Flow

Requirements Standard Plan

Hussain Muhammed, Jeffrey Doan, Kevin Zhu, Chengze Zhao, Ethan Patterson

Version 0, 2025-10-11

Table of Contents

Control Information 2
Introduction 3
(G) Goals 4
(G.1) Context and Overall Objectives *4*
(G.2) Current situation *4*
(G.3) Expected Benefits *5*
(G.4) Functionality overview *5*
(G.5) High-level usage scenarios *5*
User creates a new note 5
User adds a text box 6
User modifies a text box 6
User adds a shape 6
User modifies a shape/group of shapes 6
User links two shapes in a diagram with an arrow 6
User saves a note 6
(G.6) Limitations and Exclusions *7*
(G.7) Stakeholders and requirements sources *7*
Direct Stakeholders 7
Indirect Stakeholders 8
Other Requirements Sources 8
(E) Environment 9
(E.1) Glossary *9*
(E.2) Components *9*
(E.3) Constraints *9*
(E.4) Assumptions *10*
(E.5) Effects *10*
(E.6) Invariants *10*
(S) System 11
(S.1) Components *11*
(S.2) Functionality *11*
(S.2.1) Editor 11
(S.2.2) Shortcut Command Processor 12
(S.2.3) Rendering Engine 12
(S.2.4) Document Management 13
(S.2.5) Configuration and Customization 13
(S.2.6) Collaboration and Export 13
(S.2.7) Basis for Design and Requirements Stability 14
(S.3) Interfaces *15*
(S.3.1) Editor Interface 15
(S.3.2) Shortcut Command Processor Interface 15
(S.3.3) Rendering Engine Interface 15

(S.3.4) Canvas Manager Interface . . . 16
(S.3.5) Data Management Interface . . . 16
(S.3.6) Configuration Interface . . . 16
(S.3.7) External Interfaces . . . 16
(S.4) Detailed usage scenarios . . . 17
System Diagram . . . 17
(S.5) Prioritization . . . 17
(S.5.1) Functional Requirements Prioritization . . . 17
(S.5.2) Non-Functional Requirements Prioritization . . . 18
(S.5.3) Interface and Scenario Prioritization . . . 19
(S.5.4) Prioritization Summary . . . 19
(S.5.5) Requirement Traceability Summary . . . 20
(S.6) Verification and acceptance criteria . . . 21
Acceptance Criteria . . . 21
(S.7) Security and Safety Requirements . . . 22
7.1 Access Requirements . . . 22
7.2 Storage Requirements . . . 22
7.3 Upload Requirements . . . 22
7.4 Integrity Requirements . . . 22
7.5 Code Requirements . . . 22
(P) Project . . . 23
(P.1) Roles and Personnel . . . 23
Team Lead - Ethan . . . 23
Keynoter - Jeffrey . . . 23
Lead Reviewer - Chengze . . . 23
Team Liaison - Hussain . . . 23
Meeting Chair - Kevin . . . 23
(P.2) Imposed Technical Choices . . . 23
Hardware Binding Technical Choices . . . 24
Hardware Binding Justification . . . 24
Software Binding Technical Choices . . . 24
Software Binding Justification . . . 24
(P.3) Schedule and milestones . . . 24
Milestone 1: Project Proposal & Team Formation (Week 1–4) . . . 24
Milestone 2: Requirements and Analysis (Week 5–6) . . . 25
Milestone 3: System Design, V&V + PoC Plan (Week 7–8) . . . 25
Milestone 4: UI/UX Design and Design Document (Week 9–10) . . . 25
Milestone 5: PoC Demos (Week 11–12) . . . 25
Milestone 6: UI Development + Authentication (Week 13–14) . . . 25
Milestone 7: Feature Implementation + Design Documentation (Week 15–17) . . . 25
Milestone 8: Final Polishing (Week 18–22) . . . 25
Milestone 9: Final Demo and Expo Preparation (Week 23–25) . . . 25
Milestone 10: Project Closure and Capstone Expo (Week 26) . . . 26

(P.4) Tasks and deliverables *26*
Milestone 1: Project Proposal & Team Formation (Week 1-4) 26
Milestone 2: Requirements and Analysis (Week 5-6) 26
Milestone 3: System Design, V&V + PoC Plan (Week 7-8) 26
Milestone 4: UI/UX Design and Design Document (Week 9-10) 27
Milestone 5: PoC Demos (Week 11-12) 27
Milestone 6: UI Development + Authentication (Week 13-14) 27
Milestone 7: Feature Implementation + Design Documentation (Week 15-17) 27
Milestone 8: Final Polishing (Week 18-22) 27
Milestone 9: Final Demo and Expo Preparation (Week 23-24) 28
Milestone 10: Project Closure and Capstone Expo (Week 26) 28
(P.5) Required technology elements *28*
(P.6) Risk and mitigation analysis *29*
(P.7) Requirements process and report *30*
References **32**

Academic Integrity Disclaimer

We would like to acknowledge that as a dedicated students of McMaster University, we have thoroughly read and comprehended the Academic Integrity Policy published by the university. We are committed to upholding the principles of academic honesty and integrity in all aspects of our educational journey. We understand the importance of acknowledging the work and ideas of others, and we pledge to ensure that all our academic endeavors are conducted with the utmost originality and compliance with the university's policy.

We affirm that the content presented in this document is entirely our own, and any external sources used have been appropriately cited and referenced.

Hussain Muhammed

As I submit my work, I, **Hussain Muhammed**, take full responsibility for the integrity of my work and promise to avoid any form of plagiarism, cheating, or dishonest behavior. This acknowledgment serves as a testament to my dedication to academic excellence and the fostering of a trustworthy academic community at McMaster University.

Jeffrey Doan

As I submit my work, I, **Jeffrey Doan**, take full responsibility for the integrity of my work and promise to avoid any form of plagiarism, cheating, or dishonest behavior. This acknowledgment serves as a testament to my dedication to academic excellence and the fostering of a trustworthy academic community at McMaster University.

Kevin Zhu

As I submit my work, I, **Kevin Zhu**, take full responsibility for the integrity of my work and promise to avoid any form of plagiarism, cheating, or dishonest behavior. This acknowledgment serves as a testament to my dedication to academic excellence and the fostering of a trustworthy academic community at McMaster University.

Chengze Zhao

As I submit my work, I, **Chengze Zhao**, take full responsibility for the integrity of my work and promise to avoid any form of plagiarism, cheating, or dishonest behavior. This acknowledgment serves as a testament to my dedication to academic excellence and the fostering of a trustworthy academic community at McMaster University.

