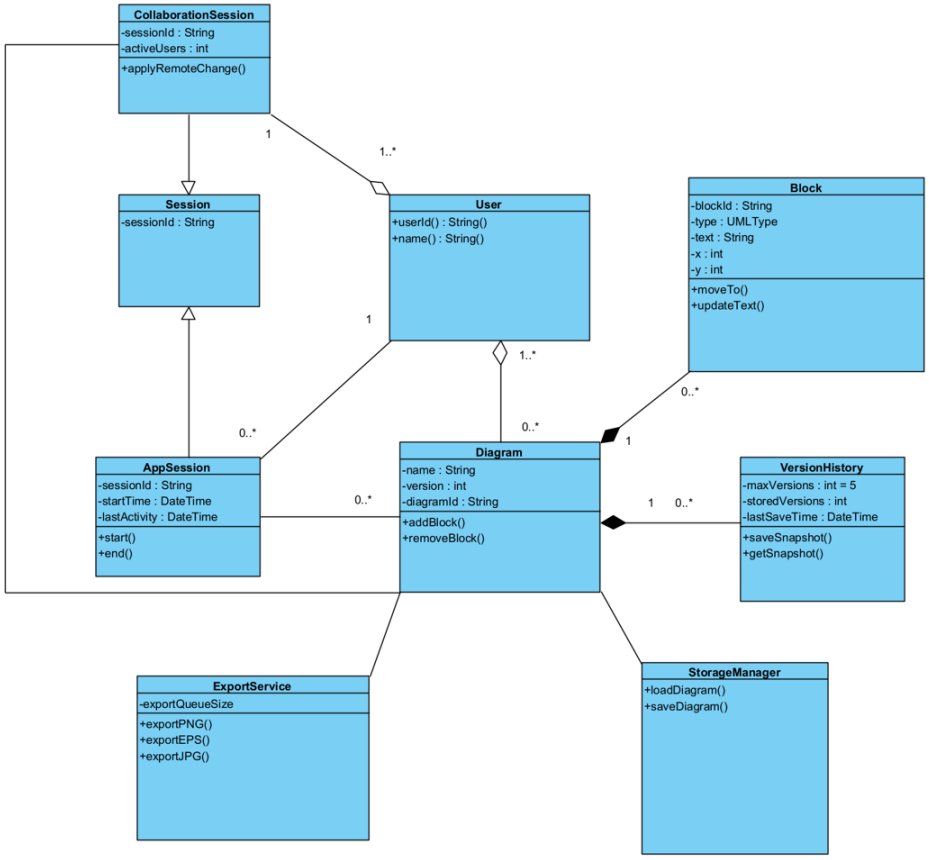

Figure 8.9: Process View-Class Diagram

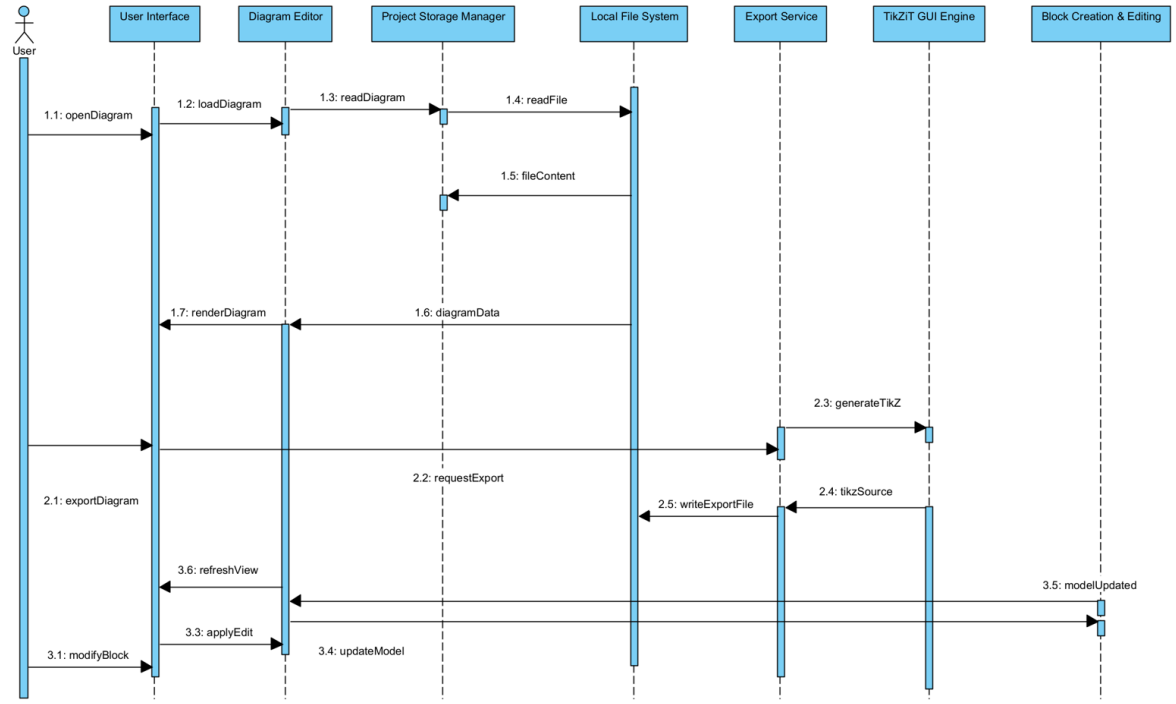

Figure 8.10: Process View-sequence Diagram

8.3.4 Implementation View

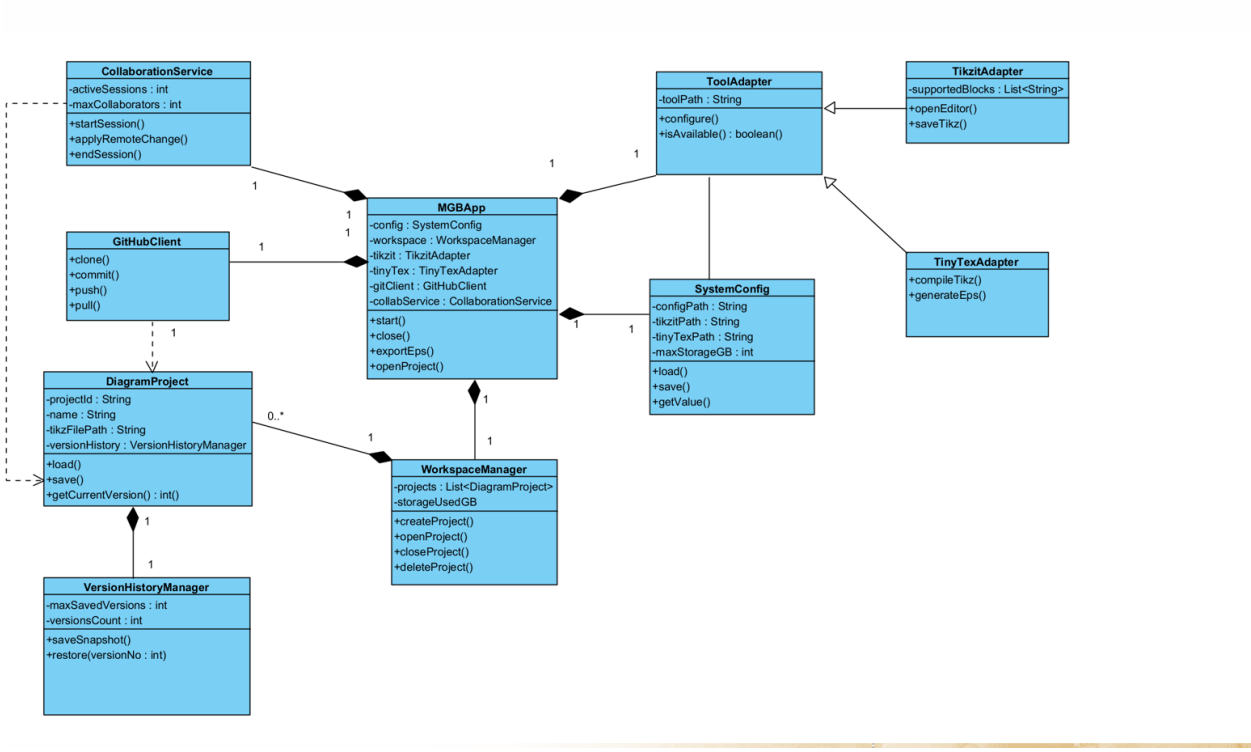

Figure 8.11: Implementation View-class Diagram

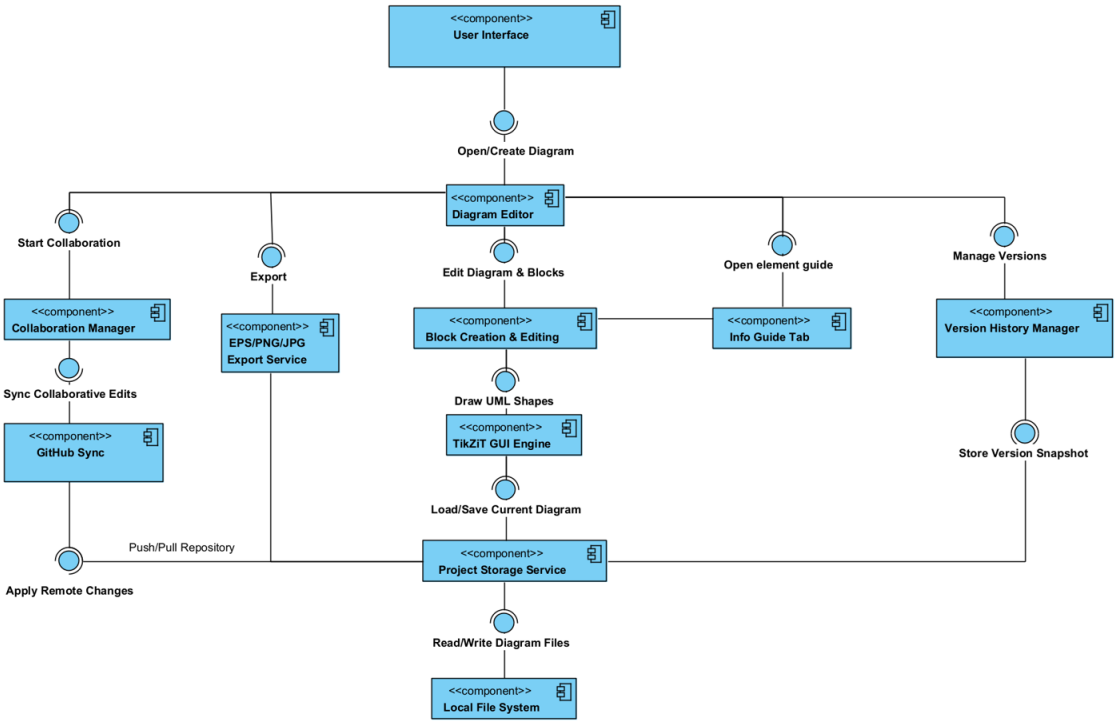

Figure 8.12: Implementation View-component Diagram

8.3.5 Deployment View

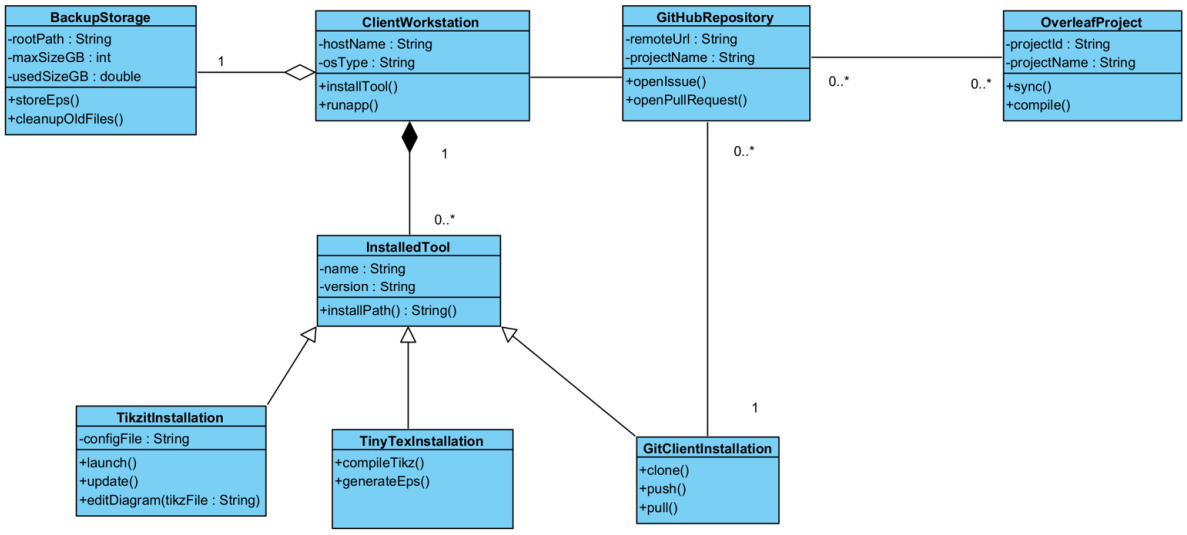

Figure 8.13: Deployment View-class Diagram

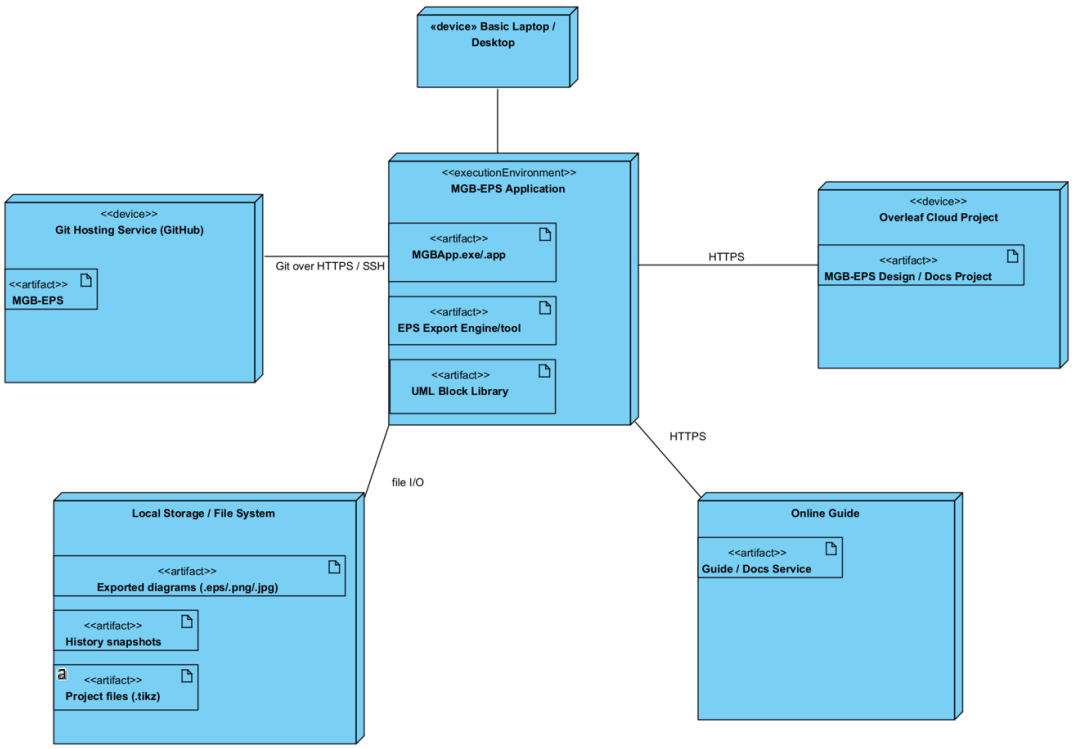

Figure 8.14: Deployment View-Deployment Diagram

Chapter 9

TikZiT Image

9.1 TikZiT on Linux

9.1.1 Working Environment

The first thing that was required was having access to Linux to actually get the app running on it. Since it was tested, and successfully built, on Windows, the instructions will be on a Windows 10 laptop. The version of windows is also very important as it adds additional steps to get the app to work.

The setup that was used is Ubuntu on Windows 10 using WSL 1 (in the beginning). This allowed us to get into a Linux space to start using linux based commands to get everything that was needed to get TikZiT up and running in desktop, NOT with the use of the web.

9.1.2 Requirements

There are quite a bit of libraries, packages, and build tools required to get TikZiT correctly built and running. This version also does not have a built in compiler such as TinyTex built into it as that would require some other methods to get it to work which have not been experimented with.

The code below will be the one that was used to install most of the required things to get the app working. Later in the section, it will talk about additional things that were needed after encountering errors in the build/run phase.

```
#Install required build tools
sudo apt-get update
sudo apt-get install -y \
    build-essential \
    git \
    cmake \
    pkg-config \
    flex \
    bison \
    wget \
    ca-certificates
```

```
12
13 #Install Qt6 packages
14 sudo apt-get install -y \
15     qmake6 \
16     qt6-base-dev \
17     qt6-tools-dev \
18     qt6-tools-dev-tools \
19     qt6-pdf-dev
20
21 #Install poppler libraries
22 sudo apt-get install -y libpoppler-dev
23
24 #Install X11 dependencies
25 sudo apt-get install -y \
26     libx11-xcb1 \
27     libxrender1 \
28     libxcb1 \
29     libxcb-xinerama0 \
30     libxkbcommon-x11-0
```

9.1.3 TikZiT Cloning