Ethan Patterson

As I submit my work, I, **Ethan Patterson**, take full responsibility for the integrity of my work and promise to avoid any form of plagiarism, cheating, or dishonest behavior. This acknowledgment serves as a testament to my dedication to academic excellence and the fostering of a trustworthy academic community at McMaster University.

Control Information

Version	Delivery		Info	
	Deadline	*Delivered*	Developers	Change
V1	Oct 10, 2025	Oct 10, 2025	Hussain, Oliver, Jeff, Kevin, Ethan	Initial release
V2				
V3				

Introduction

We have chosen the Meyer template for our Software Requirements Specification document. While our document follows the provided template for the most part, the following changes were made:

- Added this Introduction section.
- Removed author biography section. We felt that this was unnecessary.
- Removed individual section control info. We removed these because we felt it was too granular for the changes and rounds of feedback we will be using for this document. The overall document control info is still included, but it was updated to include the "Developers" and "Change" columns.
- Added an S7.adoc under the system directory. This addition corresponds to the request made in the Hazard Analysis document to include the additional safety and security requirements made there into the SRS.

(G) Goals

(G.1) Context and Overall Objectives

Flow is designed to address gaps in existing note-taking systems, which lack efficient methods for integrating text and diagrams through a unified keyboard-driven workflow. Current solutions impose trade-offs in speed, flexibility, and workflow continuity, as explored in G.2.

The goal of Flow is to create an intuitive system that allows for keyboard-driven creation and editing of notes involving text and diagrams. Although there is an expectation that there will be a learning curve to become proficient with the system, the intended benefit is that the user will be able to create and edit notes more quickly than with existing systems.

(G.2) Current situation

Flow arises from the fact that students, especially in technical disciplines such as engineering, often need to create notes containing both structured text and quick sketches of diagrams like state machines, circuits, or tables. Often times these notes also need to be created in a timely manner, as lectures can move quickly.

There are many existing options for taking notes, each with their own trade-offs:

- Pen and paper
  - Not easily editable or shareable
  - Handwriting may be slower than typing
- Tablets with stylus support (e.g. iPad with Apple Pencil)
  - Can be expensive
  - Handwriting may be slower than typing
- Traditional text editors (e.g. Microsoft Word, Google Docs)
  - Limited support for creating diagrams within the app
- Mouse-driven editors (e.g. Draw.io, OneNote)
  - Can be slow to navigate menus for specific geometry
  - Limited keyboard shortcut support for creating and editing geometry
  - May require exporting and importing images into a separate text editor for note-taking (e.g. Draw.io)
- Text-based diagram languages (e.g. Mermaid, PlantUML)
  - Separate definition from rendering, forcing users to manage an inefficient write-compile-insert cycle
  - Requires integration with another text editor for note-taking text
- Document markup languages (e.g. LaTeX, Typst)
  - Limited to a more traditional page format rather than infinite workspace like OneNote for example
  - Editing underlying text rather than geometry itself

On top of inefficiencies to workflow, an important note is that none of these solutions are fully keyboard-driven. This is a key aspect of the Vim ideology that Flow aims to embody by providing a unified system for creation and editing of notes involving both text and diagrams exclusively through the keyboard.

(G.3) Expected Benefits

There are two main expected benefits that will be delivered by Flow.

The first is that the system will provide a new interface for creating multimedia (i.e. text and diagrams) notes using only a keyboard. This will include both the creation of note files from the user interface perspective and the content in a note. This benefits the user as it provides a keyboard-only interface for creating and managing multimedia notes, which does not currently exist for this type of system.

By extension of the first, the second expected benefit is to provide the user a potential boost to their note taking speed due to the keyboard-driven nature of the system and the support for shortcuts. This assumes that at full proficiency, a user can expect to be quicker at taking simple notes with Flow than with other existing solutions outlined in G.1. For our purposes, proficiency can be defined as completion of the user manual and 10+ hours of usage. An example of a simple note can be described as a finite state machine with 5 states, at least one arrow flowing in and out of each state, labels for states and state transitions, and some corresponding description paragraph in an arbitrary location around the state machine.

(G.4) Functionality overview

Firstly, Flow will allow users to do standard note-taking application operations such as creating, opening, editing, saving, and deleting notes. Additionally, users will be able to organize their notes into folders and subfolders for better management. Notes can contain both text and diagrams, and users will be able to format text (e.g. bold, italics, underline) and create lists (e.g. bullet points, numbered lists).

Notes will be viewable as a canvas that the user can pan in all four directions (i.e. left, right, up, down). Text will live in designated text boxes where the user can insert/update/delete text. The text boxes can be moved around the canvas.

The user can insert predefined geometry as outlined in S.3. Geometry can be moved and resized. The basic "line" geometry can be used to create a sort of custom geometry in combination with other predefined geometry and lines. A combination of geometry can be selected and saved for reuse by the user. Users can define custom keyboard shortcuts mapped to custom geometry for quick insertion.

(G.5) High-level usage scenarios

User creates a new note

1. User opens the actions menu
2. User selects new note action
3. User inputs a name for the new note
4. System creates a new note file in the current working directory of the user
5. User receives new note confirmation and blank note is displayed

User adds a text box

1. User opens the insert menu
2. User selects add text box action
3. System adds a new text box to the note at the current cursor position
4. User receives confirmation that text box has been added as it is now viewable and editable

User modifies a text box

1. User selects a text box in the note
2. User modifies the text content of the text box
3. System updates the text box with the new content
4. User views the updated text box content

User adds a shape

1. User opens the insert menu
2. User selects shape option
3. System adds a new shape to the note at the current cursor position
4. User receives confirmation that shape has been added as it is now viewable and editable

User modifies a shape/group of shapes

1. User selects a shape/group of shapes in the note
2. User opens the edit menu
3. User selects edit action (e.g. resize, recolor, move)
4. System updates the selected shapes with the new properties
5. User views the updated shapes

User links two shapes in a diagram with an arrow

1. User selects the first shape in the note
2. User opens the insert menu
3. User selects add arrow action
4. User selects the second shape in the note
5. System adds an arrow linking the two shapes
6. User views the updated diagram with the new arrow

User saves a note

1. User opens the actions menu
2. User selects save note action
3. System saves the current state of the note to the file system
4. User receives confirmation that the note has been saved

(G.6) Limitations and Exclusions

While Flow is designed to increase note taking efficiency, it will not be responsible for actual note taking. That is, automated methods for note taking such as speech to text, image recognition, or AI (artificial intelligence) support are out of scope for this project. The system is designed to assist users in taking notes, but the actual content of the notes will be provided by the user.

Managing the hosting of notes is out of scope for this project. That is, the system will not be responsible for hosting notes on a server or cloud service. The system is designed to create and manage notes locally on the user's device, but the sharing of notes between other users or devices will be handled by the user through other means.

(G.7) Stakeholders and requirements sources

Broadly, the stakeholders for this project will all fall into a group of more technically savvy individuals who are likely comfortable using keyboard shortcuts in other familiar software programs. The target users should be able to memorize a combination of shortcuts and commands to align with the keyboard based usage.

- People who use keyboard based workflows and prefer not to use their mouse to take notes that involve simple diagrams.
- People who do not have access to a drawing tablet, but would like to take digital notes involving diagrams in an efficient way.
- People who often take notes on a computer and see value in learning a new system to improve their note taking efficiency.

These stakeholders may include students, software engineers, technical writers, and other professionals who frequently take notes that involve both text and diagrams. These stakeholders are the direct users of the system and characterized by the personas of John and Alice below.

Direct Stakeholders

John (Direct User)

John is a second year computer science student. He is familar with text editors and IDEs, as well as the the concept of keyboard shortcuts. He currently uses Google Docs on his computer to take notes during lectures. Recently his classes have involved more diagrams and visual elements, which he finds difficult to create in Google Docs. He is looking for a more efficient way to take notes that include both text and diagrams, ideally using his existing computer as he cannot afford to buy a new tablet device. He has configured his IDE with custom keyboard shortcuts and memorized preset shortcuts to speed up his coding. He would be open to learning a new system with similar functionality in a note taking application.

Alice (Direct User)

Alice is a senior software engineer. She often works on complex projects that involve multiple components that are documented with diagrams. She currently uses [Vim] and diagram description languages to create and manage her documentation notes. However, she finds it cumbersome to constantly rerender and then preview documents with her mouse in another window. She finds that it breaks from her preferred keyboard-driven workflow. She is extremely comfortable with modal editing and keyboard shortcuts, and would be willing to learn a new system if it could improve her efficiency and align with her existing workflow.

Indirect Stakeholders

- **Colleagues of direct users -** Colleagues of direct users may be affected by the system as they may need to collaborate on notes created by direct users or share notes with direct users.

Other Requirements Sources

- **Existing note taking applications -** Existing note taking applications can provide insights into the features and functionalities that users expect from a note taking system.
- **User feedback and reviews -** User feedback and reviews of existing note taking applications can provide insights into the strengths and weaknesses of these applications.

(E) Environment

(E.1) Glossary

- **Vim**: Vim is a highly configurable, keyboard-driven text editor built to make creating and changing any kind of text very efficient. [1]
- **UI/UX**: User Interface/User Experience.
- **PoC**: Proof of concept. Used to show feasibility of the project.
- **UML**: Unified Modeling Language. Modeling language used for diagrams to standardize modeling and minimize confusion from differences in models.
- **V&V**: Verification and validation. Testing that will be done to check if the project fulfills the requiremnts.
- **SRS** Software Requirements Specification. Document that contains requirements that will be used for the develoment of the project.

(E.2) Components

- **File system**: The system needs a to installed on a system with a file system to store files needed for the application. It will also store files holding notes, letting the user access them later.
- **Keyboard/Mouse/Trackpad/Touchscreen**: The system will use keyboard and mouse for input. These are the standard and primary input for computers. Other methods such as trackpad and touchscreen will be considered similar to mouse input and not prioritized.
- **Display**: As a computer program, the system will display its outputs with a corresponding screen. It should be able to use this display to show the user anything it need, ex current note, errors, tools, etc.

(E.3) Constraints

- **Must be compatible with at least Windows/MacOS/Linux.**: Windows/Mac/Linux are the main operating systems used by computers and correspondingly are the systems that the program must be able to run on. Priority will be given to the operating system that is the most popular, Windows then MacOS then Linux.
- **Must not need unauthorized modifications to data and other systems on device.**: Data and systems may require authorization to view or modify. To prevent errors or distrust in the program it is best to not modify files for other programs.
- **Must not break any laws (Copyrights, Digital security, etc).**: As software, this program needs to follow all laws and regulations relating to laws. These will typically be Copyright/Licensing laws and Digital Security laws. This means that actions such as collecting personal information or using other code must be checked for following the regulation for where we intend to distribute this software.

(E.4) Assumptions

- **User is understanding of how to download/install basic windows programs.**: As we cannot be there to install this program on every computer it will run on, we assume that the user is able to install it by themselves provided they are provided an installer program file.
- **User understands basic computer I/O (Keyboard, mouse, screen).**: To properly intact with the program users would be expected to know how to use their keyboard and mouse to navigate the program. This would include keyboard shortcuts.
- **User not needlessly interact with Files in the system.**: The Program will store taken note files on the computer. It is assumed that the user will not do changes to files that may cause issues when reading the files.
- **Device being run on has proper libraries or user can install them when provided with redists.**: If our program uses other libraries, they need to install for it to work. If the user can install the program than it is assumed that the user can also install required libraires. These libraries need to be provided either as a list or as redistributable files.
- **The device running the program has enough space for the program to function.**: The program will require drive space to store the program and any notes created by the user. If there is not enough space the user should be notified, and the program should continue to hold the current note.

(E.5) Effects

- **Requires storage space from computer (Lower computer storage space).**: The program will require space on the computer drive to both store the program and any notes that are taken. If the computer does not have the required space then the program may not work properly.

(E.6) Invariants

- **The system must preserve notes unless the user requests they be deleted.**: As notes are the primary goal of the system any notes created should stay open unless they are saved or the user decides that they can be deleted.

(S) System

(S.1) Components

The Flow system relies on the following software components:

- **Frontend User Interface** – Provides an interactive canvas-based workspace where users can create, edit, and navigate notes. Supports text formatting, folder organization, and insertion of geometric shapes.
- **Canvas Manager** – Handles spatial organization and user interaction for note elements (text boxes and shapes). Supports panning, positioning, resizing, and movement of objects on the canvas.
- **Rendering Engine** – Renders text and geometric shapes in real time, maintaining visual accuracy during edits, movements, and scaling. Ensures smooth visual updates and consistent element alignment.
- **Shortcut Command Processor** – Interprets keyboard inputs to trigger editing, navigation, and formatting actions. Supports customizable keybindings, macros, and command conflict detection.
- **Data Management and Storage** – Manages creation, retrieval, autosave, and deletion of notes. Maintains version history and folder hierarchies while ensuring data integrity.
- **Configuration and Customization** – Stores and applies user preferences such as layout, theme, and shortcut mappings. Provides mechanisms to reset or restore default settings.
- **Backend Logic and API Layer** – Handles local persistence and, where applicable, provides an interface for synchronization or external service integration. Ensures consistent behavior across user sessions.