After installing the requirements, we need to clone the TikZiT app from its GitHub Repo and then cd into that new directory.

```
1 git clone https://github.com/tikzit/tikzit.git
2 cd tikzit
```

9.1.4 TikZiT Build Attempt and Errors

Now that we have the stuff required to attempt a build of the app, we do just that and then fix any errors that we encounter. The following lines of code need to be run to get the app built.

```
1 mkdir build
2 cd build
3 qmake ../tikzit.pro
4 make -j$(nproc)
```

After running this code, we started to run into errors with missing packages or libraries. The code from before has all of the required packages already built into it in Subsection 9.1.2. The errors in this subsection talk about which ones that needed to be added in order to get the code fixed.

The first thing that we to install were the QT6 dev packages since we got an error telling us that the qt headers were missing.

Next, we had to install libpoppler-dev because we got an error letting us know that it was missing.

One thing that kept on occurring is an error that told us we couldn't make the build directory since it already exists. This error occurred because we tried a build attempt previously and the folder stayed there during our next build attempts, so we had to run some codes to get rid of it and retry the build.

```
rm -rf build
mkdir build
cd build
qmake ../tikzit.pro
make -j$(nproc)
```

Another error that we ran into was the mix ups between the QT versions. TikZiT was built using QT 5 but the GitHub repo mentioned that it ran fine using QT 6 through Linux, thus we also did it that way.

```
#This was a quick and easy fix. Instead of using qmake in the code, we
   needed to use qmake6.
qmake6 ../tikzit.pro
make -j$(nproc)
```

After doing this, the app was successfully built through Linux. Now this is the part where the Windows version mattered and made it harder to get the app running. Windows 10 has no built in way of showing a Linux app through WSL, unlike Windows 11 which has it built in and requires no additional steps.

9.1.5 Running Linux app in Windows 10

To allow the WSL Linux machine on the Windows 10 laptop to run the app, we needed to run it through a separate app that allows it to go through a server and run on windows. The name of the app is VcXsrv. It was a quick and simple install, and after installing it, we needed to select the options to run our app correctly through our Linux WSL.

- Multiple Windows
- Disable access control
- Start no client
- Set Display Number to 0

Just in case, we also installed some additional xcb dependencies inside WSL so that we are sure that all dependencies are there to run a GUI app.

```
#We ran this code before but this one has additional dependencies
sudo apt update
sudo apt install -y \
    libx11-xcb1 \
    libxrender1 \
    libxcb1 \
    libxcb-icccm4 \
```

```
    libxcb-image0 \
    libxcb-keysyms1 \
    libxcb-randr0 \
    libxcb-render-util0 \
    libxcb-shape0 \
    libxcb-sync1 \
    libxcb-xfixes0 \
    libxcb-xinerama0 \
    libxkb1 \
    libxkbcommon-x11-0
```

After starting the server and downloading dependencies, we would need to run some additional code in the WSL Ubuntu to link up the WSL and the X server.

```
#Setup which display is used to connect to the server from WSL to the
   VcXsrv server.
export DISPLAY=$(ip route | awk '/^default/ {print $3}'):0

#To make the display permanent, we do the following command
echo "export DISPLAY=$(ip route | awk '/^default/ {print $3}'):0" >>
   ~/.bashrc
source ~/.bashrc
```

After running all of this, with the X server running in the background, we should now be able to run the app with the build that we created previously.

```
cd ~/tikzit/build
./tikzit
```

This final code allows us to run and open the app in Windows 10 through Linux.

9.2 Docker Image Setup

We now have a working Linux version of TikZiT. It can only work on Linux since it was built there, but having it working on Windows and Macintosh. We managed to circumvent this by taking the Linux version of the app that was installed through WSL, and create a Docker Image that will allow us to take it, upload it to Digital Ocean, and then create a separate web app with a domain that would allow anyone to view the website and use the app.

9.2.1 Required Files

To get the TikZiT app into a Docker format and allow it to run on a web browser without having to completely rewrite it into a web app, we need to add 4 important files into the root folder of the TikZiT app.

Dockerfile:

```
# ---------- Stage 1: Build TikZiT ----------
FROM ubuntu:22.04 AS builder
```

```
ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential git cmake pkg-config wget ca-certificates \
    qmake6 qt6-base-dev qt6-tools-dev qt6-tools-dev-tools qt6-pdf-dev \
    flex bison \
    libx11-xcb1 libxrender1 libxcb1 libxkbcommon-x11-0 \
    libpoppler-dev \
    && apt-get clean && rm -rf /var/lib/apt/lists/*

WORKDIR /src
# Copy source into container
COPY . /src

# --- CHANGE 1: Fix compilation error (Qt API mismatch) ---
# Replaces 'pagePointSize' with 'pageSize' in the source code before building
RUN sed -i 's/pagePointSize/pageSize/g' src/data/pdfdocument.cpp

RUN mkdir -p /src/build && cd /src/build && qmake6 ../tikzit.pro && make -j$(nproc)

# Collect built binary and required shared libs into /out
RUN mkdir -p /out/bin /out/lib && \
    cp /src/build/tikzit /out/bin/ || cp /src/build/release/tikzit /out/bin/ || true && \
    ldd /out/bin/tikzit | awk '/=>/ {print $(NF-1)}' | grep '^/' | xargs -I '{}' cp -v '{}' /out/lib || true

# ---------- Stage 2: Runtime ----------
FROM ubuntu:22.04
ENV DEBIAN_FRONTEND=noninteractive
ENV DISPLAY=:1
EXPOSE 8080 5901

# minimal runtime deps + VNC/noVNC environment
# minimal runtime deps + VNC/noVNC environment
RUN apt-get update && apt-get install -y --no-install-recommends \
    xfce4 xfce4-goodies \
    tigervnc-standalone-server tigervnc-common tigervnc-tools \
    websockify novnc supervisor python3 git wget ca-certificates \
    libxcb-xinerama0 libxkbcommon-x11-0 \
    # The "Nuclear" Base
    qt6-base-dev \
    # PDF Support
    libqt6pdf6 libpoppler-dev \
    # --- NEW: VISUAL FIXES (Icons) ---
```

```
    libqt6svg6 \
    adwaita-icon-theme \
    # --- NEW: DOXYGEN SUPPORT ---
    doxygen \
    graphviz \
    # --------------------------------
    # REQUIRED FONTS & UTILS
    dbus-x11 x11-xserver-utils xfonts-base xfonts-100dpi xfonts-75dpi xfonts-scalable \
    # XCB LIBS
    libxcb-cursor0 libxcb-icccm4 libxcb-image0 libxcb-keysyms1 libxcb-randr0 \
    libxcb-render-util0 libxcb-shape0 libxcb-shm0 libxcb-sync1 libxcb-xfixes0 libxcb-xkb1 \
    && apt-get clean && rm -rf /var/lib/apt/lists/*

# create a non-root user for the session
RUN useradd -m -s /bin/bash tikzituser

# copy built app (We only copy the binary, relying on installed libs for stability)
COPY --from=builder /out/bin/tikzit /usr/local/bin/tikzit

# --- CHANGE 3: Removed the manual library copy and LD_LIBRARY_PATH ---
# We installed the libraries via apt in Change 2, which solves the "UndefinedVar" warning
# and prevents "Qt platform plugin xcb not found" errors.

# install noVNC (Pinned to v1.4.0 to fix JS errors)
RUN git clone --depth 1 --branch v1.4.0 https://github.com/novnc/noVNC.git /opt/noVNC || true
RUN git clone --depth 1 https://github.com/novnc/websockify.git /opt/noVNC/utils/websockify || true

# copy supervisor config and entrypoint
COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf
COPY entrypoint.sh /usr/local/bin/entrypoint.sh
RUN chmod +x /usr/local/bin/entrypoint.sh

VOLUME ["/home/tikzituser/.vnc", "/home/tikzituser/.local/share/tikzit"]

ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]
CMD ["supervisord", "-n", "-c", "/etc/supervisor/conf.d/supervisord.conf"]
```

entrypoint.sh:

```
#!/usr/bin/env bash
set -euo pipefail

# default env vars
VNC_PASSWORD="${VNC_PASSWORD:-tikzitMGBadmin}"
VNC_GEOMETRY="${VNC_GEOMETRY:-1280x800}"
VNC_DISPLAY="${VNC_DISPLAY:-:1}"
VNC_USER="${VNC_USER:-tikzituser}"
USER_HOME="/home/$VNC_USER"

# ensure user exists (should already in Dockerfile)
if ! id "$VNC_USER" >/dev/null 2>&1; then
  useradd -m -s /bin/bash "$VNC_USER"
fi

# create vnc password
mkdir -p "$USER_HOME/.vnc"
echo "$VNC_PASSWORD" | vncpasswd -f > "$USER_HOME/.vnc/passwd"
chown -R "$VNC_USER:$VNC_USER" "$USER_HOME/.vnc"
chmod 600 "$USER_HOME/.vnc/passwd"

# create xstartup to launch XFCE and tikzit in the X session
# create xstartup to launch XFCE and tikzit in the X session
cat > "$USER_HOME/.vnc/xstartup" <<'EOF'
#!/bin/sh
xrdb $HOME/.Xresources

# 1. Start TikZiT in the BACKGROUND with a delay so XFCE loads first
(sleep 3 && /usr/local/bin/tikzit &>/tmp/tikzit.log) &

# 2. Start XFCE in the FOREGROUND (Do NOT use '&')
# This keeps the container running.
exec startxfce4
EOF

chown "$VNC_USER:$VNC_USER" "$USER_HOME/.vnc/xstartup"
chmod +x "$USER_HOME/.vnc/xstartup"

rm -rf /tmp/.X11-unix /tmp/.X*-lock
mkdir -p /tmp/.X11-unix
chmod 1777 /tmp/.X11-unix

# ensure noVNC exists in /opt/noVNC
if [ ! -d /opt/noVNC ]; then
  # Use the pinned version to avoid the "init_logging" error
  git clone --depth 1 --branch v1.4.0 https://github.com/novnc/noVNC.
   git /opt/noVNC
```

```
  git clone --depth 1 https://github.com/novnc/websockify.git /opt/noVNC/utils/websockify
fi

# start supervisord (will start vncserver and novnc)
exec "$@"
```

supervisord.conf:

```
[supervisord]
nodaemon=true
logfile=/var/log/supervisord.log
logfile_maxbytes=0

[program:vncserver]
# We use -SecurityTypes VncAuth to force the server to accept the password file
command=/usr/bin/vncserver :1 -fg -geometry 1280x800 -depth 24 -localhost no -SecurityTypes VncAuth
user=tikzituser
autorestart=true
environment=USER="tikzituser",HOME="/home/tikzituser",DISPLAY=":1"
stdout_logfile=/var/log/vncserver.stdout.log
stderr_logfile=/var/log/vncserver.stderr.log

[program:novnc]
command=/usr/bin/websockify --web=/opt/noVNC --wrap-mode=ignore 8080 localhost:5901
user=tikzituser
autorestart=true
stdout_logfile=/var/log/novnc.stdout.log
stderr_logfile=/var/log/novnc.stderr.log

; optional: start tikzit outside of the vnc session (not usually recommended)
;[program:tikzit]
;command=/usr/local/bin/tikzit
;user=tikzituser
;autorestart=true
;stdout_logfile=/var/log/tikzit.stdout.log
;stderr_logfile=/var/log/tikzit.stderr.log
```

docker-compose.yml

```
version: "3.8"
services:
  tikzit:
    build: .
    image: mylocal/tikzit:latest
```

```
    ports:
      - "8080:8080"   # noVNC web UI
      # - "5901:5901" # optional direct VNC
    environment:
      - VNC_PASSWORD=tikzitMGBadmin
    volumes:
      - ./data:/home/tikzituser/.local/share/tikzit
      - .:/src
    restart: unless-stopped
```

9.2.2 Error Fixes

The files in subsection 9.2.1 are the final versions that are used to build the TikZiT app in Docker and have it up and running in the web, but in order to get these files in a working state, a lot of errors needed to be fixed before a full working application. The following will show the errors that occurred and what done in order to fix them.