(S.2) Functionality

(S.2.1) Editor

Functional Requirements

1. **Create and edit notes**: The Editor shall allow users to create, open, edit, save, and delete notes. (F211)
2. **Organize notes**: The Editor shall allow users to organize notes into folders and subfolders for better management. (F212)
3. **Format text**: The Editor shall support text formatting (bold, italics, underline) and list creation (bulleted and numbered). (F213)
4. **Insert text boxes**: The Editor shall allow users to create and move text boxes within the canvas. (F214)
5. **Insert shapes**: The Editor shall allow users to insert predefined geometric shapes (circle, rectangle, triangle, hexagon, octagon, star, arrow). (F215)
6. **Reposition and resize elements**: The Editor shall allow text boxes and shapes to be freely moved and resized within the canvas. (F216)
7. **Canvas navigation**: The Editor shall allow users to pan across the workspace in all directions. (F217)

Non-Functional Requirements

1. **Usability**: The Editor shall maintain a clean and intuitive layout to minimize user confusion. (NF211)
2. **Responsiveness**: Canvas interactions (e.g., drag, resize, type) shall respond with less than 50ms latency. (NF212)
3. **Accessibility**: The interface shall comply with WCAG accessibility standards, including keyboard-only navigation. (NF213)

(S.2.2) Shortcut Command Processor

Functional Requirements

1. **Command handling**: The Shortcut Processor shall interpret keyboard commands to perform editor actions. (F221)
2. **Custom keybindings**: Users shall be able to define custom keybindings for supported actions. (F222)
3. **Conflict detection**: The system shall detect and alert users when keybinding conflicts occur. (F223)
4. **Macro support**: The system shall support macros for chaining multiple actions into a single shortcut. (F224)

Non-Functional Requirements

1. **Low latency**: Command input shall execute within 30ms of user keypress. (NF221)
2. **Determinism**: Shortcut actions shall always yield consistent results. (NF222)
3. **Discoverability**: The system shall provide a reference menu or help overlay for all available shortcuts. (NF223)

(S.2.3) Rendering Engine

Functional Requirements

1. **Render text**: The Rendering Engine shall display formatted text accurately within text boxes. (F231)
2. **Render shapes**: The Rendering Engine shall render all supported geometric shapes with smooth scaling and edges. (F232)
3. **Layer management**: The Rendering Engine shall manage draw order for overlapping elements (text, shapes). (F233)
4. **Viewport control**: The Rendering Engine shall maintain a consistent visual experience while panning or zooming. (F234)
5. **Real-time updates**: Rendering updates shall occur seamlessly as the user types, moves, or resizes elements. (F235)

Non-Functional Requirements

1. **Performance**: Rendering shall maintain a minimum of 60 frames per second for at least 95% of the active usage time under typical workloads. (NF231)

2. **Consistency**: Shapes and text shall retain consistent proportions and alignment across devices. (NF232)
3. **Scalability**: Rendering shall handle up to 500 on-canvas elements without major performance degradation. (NF233)

(S.2.4) Document Management

Functional Requirements

1. **File operations**: The system shall allow users to save, open, rename, and delete notes. (F241)
2. **Autosave**: The system shall autosave changes at regular intervals (≤60 seconds). (F242)
3. **Export/Import**: The system shall support exporting notes to standard formats (e.g., PDF, Markdown) and importing from compatible sources. (F243)
4. **Version history**: The system shall maintain previous versions of a note for rollback. (F244)
5. **Folder synchronization**: The system shall maintain logical consistency between folder hierarchies and saved note structures. (F245)

Non-Functional Requirements

1. **Data integrity**: Files shall be protected from corruption or loss during save operations. (NF241)
2. **Storage efficiency**: Notes shall use minimal disk space through compression and text-based storage. (NF242)
3. **Compatibility**: Exported documents shall remain visually consistent across standard PDF viewers. (NF243)

(S.2.5) Configuration and Customization

Functional Requirements

1. **User preferences**: The system shall store preferences such as theme, layout, and keybindings. (F251)
2. **Profile management**: The system shall allow users to reset preferences or restore defaults. (F252)
3. **Session persistence**: The system shall remember the last open note and workspace view between sessions. (F253)

Non-Functional Requirements

1. **Security**: Preference data shall be stored locally and securely. (NF251)
2. **Consistency**: Preferences shall be consistently applied across all sessions. (NF252)

(S.2.6) Collaboration and Export

Functional Requirements

1. **Sharing**: The system shall allow users to share notes by exporting and sending files. (F261)
2. **Clipboard operations**: The system shall support standard clipboard functions (copy, cut, paste) for both text and shapes. (F262)
3. **Integration hooks**: The system shall provide a basic API for external extensions (e.g., plugins or synchronization services). (F263)

Non-Functional Requirements

1. **Fidelity**: Exported and shared notes shall retain accurate formatting and positioning. (NF261)
2. **Portability**: Shared files shall be viewable on other supported devices without requiring modification. (NF262)

(S.2.7) Basis for Design and Requirements Stability

This section makes inputs/outputs explicit and classifies requirements by likelihood of change so an external team could design, build, and test from this document.

Inputs and Outputs (Overview)

Component	Inputs	Outputs
Editor	Keyboard shortcuts, text entry, pointer events	Updated canvas model (text blocks, shapes), selection state, feedback messages
Shortcut Command Processor	Key events, user keybinding config	Dispatched editor actions, conflict warnings/help overlay
Rendering Engine	Canvas model, viewport (pan/zoom), style/theme	On-screen frames, redraw events
Document Management	File paths, save/load/export/version requests	Persisted files, autosave snapshots, exported PDFs/PNGs/Markdown, version IDs
Configuration & Customization	Preference changes (theme, keybindings), session state	Stored settings, applied themes/keymaps, restored workspace
Collaboration & Export	Share/export requests, clipboard ops	Shared/exported artifacts, clipboard content

Unlikely to Change (Stable Baseline)

- **Core editing & layout:** [F211], [F212], [F214], [F215], [F216]
- **Navigation & rendering basics:** [F217], [F231]–[F235]
- **File operations & autosave:** [F241], [F242], [F243]
- **Reliability & integrity:** [NF241], [NF215] (via [F241]–[F244])

- **Responsiveness & usability:** [NF211], [NF212], [NF214] These define the essential product behavior and quality bar and are expected to remain stable across the project.

Likely to Change (Evolving / Context-Dependent)

- **Advanced commands & customization:** [F222], [F223], [F224], [F252]
- **Export surface & integration hooks:** [F243], [F261], [F263] (formats/APIs may evolve with feedback)
- **Versioning policy & folder sync details:** [F244], [F245]
- **Security & scalability targets (if sync or large docs expand):** [NF216], [NF233], [NF218], cross-platform polish [NF213] These items depend on user feedback, platform constraints, or future scope and may be refined iteratively.

(S.3) Interfaces

(S.3.1) Editor Interface

The Editor serves as the main user interaction layer, capturing keyboard input and reflecting changes on the canvas. It connects with the Shortcut Processor to interpret keystrokes and the Canvas Manager to handle spatial arrangement.

- Communicates with the **Shortcut Processor** to receive command execution signals (e.g., insert text, create shape).
- Interacts with the **Canvas Manager** for element positioning, resizing, and movement.
- Sends update requests to the **Rendering Engine** to refresh visual content on screen.

(S.3.2) Shortcut Command Processor Interface

The Shortcut Processor translates user keystrokes into actionable commands for other modules. It ensures that all user actions can be performed through keyboard interactions.

- Receives raw key inputs from the **Editor** and interprets them based on predefined command bindings.
- Issues editing and transformation commands to the **Canvas Manager** and **Editor**.
- Reads configuration settings (e.g., shortcut customization) from the **Configuration module**.

(S.3.3) Rendering Engine Interface

The Rendering Engine is responsible for visually displaying both text and diagram elements on the canvas.

- Receives the scene graph and layout information from the **Canvas Manager**.
- Communicates with the **Editor** to reflect real-time updates (e.g., text entry, shape manipulation).
- Returns rendering feedback such as layout dimensions or bounding boxes to support editing operations.

(S.3.4) Canvas Manager Interface

The Canvas Manager controls element layout and manages spatial operations within the canvas.

- Receives editing and movement commands from the **Editor** and **Shortcut Processor**.
- Sends structured layout and positioning data to the **Rendering Engine** for drawing.
- Interacts with **Data Management** to store and retrieve geometric and positional data.

(S.3.5) Data Management Interface

The Data Management module stores all user data including notes, diagrams, and folder structures. It provides a consistent API for data access and persistence.

- Provides read/write access to documents for the **Editor** and **Canvas Manager**.
- Exposes an API for file operations such as create, open, save, delete, and export.
- Retrieves configuration and user profile data from the **Configuration module**.

(S.3.6) Configuration Interface

The Configuration component manages user preferences such as theme, keybindings, and autosave intervals.

- Provides user preferences and settings to the **Editor**, **Shortcut Processor**, and **Rendering Engine**.
- Receives updates from **Data Management** to persist modified configuration values.

(S.3.7) External Interfaces

Flow interacts with several external systems to enhance usability and persistence.

- **User Interface:** Keyboard-centric interaction; supports optional pointer input for selection and navigation.
- **File System:** Enables saving, opening, and exporting notes to supported formats (e.g., `.flow`, `.pdf`, `.png`).
- **Clipboard:** Allows copying and pasting of text and shape objects both within the app and across other applications.
- **Operating System:** Manages window resizing, DPI scaling, and accessibility features such as high-contrast mode.

Filesystem - Read/Write Access: open/save documents, export artifacts (PDF/PNG/Markdown), import existing files.

Clipboard - Inter-app Copy/Paste: text (UTF-8/RTF) and images for exported selections; intra-app copy/paste for shapes and groups.

Operating System - Windowing & Display: resize, DPI scaling, high-contrast settings propagated to Rendering/Editor.

(S.4) Detailed usage scenarios

Flow follows a modular, layered architecture that separates user interaction, core logic, and data management to support extensibility, maintainability, and responsive performance. The system consists entirely of software components, and all user operations occur within the text-based diagramming interface.

The architecture can be divided into three main layers:

- **Frontend Layer** This layer manages all user interactions. It includes:
  - **Editor Interface**: Displays the workspace where users type, draw, and manipulate diagrams.
  - **Canvas Manager**: Handles graphical positioning, panning, and zooming of text and shapes.
  - **Shortcut Processor**: Captures keyboard input, interprets commands, and routes them to the correct subsystems.
- **Core Processing Layer** The middle layer contains the system logic that manages document data and rendering. It includes:
  - **Rendering Engine**: Converts text and geometric descriptions into visible diagrams on the canvas.
  - **Data Manager**: Handles creation, update, deletion, and organization of notes, shapes, and folders.
  - **Configuration Manager**: Stores user preferences, key bindings, and editor layout settings.
- **Storage and Integration Layer** This layer manages persistence and external communication:
  - **Local File System**: Responsible for reading/writing files (e.g., saving and loading notes).
  - **Cloud Sync** (future extension): Allows sharing and synchronization of documents across devices.

Together, these components ensure that user inputs are processed efficiently, documents are rendered accurately, and all data is persistently stored.

System Diagram

The following figure illustrates a finite state machine representation of the mode and action states a user transitions through when creating and editing elements such as shapes or text boxes. The actual keys or triggers for these transitions may vary, and certain side effects—like rendering the completed element after input is processed—are not explicitly depicted.

[S.4] | *models/S.4.png*

Figure 1. Activity diagram for Flow user interaction flow

(S.5) Prioritization

(S.5.1) Functional Requirements Prioritization

Table 1. Activity Diagram for Flow User Interaction Flow

Requirement ID	Description	Priority	Rationale
F211	Create, open, edit, and delete notes	**High**	Fundamental to all system operations; without it, the app cannot fulfill its main purpose.
F212	Organize notes into folders and subfolders	**High**	Core for content management and navigation.
F213	Insert text boxes and edit content	**High**	Enables primary user interaction for note-taking.
F214	Insert predefined geometric shapes (circle, rectangle, etc.)	**High**	Core to diagramming capability and differentiates the app from plain text editors.
F215	Move, resize, and position text boxes or shapes	**High**	Essential for flexible diagram creation.
F216	Pan and zoom canvas	**Medium**	Important for large diagrams but not critical to minimal function.
F217	Save and load notes from local storage	**High**	Data persistence and continuity.
F218	Export notes (PDF, PNG, Markdown)	**Medium**	Important for sharing, but not needed for MVP.
F219	Undo/redo editing actions	**Medium**	Improves usability; can be implemented incrementally.
F220	Optional: sync or backup to cloud	**Low**	Value-add feature, non-critical for core offline operation.

(S.5.2) Non-Functional Requirements Prioritization

Requirement ID	Description	Priority	Rationale
NF211	Low-latency input and rendering ($\leq$30ms response)	**High**	Critical for responsiveness and user satisfaction.
NF212	Local data persistence and autosave	**High**	Prevents data loss; essential for reliability.