The very first thing that is required is having the Docker Desktop App. This allows the computer to connect to Docker and then connect to the web to show the application. The first major error that occurred when trying to build the first iteration of the TikZiT app was the discrepancy between Ubuntu WSL being in WSL 1 and the Docker Desktop App being in WSL 2. To fix this, we needed to update Ubuntu to WSL 2 so that both apps are in the same WSL version.

```
wsl --set-version Ubuntu 2
```

This code will update Ubuntu to WSL 2, and will take some time to update everything. After the update, we need to look into and change some settings in the Docker Desktop App.

1. In Docker Desktop on Windows
2. Go to: Settings → General → Use the WSL 2 based engine Make sure the box is checked
3. Then go to: Settings → Resources → WSL Integration Enable Docker integration for Ubuntu distribution
4. Click Apply and Restart

After this, we get the correct WSL version for both the Docker Desktop App and Ubuntu. Next, we have the errors that we got with the files that are required to get TikZiT up and running with Docker.

The first error that we get is the discrepancies between the versions of the QT library. TikZiT was originally built using QT 5 libraries, but we are trying to install QT 6 libraries into our Docker project because the GitHub page of TikZiT mentions that they got it to work on Linux using QT 6.2 and it is also more up to date. The TikZiT code is calling pagePointSize(), but in the version of Qt installed in the Docker container, that function has been renamed or deprecated in favor of pageSize(). In order to fix this, we added a sed command to automatically replace the old problematic function with the new one. There were also some minor warnings that we fixed so we stopped getting that warning when running the code.

```
# --- CHANGE 1: Fix compilation error (Qt API mismatch) ---
# Replaces 'pagePointSize' with 'pageSize' in the source code before building
RUN sed -i 's/pagePointSize/pageSize/g' src/data/pdfdocument.cpp

# --- CHANGE 2: Add Qt runtime libraries explicitly ---
    # Copying libs manually (Change 3) is fragile. Installing basic Qt6 libs ensures plugins work.
    libqt6widgets6 libqt6gui6 libqt6core6 libqt6pdf6 \

# --- CHANGE 3: Removed the manual library copy and LD_LIBRARY_PATH ---
# We installed the libraries via apt in Change 2, which solves the "UndefinedVar" warning
# and prevents "Qt platform plugin xcb not found" errors.
```

After this, we finally get somewhere in the web application. We finally managed to get the app built and run using the following commands

```
#We have a docker-compose.yml file for a reason, so we use it to build and run the app
docker-compose build

#The following command is used to run the app so that we can view it in the web

docker-compose up

#The following one is to take down the localhost so that it stops running in the web

docker-compose down
```

Now, since we get some visuals in the web, we can use them to get help and fix their issues. The following is a directory that we see when going into the web app:

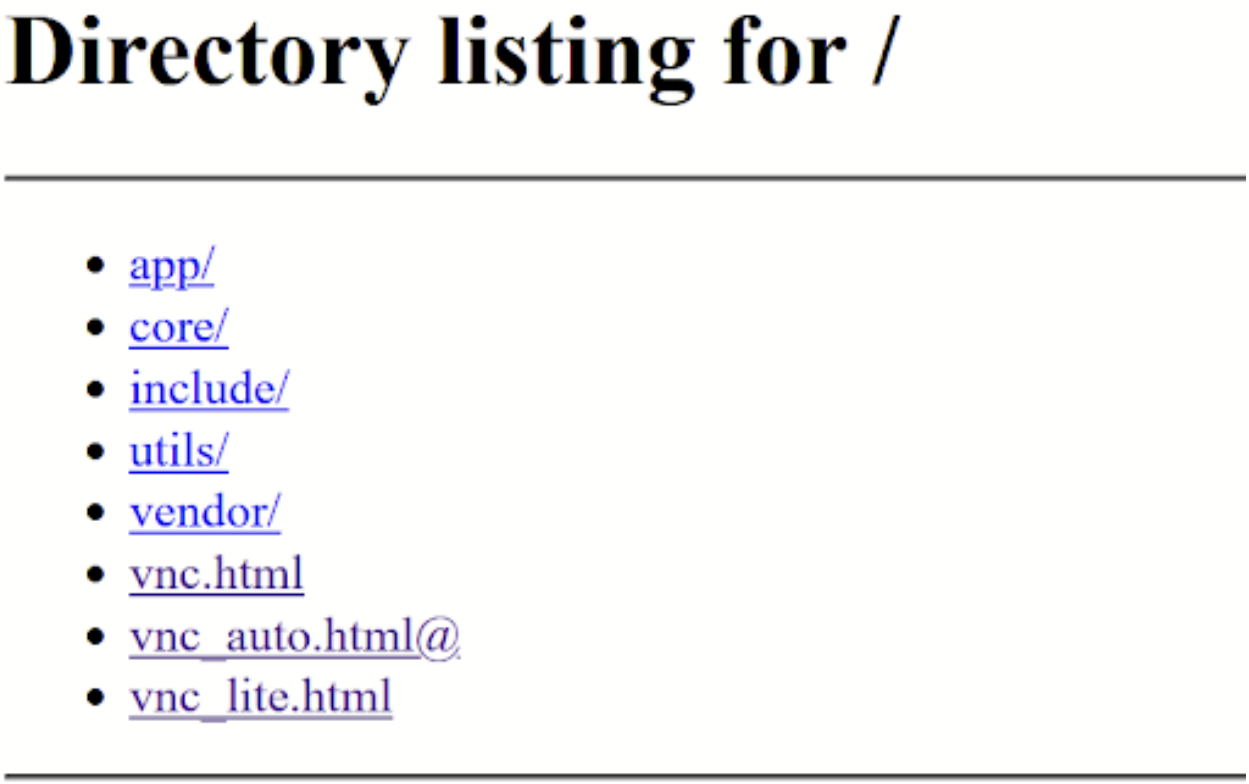

Figure 9.1: Simple Directory for TikZiT app

Chapter 10

Prototype

10.1 Issue Tracking

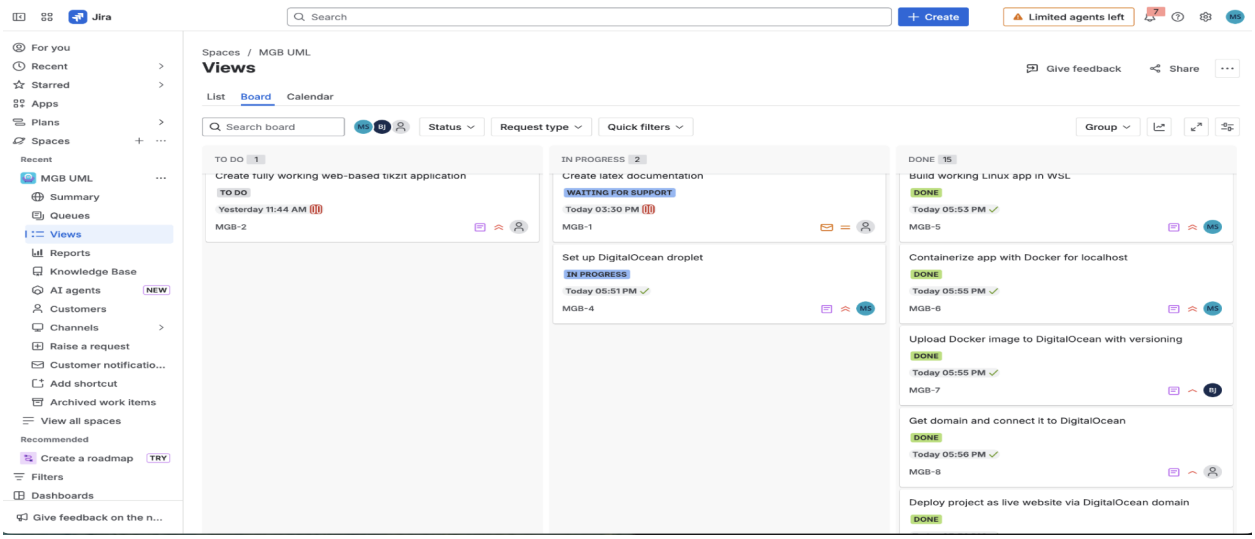

Figure 10.1: Issue Tracking Jira

We used the Jira board as the primary tool for tracking issues and planning the MGB UML project. Compared to GitHub issues, the Jira board is more professional and fits our agile process better. The work items follow a column-based structure, which includes To Do, In Progress, and Done sections. The team can monitor task statuses through this visual workflow, which shows their current status and shows their progress through DevOps stages (build, containerize, deploy, document) and shows how requirements link to finished work items for issue-tracking integration as needed.

10.2 CI/CD Pipeline

10.2.1 Triggering the Pipeline

Figure 10.2: Pipeline Triggering Commands

The pipeline triggering starts locally on the developer's computer. After a developer makes a change in either the Latex document or the source code, they run the script `publish_all.sh` to start the pipeline.

10.2.2 Version Control

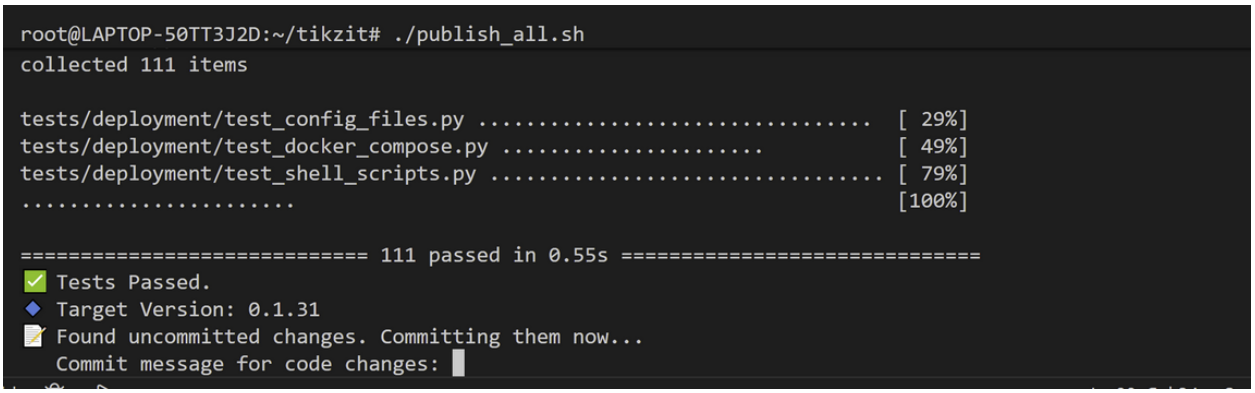

Figure 10.3: Pipeline Commit Message

The `publish_all.sh` script is also where it asks the user to add a commit message and then the script automatically changes the version of the app and pushes it to GitHub and DigitalOcean.