Requirement ID	Description	Priority	Rationale
NF213	Cross-platform support (Windows/macOS/Linux)	**Medium**	Important for accessibility, can be phased after MVP.
NF214	Accessible keyboard shortcuts and discoverable command palette	**High**	Core to product identity and usability.
NF215	File integrity during save/export (atomic operations)	**High**	Ensures stability and prevents corruption.
NF216	Security of local data (no background network traffic)	**Medium**	Supports user privacy, non-blocking for MVP.
NF217	Visual consistency and intuitive UI layout	**Medium**	Improves user experience, can evolve iteratively.
NF218	Performance at scale (500+ elements)	**Low**	Optimization target for later releases.

(S.5.3) Interface and Scenario Prioritization

Interface / Scenario	Priority	Rationale
User interface for note editing (canvas, text boxes, shapes)	**High**	Central to all use cases.
File I/O interface (open/save/export)	**High**	Necessary for persistence.
Keyboard command interface	**High**	Defining feature of Flow; critical for workflow efficiency.
Shape palette and geometry insertion interface	**Medium**	Adds value but not blocking to MVP.
Optional sign-in/sync interface	**Low**	Useful enhancement but out of scope for current build.

(S.5.4) Prioritization Summary

High Priority

Core functionality necessary for a minimal viable product:

- Text and diagram creation/editing — F211–F215, NF211–NF214
- Local save/load and autosave — F217, NF212, NF215

- Keyboard command interface and discoverability

Medium Priority

Enhancements that improve usability and reach:

- Canvas navigation and export features — F216, F218
- Cross-platform compatibility and UI refinement — NF213, NF217

Low Priority

Future or extended features beyond MVP:

- Cloud sync and user profiles — F220
- Large document optimization — NF218
- Extended collaboration or plugin systems

(S.5.5) Requirement Traceability Summary

The following table summarizes cross-references between functional (F) and non-functional (NF) requirements, showing how system behaviors rely on supporting qualities such as performance, usability, and reliability.

Requirement ID	Related Requirements	Relationship Type
F211	NF211, NF214	Functional editing depends on responsive input and accessible keyboard controls.
F212	NF217	Folder organization relies on consistent and intuitive UI layout.
F213	NF211, NF214	Text editing requires low-latency input and keyboard shortcut support.
F214	NF211, NF217	Shape insertion depends on responsive rendering and consistent visuals.
F215	NF211	Moving and resizing objects depend on smooth frame updates and responsiveness.
F216	NF211, NF217	Canvas navigation performance linked to input responsiveness and UI clarity.
F217	NF212, NF215	Save/load operations depend on autosave and file integrity guarantees.

Requirement ID	Related Requirements	Relationship Type
F218	NF215	Exporting notes depends on reliable file handling.
F219	NF211, NF215	Undo/redo relies on efficient rendering and reliable state management.
F220	NF216	Cloud sync depends on secure and private local data handling.
NF211	F213–F219	Input latency affects all core editing functions.
NF212	F217	Autosave directly supports data persistence.
NF214	F211, F213	Keyboard accessibility enhances main editing workflow.
NF215	F217–F219	File integrity underpins all save/export functions.
NF217	F212, F214, F216	UI layout consistency supports organizational and diagramming tasks.

This matrix shows that nearly all functional behaviors are reinforced by one or more non-functional properties, ensuring a cohesive and reliable user experience for Flow.

(S.6) Verification and acceptance criteria

Verification ensures that Flow meets the requirements defined in S.2 and S.3, while acceptance confirms the system fulfills user expectations. Testing will be conducted through the following methods:

- **Unit Testing** – verify that each core module (e.g., note management, canvas engine, geometry rendering) works as intended.
- **Integration Testing** – ensure that modules interact correctly, such as between the text editor and the file system.
- **System Testing** – validate end-to-end workflows like creating, editing, and saving notes.
- **User Acceptance Testing (UAT)** – confirm usability and responsiveness meet project expectations.

Acceptance Criteria

Category	Criteria
Functional Requirements	All listed functionalities in S.2 perform correctly without errors or data loss.

Category	Criteria
Non-Functional Requirements	Application runs smoothly, responds within acceptable delay, and maintains usability.
Interfaces	User interface is consistent with design and intuitive to operate.
Reliability	The system preserves data correctly between sessions.

The project will be considered accepted when all core functions operate without critical issues, usability testing is satisfactory, and the implemented features align with user expectations and requirements.

(S.7) Security and Safety Requirements

7.1 Access Requirements

ACR1. Users will not be able to access documents and notes written by other users.

7.2 Storage Requirements

SR1. Users' notes will be stored locally on their devices.

7.3 Upload Requirements

UR1. Users will not be able to upload specific file types (i.e., .exe, .js, .bat, .sh) to prevent malicious file uploads. UR2. Uploads will be stored in a secure directory that is not publicly accessible.

7.4 Integrity Requirements

IR1. The application will not modify any user-written data unnecessarily.

7.5 Code Requirements

CR1. Code dependencies will be kept up to date. CR2. Debug logs and stack traces will not be exposed and will produce generic error messages instead.

(P) Project

(P.1) Roles and Personnel

Aside from every member contributing as Software Developers / Software Engineering Students, each person is responsible for a team/admin role.

Team Lead - Ethan

Qualifications - Must have past leadership experience in a technical environment - Familiar with meeting tools (i.e., When2Meet, Google Calendar, Teams) - Experience with task delegation

Responsibilities - Schedules meetings and events - Organizes tasks and deliverables

Keynoter - Jeffrey

Qualifications - Good organizational skills - Familiar with note-taking applications (i.e., Google Drive, OneNote, Notability)

Responsibilities - Takes notes for meetings - Keeps track of information for later use

Lead Reviewer - Chengze

Qualifications - Excellent knowledge of coding languages - Experience with code reviews and GitHub

Responsibilities - Default reviewer on GitHub if no other member is available - Ensures all PRs are reviewed properly

Team Liaison - Hussain

Qualifications - Good communication skills - Good organizational skills regarding emails and updates

Responsibilities - Main communication point between the team and external parties - Primary point of contact for TAs and the Professor

Meeting Chair - Kevin

Qualifications - Good time management skills - Experience with leading technical meetings

Responsibilities - Manages meeting topics and agenda - Ensures meetings stay on task and members are focused

(P.2) Imposed Technical Choices

Hardware Binding Technical Choices

- Must be used on a laptop, desktop, and/or personal computer
- Must mainly run using a keyboard with minimal to zero mouse/trackpad involvement
- Minimal System Requirements:
- CPU: 1.6 GHz or faster, 2-core processor OR Apple Silicon (M1/M2/M3)
- GPU: DirectX 9 or later WITH WDDM 2.0 DRIVER
- Memory (RAM): 4 GB RAM (64-bit); 2 GB RAM (32-bit)
- Storage: At least 5 GB of available disk space

Hardware Binding Justification

- The application must be easily accessible. Having Flow run on a variety of devices aligns with the company's goal of having an easily accessible note-taking application for all students in technical disciplines.
- Having lower system requirements ensures that the application reaches a wider group of students who usually cannot afford expensive setups and/or devices.
- The binding constraint of having mostly, if not only, keyboard involvement stems from the project's goal of developing a note-taking application which allows students to easily take notes and construct diagrams in class seamlessly using a keyboard.

Software Binding Technical Choices

- Must be a standalone application
- Must not be an extension to a pre-existing note-taking application
- Must be usable on iOS, Windows, and Linux-operated devices

Software Binding Justification

- Having a standalone application and not an extension is an imposed technical choice as it contributes to the capstone's requirement of meeting a set complexity level.
- This technical choice also aligns with the group's goal of having more freedom in the software's UI and keybindings. Being an extension restricts the available bindings accessible to our program.
- Having the software run on multiple operating systems aligns with the company's goal of reaching a wider audience of students and making the application more accessible.

(P.3) Schedule and milestones

Nothing available at this point.

Milestone 1: Project Proposal & Team Formation (Week 1–4)

Task: - Begin introductory functions for the project, such as the proposal and basic team formation.

Key Dates: - Teams formed, project proposal due — **Sept 15th, 2025** - Problem Statement, Goals, and Development Plan due — **Sept 22nd, 2025** - Peer Review (Problem Statement, [PoC], [V&V], Development

Plan) due — **Sept 24th, 2025**

Milestone 2: Requirements and Analysis (Week 5–6)

Task: - Complete the requirements documentation

Key Dates: - [SRS] + Hazard Analysis due — **Oct 10th, 2025**

Milestone 3: System Design, V&V + PoC Plan (Week 7–8)

Task: - Begin technical planning for the application and PoC

Key Dates: - [V&V] Plan due — **Oct 27th, 2025**

Milestone 4: UI/UX Design and Design Document (Week 9–10)

Task: - Begin UI/UX design (interface and front-end mockups) and design documentation

Key Dates: - Design Document due — **Nov 10th, 2025**

Milestone 5: PoC Demos (Week 11–12)

Task: - Develop and demo a [PoC]

Key Dates: - [PoC] demo dates — **Nov 17th, 19th, 20th, 24th, 26th, and 27th, 2025**

Milestone 6: UI Development + Authentication (Week 13–14)

Task: - UI build and basic user authentication

Key Dates: - Classes begin — **Jan 5th, 2026**

Milestone 7: Feature Implementation + Design Documentation (Week 15–17)

Task: - Implement core features - Continue refining and submitting updated design documentation

Key Dates: - Design Document due — **Jan 19th, 2026**

Milestone 8: Final Polishing (Week 18–22)

Task: - UI polish & bug fixing - Finalize demo day logistics and prepare for dry runs - Submit extras and V&V report - Final regression testing and documentation polish

Key Dates: - Extras and [V&V] due — **Mar 9th, 2026**

Milestone 9: Final Demo and Expo Preparation (Week 23–25)

Task: - Prepare for final demo and Capstone Expo

Key Dates: - Final Demos — **Mar 23rd, 25th, 26th, 28th, 29th, 2026** - Poster and Video due — **Apr 2nd, 2026**

Milestone 10: Project Closure and Capstone Expo (Week 26)

Task: - Submit final documentation - Submit course evaluation - Capstone Expo presentation

Key Dates: - Final Documentation due — **Apr 6th, 2026** - Capstone Expo — **Apr 7th, 2026** - Course Evaluation due — **Apr 8th, 2026**

(P.4) Tasks and deliverables

Note: The tasks listed under each milestone are listed in order of importance/priority. These tasks also include their key dates, which is an important factor we considered when figuring out priority. Additionally, the requirements associated with specific tasks are also listed as a sub-bullet and are linked to the respective requirement.

Milestone 1: Project Proposal & Team Formation (Week 1-4)

- Team Introduction
  - Sept 15th 2025
- Project Proposal
  - Sept 15th 2025
- Problem Statement, Goals, and Development Plan
  - Sept 22nd 2025
- Peer Review for this milestone
  - Sept 24th 2025

Milestone 2: Requirements and Analysis (Week 5-6)

- Create Software Requirements Specification (SRS)
  - Oct 10th 2025
- Conduct Hazard Analysis and document hazards
  - Oct 10th 2025

Milestone 3: System Design, V&V + PoC Plan (Week 7-8)

- Define app architecture
  - Oct 21st 2025
- PoC implementation timeline
  - Oct 24th 2025
- Create a verification and validation plan
  - Oct 27th 2025
- Create UML diagrams and data models
  - Oct 31st 2025

Milestone 4: UI/UX Design and Design Document (Week 9-10)

- Submit the design document with a feedback loop
  - Nov 10th 2025
- Mock up UI/UX designs with navigation flow
  - Nov 14th 2025

Milestone 5: PoC Demos (Week 11-12)

- Work on PoC and have a working demo prototype
  - i.e., note creation, keyboard shortcuts, storage, retrieval
  - Nov 17th, 19th, 20th, 24th, 26th, & 27th

Milestone 6: UI Development + Authentication (Week 13-14)

- Build UI based on the design from Milestone 4
  - Aligns with the requirements listed under S.2.5 Configuration and Customization
  - Nov 5th 2025
- Implement basic features and important engines
  - Aligns with the requirements listed under S.2.3 Rendering Engine and S.2.4 Document Management
  - Jan 9th 2026

Milestone 7: Feature Implementation + Design Documentation (Week 15-17)

- Submit the finalized design documentation
  - Jan 19th 2026
- Implement core features
  - Aligns with the requirements listed under S.2.1 Editor, S.2.2 Shortcut Command Processor
  - Jan 23rd 2026
- Add advanced features to the application
  - Aligns with the requirements listed under S.2.5 Configuration and Customizations and S.2.6 Collaboration and Export
  - Jan 30th 2026

Milestone 8: Final Polishing (Week 18-22)

- UI polish & bug fixing
  - Feb 11th 2026
- Finalize demo day logistics and prepare for dry runs
  - Mar 2nd 2026
- Submit extras and V&V report
  - Mar 9th 2026
- Final regression testing and documentation polish
  - Mar 13th 2026

Milestone 9: Final Demo and Expo Preparation (Week 23-24)

- Final demo run-through and practice sessions
    - Mar 20th 2026
- Present final demo
    - Mar 23rd, 25th, 26th, 28th, & 29th 2026
- Create a poster and a video for the Capstone Expo
    - Apr 2nd 2026

Milestone 10: Project Closure and Capstone Expo (Week 26)

- Submit final documentation
    - Apr 6th 2026
- Submit course evaluation
    - Apr 7th 2026
- Capstone Expo presentation
    - Apr 8th 2026

(P.5) Required technology elements

Throughout the development of Flow, a combination of software and hardware systems will be required to ensure effective implementation, testing, and deployment. While the system's core functionality of translating user keyboard input seamlessly into both text and visual diagrams can operate without specific frameworks, certain external technologies will be required for supporting successful user interaction, visualization, and overall system performance.