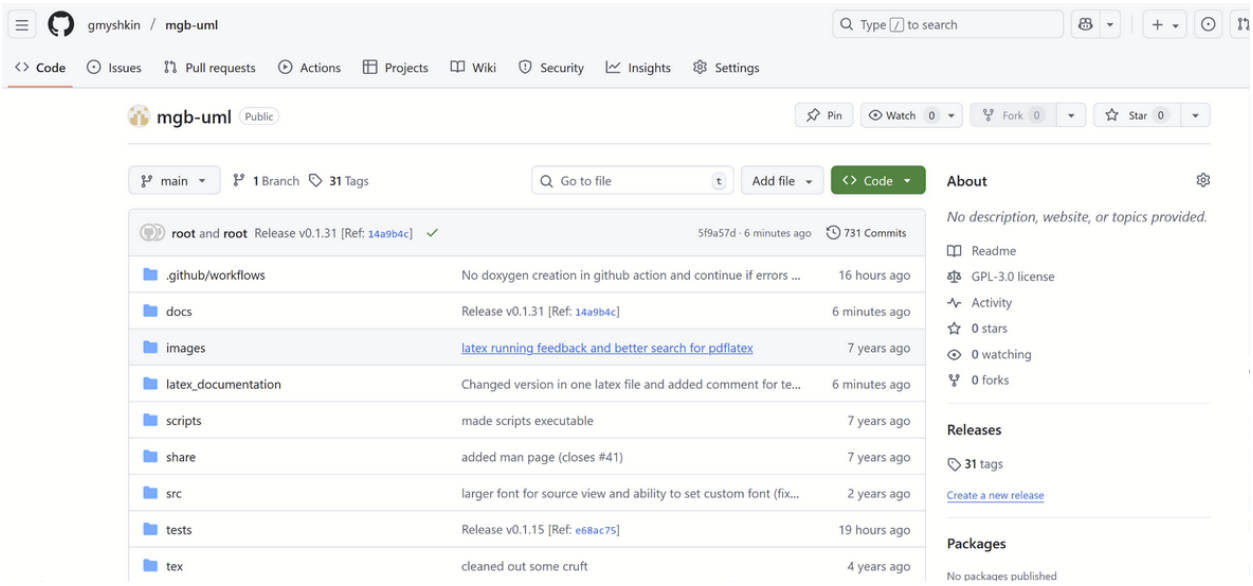

Figure 10.4: GitHub Commit Hash on Main Page

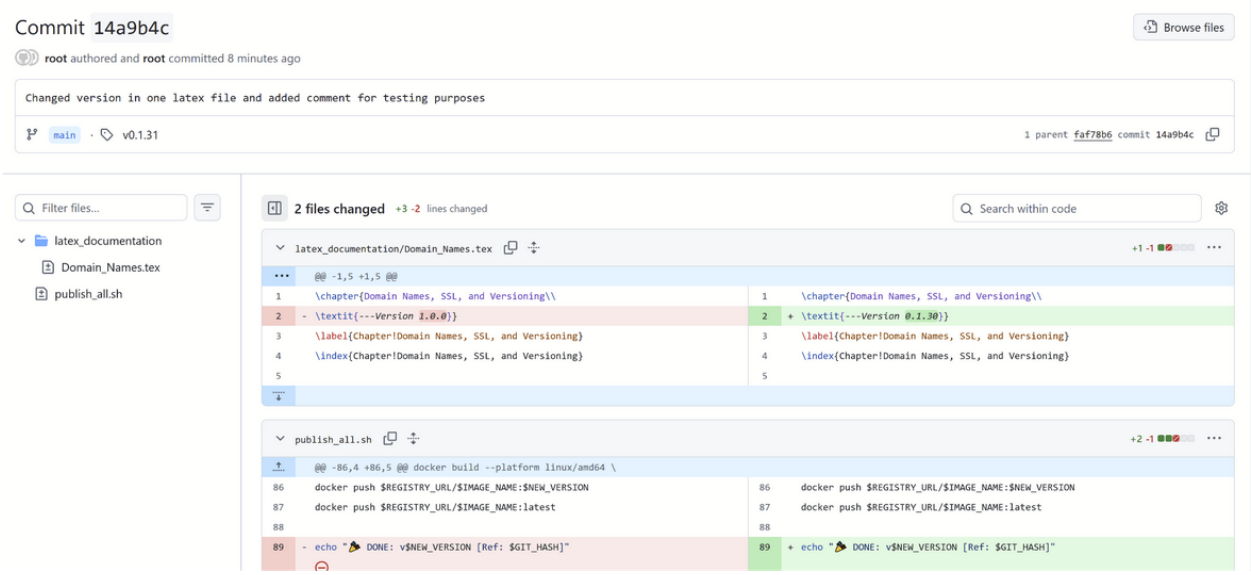

Figure 10.5: GitHub Commit Hash

Figure 10.4 shows that when a commit is created and pushed to GitHub, it will show the hash number of the commit so that a developer can reference it easily. Figure 10.5 shows the changes created in the commit when pressing the hash number.

10.2.3 Build Process

```
[+] Building 8.0s (9/26)                                                                                    docker:default
 => CACHED [builder 2/7] RUN apt-get update && apt-get install -y --no-install-recommends     build-essential git cmake pkg-confi  0.0s
 => CACHED [builder 3/7] WORKDIR /src                                                                                        0.0s
 => [builder 4/7] COPY . /src                                                                                                3.8s
 => [builder 5/7] RUN sed -i 's/pagePointSize/pageSize/g' src/data/pdfdocument.cpp                                           0.7s
 => [builder 6/7] RUN mkdir -p /src/build && cd /src/build && qmake6 ../tikzit.pro && make -j$(nproc)                       2.6s
 => => # /mkspecs/linux-g++ -o tikzdocument.o ../src/data/tikzdocument.cpp
 => => # g++ -c -pipe -O2 -Wall -Wextra -D_REENTRANT -fPIC -DQT_DEPRECATED_WARNINGS -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_PDF_LIB -DQT_GU
 => => # I_LIB -DQT_NETWORK_LIB -DQT_CORE_LIB -I/src -I. -I../src -I../src/gui -I../src/data -I/usr/include/x86_64-linux-gnu/qt6 -I/usr
 => => # /include/x86_64-linux-gnu/qt6/QtWidgets -I/usr/include/x86_64-linux-gnu/qt6/QtPdf -I/usr/include/x86_64-linux-gnu/qt6/QtGui -I
 => => # /usr/include/x86_64-linux-gnu/qt6/QtNetwork -I/usr/include/x86_64-linux-gnu/qt6/QtCore -I. -I. -I/usr/lib/x86_64-linux-gnu/qt6
 => => # /mkspecs/linux-g++ -o undocommands.o ../src/gui/undocommands.cpp
```

Figure 10.6: Pipeline Commit Message

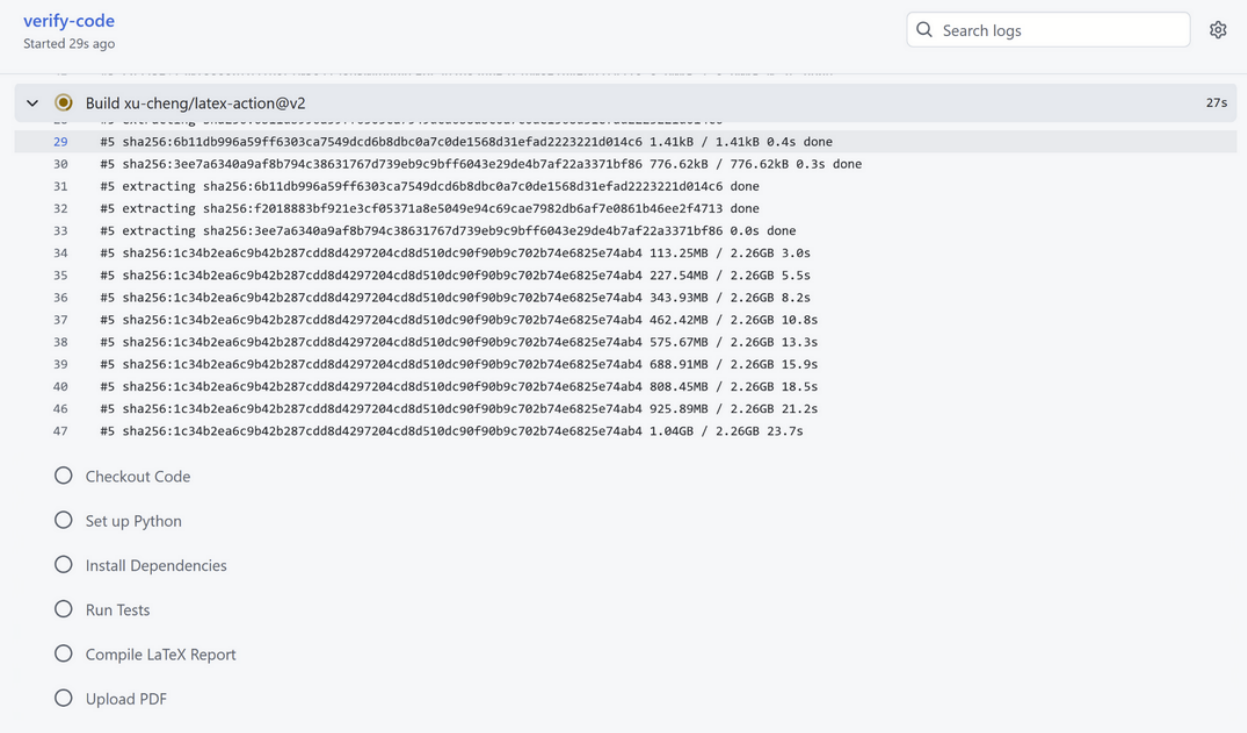

Figure 10.7: Pipeline Commit Message

The above two images show the build process being done in both the terminal and GitHub actions.

10.3 Automated Tests

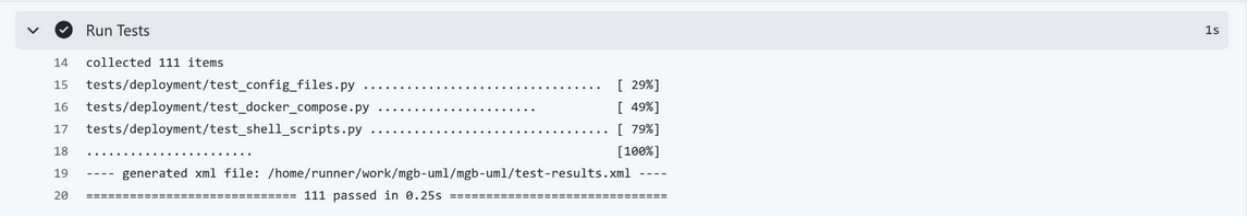

Figure 10.8: Pipeline Commit Message

The above image shows how the tests are done. They are python tests created and automatically tested in GitHub actions.

10.4 Versioning

We have two different methods of versioning since the application is pushed to both GitHub as its own version and to DigitalOcean as its own image (or as tags with numbers)

10.4.1 GitHub Versioning

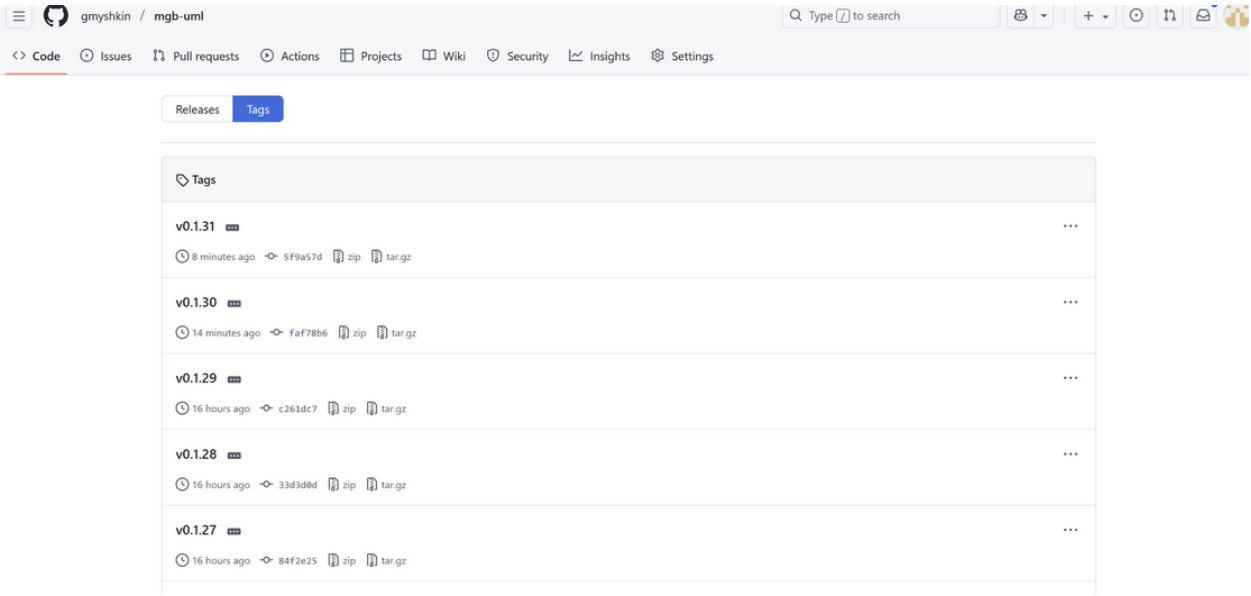

Figure 10.9: Pipeline Commit Message

This image shows all of the versions that appear in GitHub when they are pushed there. This allows a developer to take a look at one of them and download their files.

10.4.2 DigitalOcean Versioning

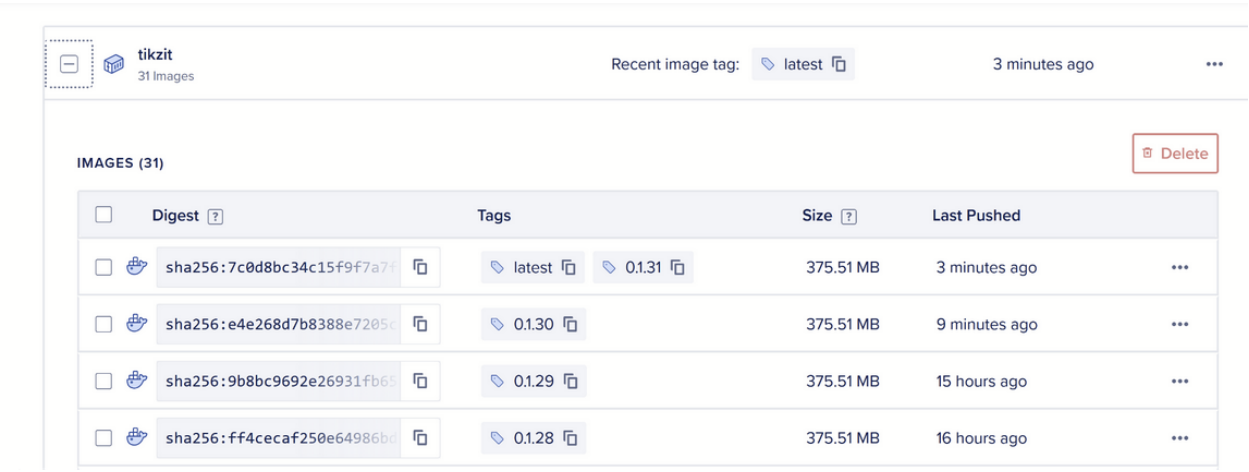

Figure 10.10: Pipeline Commit Message

This image shows how versioning works in DigitalOcean. They are pushed there as different image tags and allows a person to download them depending on which version they want to use. This also allows the web version of the app to change depending on which tag is chosen when redeploying the app on the DigitalOcean droplet.

10.5 Configuration Files

The following are the files we used and their explanations:

- **`Docker-compose.prod.yml:`** Sets up production environment with Tikzit, Nginx, and Certbot and Powers the live website.
- **`Multi-user-manager:`** Created dedicated TikZiT container and nginx route for new users.
- **`publish_all.sh:`** Automates workflow, updating version numbers, tagging git releases, etc and Pushes version and images to DigitalOcean registry.
- **Dockerfile** Defines how Tikzit runs in the linux desktop and Includes Qt libraries.
- **`supervisord.conf:`** Runs VNC server and noVNC gateway and, ensures its running and restarts on crash.
- **`nginx.conf:`** Handles HTTP and HTTPS traffic and Includes per user routing configs.
- **`main.yml:`** Automatically does testing and generates a pdf of the results in GitHub Actions.

10.6 Cloud Execution

The app was running on a DigitalOcean droplet with a domain of mgb-uml.me.

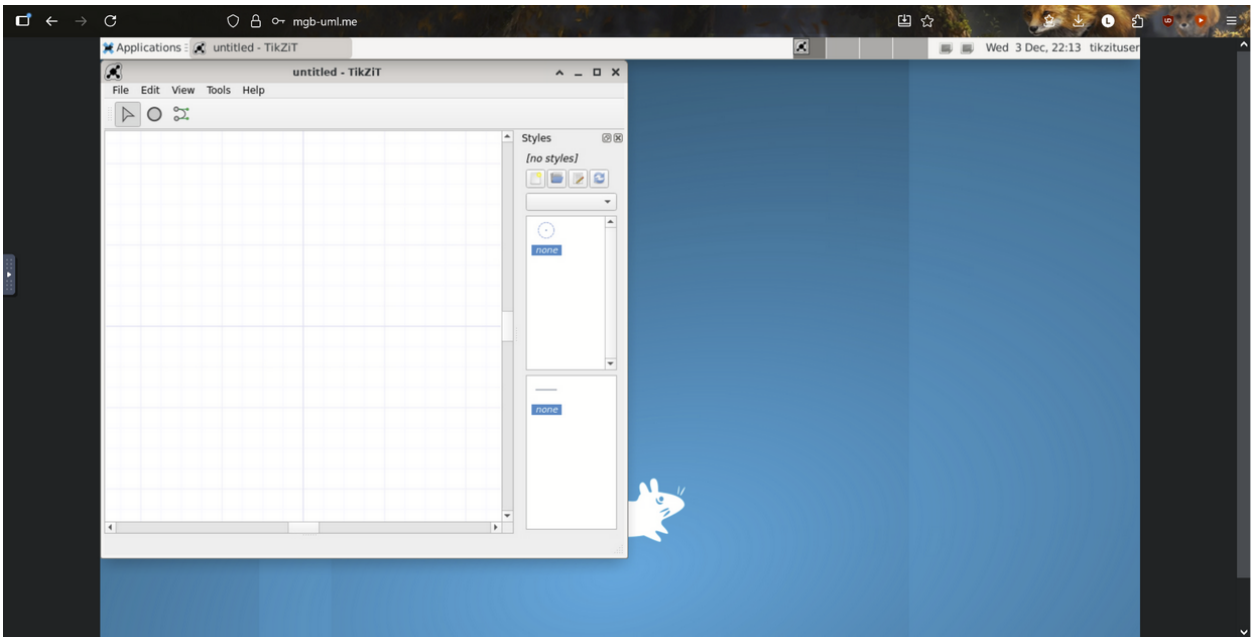

Figure 10.11: How the Web App looks like

```
To delete this message of the day: rm -rf /etc/update-motd.d/99-one-click
Last login: Wed Dec  3 06:03:18 2025 from 162.243.188.66
root@mgb-uml:~# cd tikzit_deployment/
root@mgb-uml:~/tikzit_deployment# ./update_and_redeploy.sh
⬇Force-Pulling latest image from Registry...
latest: Pulling from mgb-uml/tikzit
63e5bc7682b8: Already exists
a021f968773d: Already exists
5e23fd4d1bfd: Already exists
d0439694476e: Already exists
698311175a39: Already exists
ea6cc5936e6d: Already exists
8b3202480abe: Already exists
21a38005a625: Already exists
c0860c6151e6: Already exists
4521cf709b37: Already exists
5876b8dca965: Already exists
8f32a62de2b2: Extracting [================>                                  ]  15.24MB/47.36MB
4f4fb700ef54: Download complete
bd9466c4a0c6: Download complete
855924b31910: Download complete
```