External software systems that are expected to be included are:

- **Graphical User Interface (GUI) Frameworks**: To create an intuitive and user-friendly interface, GUI frameworks (such as Electon) will be necessary. These frameworks facilitate the development of cross-platform applications, ensuring that Flow can run on various operating systems without any compatibility issues.
- **Graphics / Rendering Libraries**: For rendering shapes or visual diagrams, rendering libraries will be required. These libraries provide robust tools for creating dynamic and interactive visualizations, which are essential for the integration of diagrams into a note-taking environment for Flow.
- **Parser / DSL Libraries**: To interpret text-based commands and convert them into visual elements, parser libraries or domain-specific language (DSL) tools will be necessary. These libraries help in efficiently processing user input and translating it into the desired graphical representations.
- **Drawing Libraries**: To facilitate the creation and manipulation of custom diagrams, drawing libraries will be required. These libraries would offer functionalities for creating shapes, lines, and other graphical elements that users can interact with within the Flow application.

External hardware systems that may be required include:

- **Keyboard (Primary Input Device)**: As Flow is designed to translate keyboard input into text and diagrams, a standard keyboard will be essential for user interaction. The keyboard serves as the primary input device, allowing users to enter the commands and text efficiently.

(P.6) Risk and mitigation analysis

Just like any other software project, the Flow project faces potential risks that could impact its schedule, performance, or overall success. Identifying these risks early and developing mitigation strategies is crucial to make sure the project stays on track. The following are some of the key risks and their corresponding mitigation strategies:

Risk	Mitigation Strategy
Technical Challenges: As we work through complex features, we might encounter unforeseen technical difficulties and may find it hard to implement certain functionalities (such as integrating diagrams with text seamlessly). This could affect and extend Milestone 6 and/or Milestone 7 as seen in P.4 as these are the major development phases of the project.	**Prototyping and Research**: We will allocate time for prototyping and research in the early stages of development to explore potential technical challenges, such as the one mentioned, and their solutions. This will help us identify and address issues before they become major roadblocks.
Team Coordination and Workload Balance: With a team of multiple members, there is a risk of miscommunication or poor distribution of work, which could lead to delays or inconsistencies in the project.	**Regular Meetings and Clear Communication Channels**: Version control systems (e.g., GitHub) will be used to manage code contributions and track changes. Regular team meetings will be held to discuss progress, address any issues, and ensure everyone is aligned with the project goals.
Time constraints: Team members may have other commitments (e.g., academic, personal) that could limit their availability to work on the project, potentially leading to delays.	**Flexible Scheduling and Prioritization**: We will attempt to create a flexible schedule that leaves room for adjustments based on team members' availability. Prioritizing tasks and focusing on critical features (following the order of the tasks, as mentioned in P.4) will help ensure that essential components are completed on time.
Scope Creep: There is a risk of adding new features or changing requirements during the development process, which could lead to delays and increased complexity.	**Follow Requirements Document**: When facilitating the development process, we will adhere to the requirements outlined in this document. Any changes or additions to the project scope will be carefully evaluated and approved by the team before being implemented.
Project Quality Assurance: There is a risk that the final product may not meet the desired quality standards (e.g. latency is higher than mentioned in [NF211]), leading to user dissatisfaction or the need for extensive rework.	**Testing and Feedback Loops**: We will implement a robust testing process, including unit tests, integration tests, and user acceptance testing. Regular feedback loops with potential users will help us identify and address issues early in the development process.

By proactively identifying these risks and implementing the corresponding mitigation strategies, we aim to minimize potential disruptions and ensure the successful completion of the Flow project.

SWOT Analysis

Strengths	**Weaknesses**
• Innovative solution to a common problem, which is the difficulty of integrating diagrams with text seamlessly. This unique feature can provide users a more efficient way to create and manage their documents. • Strong potential user base, as many students and professionals require tools for creating documents with integrated diagrams.	• Limited initial resources, as the project is being developed by a small team of students, who may also have other academic commitments, which could impact the project's timeline and quality. • Potential technical challenges in implementing the seamless integration of diagrams with text, which may require significant research and development effort.
Opportunities	**Threats**
• Growing demand for productivity tools, as more people are working remotely and require efficient ways to create and manage their documents. • Potential for integration with other popular tools and platforms, which could enhance the functionality and reach of the Flow project.	• Competition from established players (e.g., Draw.io, Lucidchart) in the diagramming and document creation space, which may have more resources and a larger user base. • Technical challenges that could delay the project timeline or impact the quality of the final product.

(P.7) Requirements process and report

Requirements Process The requirements elicitation process was conducted in a simplified and collaborative manner, reflecting the project's scope and resources. The team collectively engaged in discussions to identify and document the essential features, goals, and requirements for the project.

The process involved the following steps:

- **Brainstorming Sessions**: Initial brainstorming sessions to gather the main ideas and requirements. Each member contributed their perspectives on what the project should achieve. Factors considered included key stakeholders, user needs, technical feasibility, and project risks.
- **Informal Discussions**: Ongoing informal discussions to refine and clarify the requirements. This included addressing any ambiguities and ensuring a shared understanding among team members. Finer details, including specific requirements and project scope were discussed and agreed upon.
- **Documentation**: The requirements were documented in a collaborative manner, with team members contributing to the creation of this Software Requirements Specification (SRS) document. The document was reviewed and updated as needed to reflect any changes or new insights gained during the process.
- **Feedback Integration**: Although formal stakeholder feedback was limited, the team acted as both developers and stakeholders, ensuring that the requirements aligned with the project's goals and constraints. Further feedback from peers and mentors will be sought in future iterations.

For future iterations, a more structured elicitation process may be adopted, including user research, prototyping, and systematic requirements validation to ensure comprehensive coverage of user needs and project goals.

References

- [1] Vim - the ubiquitous text editor. 2025. https://www.vim.org/