Figure 10.12: DigitalOcean image being downloaded

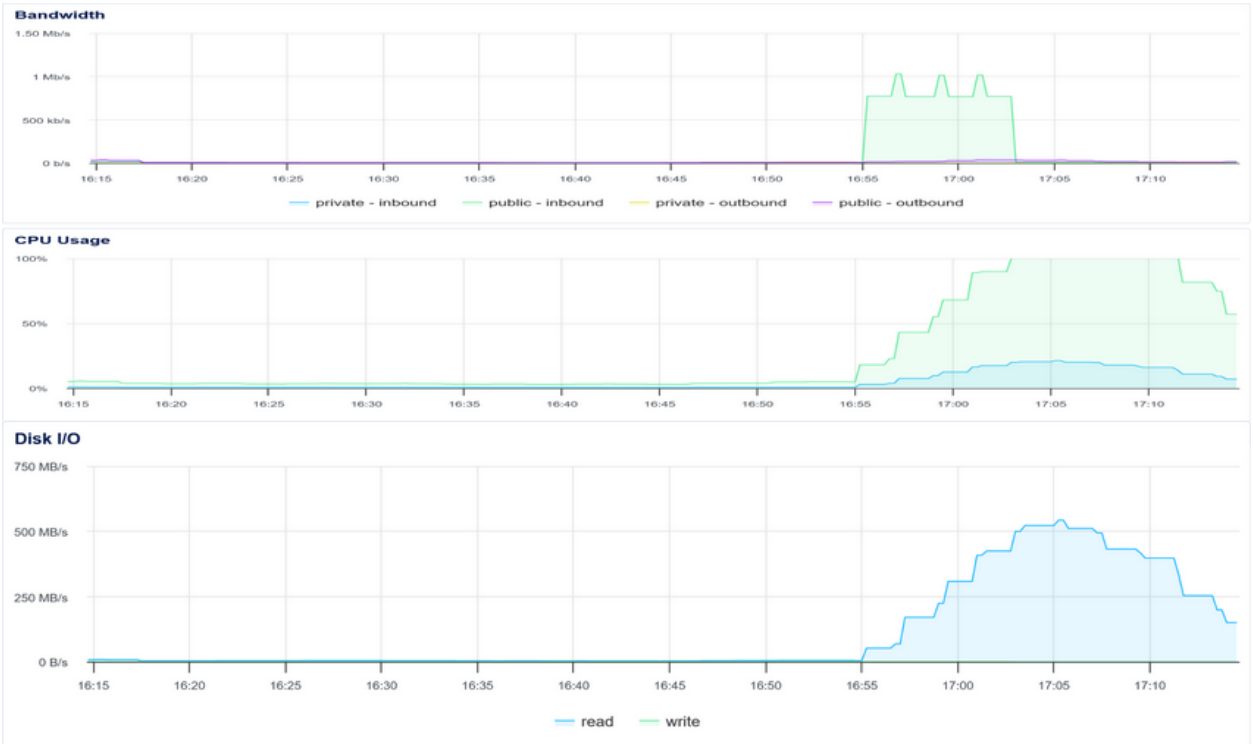

Figure 10.13: DigitalOcean Graphs that monitor state of droplet

Figure 10.11 shows us how the web app looks like on the domain when entering it through noVNC Figure 10.12 shows us the process of the DigitalOcean droplet downloading the new changes that were pushed to the DigitalOcean registry of the app's immage. Figure 10.13 shows us the graphs of the DigitalOceans droplet when it has changes being done to it such as uploading or downloading new image tags. It shows us the bandwidth, CPU Usage, and Disk IO.

10.7 Reverse Documentation

Figure 10.14: Class List of Doxygen's Reverse Documentation

This figure above shows us that Doxygen was used to create Reverse Architecture of the app that was uploaded to DigitalOcean and GitHub. This reverse documentation also shows the version of the app that the document used.

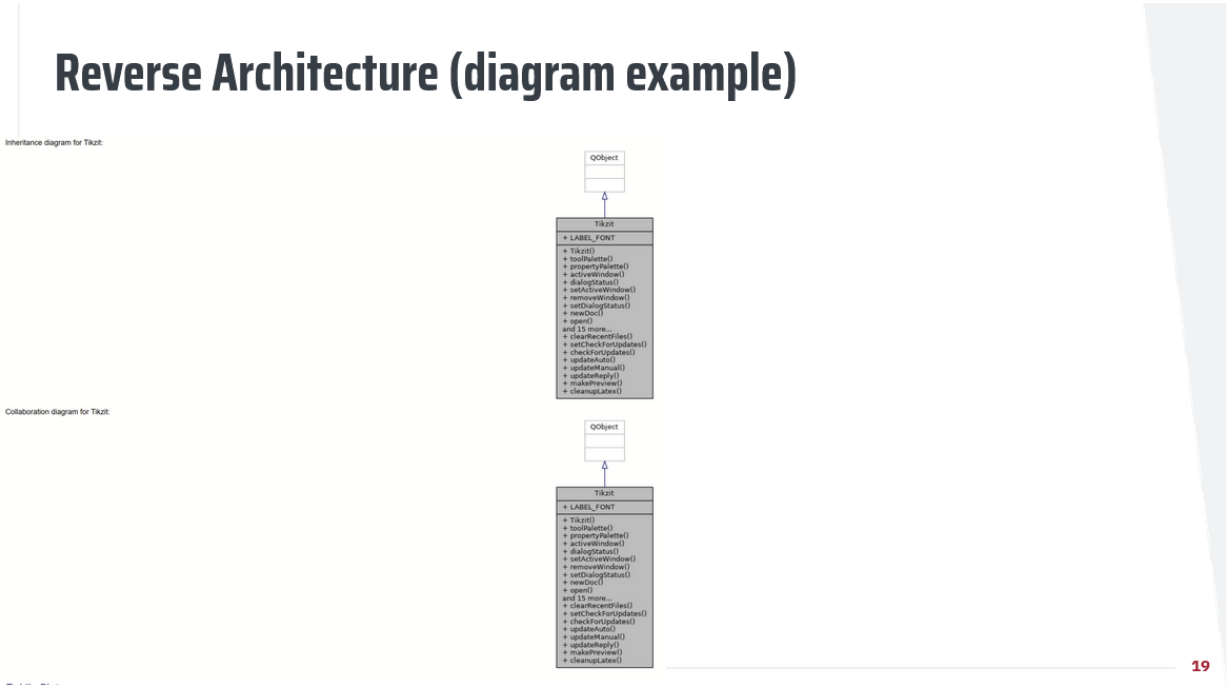

Figure 10.15: Example of Doxygen's Created Diagram

The above image is just an example diagram that Doxygen created from our app.

10.8 Current Work

The following is a list of items that we are currently working on:

- Updating Performance of Web Based Desktop App
- Guide inclusions and features
- Updating tikzit with preloaded items
- Full initial prototype requirements list for next semester
- Testing & feedback to refine requirements
- Stabilizing deployment & devops pipeline

10.9 Future of Project

The following is a list of items that we will be working on in the future:

- Update UI

- Better Login for Multi Users
- UML Additions to TikZiT
- EPS Exportation
- Mac os and Windows os support
- noVNC easy file transfer to local computer
- Download app on any OS system

Appendix A

Weekly Reports

A.1 Week Report 13 (12/12/2025)

During this past week, we focused on preparing a complete product demonstration and ensuring that our GitHub pipeline was fully functional. This included verifying that automated workflows run correctly and that the demonstration clearly shows the current state and capabilities of the project.

Going forward, we will focus on improving platform compatibility and clarifying our remaining deliverables. In particular, we will work on getting the project running properly on macOS and defining clear, achievable goals for the end-of-May deliverables to ensure a smooth wrap-up of the project.

What we have done:

- Created a product demonstration
- Got the GitHub pipeline working

What we are going to do:

- Work on getting the macOS version working
- Define clear goals for end-of-May deliverables

A.2 Week Report 12 (12/5/2025)

During this past week, we focused on the DevOps side of the project, which included auto-updating the version number, setting up Doxygen, and setting up GitHub, which are some essential steps moving toward the goal of getting the project working on the web. We also got the app to work online through a domain. We also implemented reverse architecture through Doxygen in our project.

Next week, we will focus on preparing the last Demo and be ready to go. We would also start working on the final Preliminary Implementation Report and update the Overleaf as needed.

What we have done:

- Added auto versioning
- Updated back-end development
- Setting up GitHub
- Online web with domain
- Implemented Doxygen

What we are going to do:

- Prepare presentation for last Demo
- Start looking at the final Preliminary Implementation Report.

A.3 Week Report 11 (11/13/2025)

During this past week, we updated our document to have a version number included. We also added the ui design and updated acitivty diagrams. In the background, we have updated the project to allow for potential web application usage. We also managed to create an image of TikZiT that, for now, works locally on a web browser.

Next week, we will hope to have a presentation for the 3rd demo ready to go, as well as fix any mistakes regarding the things presented in said demo.

What we have done:

- Added versioning
- Added Ui design
- Updated back end development
- Local web browser TikZiT

What we are going to do:

- Prepare presentation for demo 3
- Fix anything that comes up during demo 3 that appears to need fixing.

A.4 Week Report 10 (11/06/2025)

During this past week, we updated our document to allow for dynamic updating with our requirements and use cases, as well as changing the labeling to better fit our style. We also got tikzit working with linux, and are getting a better understanding of the code

Next week, we will further analyze the code for TikZit and continue working toward the prototype of the product. We are also planning on updating our diagrams to be more accurate. Creating an image of TikZiT that would work online would also be our goal throughought the next weeks.

What we have done:

- Got tikzit working with linux
- Worked on tikzit code
- Updated requirement back end

What we are going to do:

- Further work on TikZiT baseline
- Updating diagrams.
- Creating a TikZiT image to launch on Digital Ocean Docker Droplet

A.5 Week Report 9 (10/31/2025)

During this past week, we focused on finalizing Chapter 6 and strengthening requirement traceability. We completed and inserted the use case and activity diagrams for UC-01, UC-02, and UC-03, and wrote full use-case specifications for the use cases. We also reviewed Tables in Chapter 2 to confirm UR/SR/DR/NFR coverage and tightened references across the document.

Next week, we will further analyze the code for TikZit and start working toward the prototype of the product. We also planning on make the links for the use cases and requirements dynamic for easier use and link in the future.

What we have done:

- Added Use Case Diagrams and Activity Diagrams for all use cases.
- Wrote detailed Use Case Specifications for all use cases.
- Updated user stories

What we are going to do:

- Further work on TikZiT baseline
- In overleaf, make the links dynamic for all the use cases and requirements
- Working toward making the prototype/version 1.0

A.6 Week Report 8 (10/24/2025)

During this past week, we made significant progress in refining our project documentation and tool setup. We expanded the list of requirements and categorized them into system, user, domain, and non-functional requirements. These were then linked to their corresponding user stories and use cases to ensure proper traceability. We also updated our user stories and stakeholder lists to better reflect the current project scope. In addition, we successfully got TikZiT working on Windows, allowing all members to modify and run the source code consistently.

Next week, we will continue refining our baseline version of TikZiT and further develop our activity and use case diagrams. We will also begin working on detailed UML diagrams to support the project documentation and design process.

What we have done:

- Added and categorized requirements (system, user, domain, non-functional)
- Linked requirements to user stories and use cases
- Updated user stories
- Updated stakeholders
- Successfully got TikZiT working on Windows

What we are going to do:

- Further work on TikZiT baseline
- Continue developing activity and use case diagrams
- Begin working on UML diagrams for the project

A.7 Week Report 7 (10/17/2025)

During this past week, we have continued to work on making sure that the source code for TikZiT can be installed and modified on a Windows machine. We are starting to understand what would be the best way to install the source code and how to modify it in a way that works. We also started to work on activity diagrams that would be of use for our project.

Next week, we will make sure that all of us have a steady way to work on the code from the source code. We will continue to look through the code and modify it for our needs. We will also make it so our activity diagrams are ready to be completed and put into our documentation.

What we have done:

- Set up source code for Windows
- Work on activity diagrams
- Looked through code and understand what we need to change

What we are going to do:

- Finish up activity diagrams
- Get everyone up and working on the source code
- Try to correctly build a working modified TikZiT app

A.8 Week Report 6 (10/10/2025)

During this past week, we have added to the stakeholders list, worked on getting Tikzit to work on windows, and created a paper drawing of what the final code should result in for the player. We also started modifying and looking at the code, getting a small understanding of it.

Next week, we will continue to focus on getting the source code for TikZit on windows, further improve paper prototype, and further understanding of the code.

What we have done:

- Created a fully complete stakeholders list
- Created basic prototype for finalized product
- Worked directly with code for tikzit base

What we are going to do:

- Continue to work on getting windows started
- Continue to flesh out paper prototype
- Work on creating at least one feature

A.9 Week Report 5 (10/02/2025)

During this past week, we had to work on a presentation, and in order to do that we ended up having to do a number of other things. This included working on a timeline, a clear list of roles and responsibilities, a development plan, coming up with our methods, and determining if an irb approval was needed. It was also spent turning everything into a presentation.

Next week, We will continue where we left off, since this week got derailed with catching up on things we missed and creating a presentation, so the to do list is the same.

What we have done:

- Came up with a list of responsibilities and who does what
- Created a timeline and development plan, as well as determining more details on how we will do this
- Creating a presentation for this information as well as the other information we had

What we are going to do:

- User story collection
- Deeper understanding of Tikzit and Tinytex code
- Basic prototyping

A.10 Week Report 4 (09/25/2025)

During this past week, we researched different tools and open source software to figure out how and where we want to start. Tikzit [1] is a drawing tool that works directly with tikz, and makes it easier to port to latex. The reason we have decided to start with Tikzit for our basis is two-fold. Firstly, because of that link to latex it will make porting to overleaf more simplified, as overleaf is based entirely around latex. Secondly, it is a powerful drawing tool in general, and has a lot of what we are working for, so using it as a base means some of the features in our needs and want list are already fulfilled, such as nearly infinite zooming and being able to create new "blocks" while having some preset. We recompiled the source code of Tikzit and got it working, and played around with it, trying to get an understanding of fully how it works both in practice and in the code. A preview function exists inside of Tikzit, but to get it working, we need a pdflatex compiler that would compile our latex diagrams directly in the app. We found that using Tinytex [2] works for our needs, but needs some tinkering to get it to fully function. We started to look into combining the two programs into one so that a user can get the diagrams and their previews all in one, especially since both of these programs are open source. We also worked on the presentation, fixing mistakes in the documentation, and also added some things like the glossary, and added some things to the bibliography. We also collected some user stories.

Next week, We will likely split into two camps, with half of our attention going to more user stories to get more defined ideas for what is needed, and the other is going to work with Tikzit/Tinytex more and start getting some coding done.

What we have done:

- Decided on bases for project
- Recompiled source code for Tikzit to get it working
- Got Tikzit preview to work with the help of Tinytex
- Fixed documentation and updated it where needed
- Added glossary and bibliography
- Worked on presentation
- User story collection

What we are going to do:

- User story collection
- Deeper understanding of Tikzit and Tinytex code
- Basic prototyping

A.11 Week Report 3 (09/18/2025)

During this past week, we officially wrote out the use case and requirement tables, linking them with each other, adding a glossary to keep track of unique terms for our project and report, made the weekly reports an appendix, added the team declaration including key drivers and constraints, as well as adding a constraints table for future use. Also fixed some bugs with the documentation in order to make our document for readable and concise. We also took a closer look at the open source code and umlet. We decided against rushing into coding based off last week because we wanted to be more prepared for when we actually begin without rushing in without a plan.

Next week, the focus will be on user stories, how the conversion might work between a graphic, convert it to tkz, and convert THAT into latex and eps, look into open source programs that know how to draw or let the person draw using JAVA. We are also going into the constraints more as well. Finally, we are hoping to have a list of specific sites we can use as a refrence, either for user stories or how to begin the project.

What we have done:

- Use case and requirements table
- Created a glossary
- Bug fixing with documents

What we are going to do:

- User story collection
- Look more into prototyping
- Refrences

A.12 Week Report 2 (09/12/2025)

During the past week our team concentrated on completing and refining the use case documentation for the MBG-EPS UML drawing tool. We carefully reviewed each user requirement, confirmed that every feature such as creating and exporting class, sequence, and activity diagrams is clearly tied to a corresponding use case, and ensured that the overall design remains practical for implementation.

Next week, we will turn our attention to planning the first coding sprint. We will begin coding the core application framework, set up the shared GitHub repository and testing environment, and start implementing basic UML element functionality. These next steps will mark our transition from documentation to active development and keep us on track with the project schedule.

A.13 Week Report 1 (09/05/2025)

This week we met up with our group members and created our Overleaf so that we can start documenting our Senior Project. We decided what we will be working on and how we will accomplish it. We looked into the project that allows a user to create a UML diagram that can then be exported into the EPS format. We are also thinking of making it a web based application.

Next we will start looking at the open source code of draw.io and umlet, as well as get a finalized plan so that by week 3 we can begin working in nearly full force. We will take a look at how the other applications work, what they do well, what they can do better, and how we might incorporate specific ideas. A list of action items. What we have done:

- Create team
- Created basic idea and plan
- Created initial document
- Collecting ideas and resources to use as a reference.

What we are going to do:

- Download the source code
- Create a list of specific features to include, or details from what we want from it
- Create a (tentative) schedule for when we need certain features or milestones met
- Create some diagrams for how the code should look and operate

One of the issues that we are experiencing is being able to use Overleaf with every member of our team. We are currently talking about it with the professor and will be able to get our Overleaf functioning in time to finish our assignments.

[3]

Glossary

Block A piece that can be dragged in the drag and drop system of our uml project. Essentially, this referes to any individual object the user may drag and edit. 10

Diagram A single individual result (essentially, a file) of our app. A diagram is essentially one single collection of blocks and whatever the user has added. 10, 11

Must This defines the first highest priority requirement. All of the tasks, requirements, or anything that is marked this way are build in the current version. 6, 7, 8

Should This defines the second highest priority requirement. The system should implement all of the tasks, requirements, or anything that is marked this way, but if resources are limited, it can be left out of the current version. Build in next version. 7, 8

Would This defines the lowest priority requirement. The system would like to implement all of the tasks, requirements, or anything that is marked this way, but only if resources are available. It can be left out of all future versions. 7, 8

Bibliography

[1] (2020) Tikzit. [Online]. Available: https://tikzit.github.io/#source

[2] (2025) Tinytex. [Online]. Available: https://yihui.org/tinytex/

[3] (2002) Umlet. [Online]. Available: https://umlet.com/

[4] (2002) Visual paradiam. [Online]. Available: https://www.visual-paradigm.com/

[5] (2008) Github. [Online]. Available: https://github.com/

Index