



ENGINEERING
Computing & Software

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
(G) Goals	3
<i>(G.1) Context and Overall Objectives</i>	<i>3</i>
<i>(G.2) Current situation</i>	<i>3</i>
<i>(G.3) Expected Benefits</i>	<i>4</i>
<i>(G.4) Functionality overview</i>	<i>4</i>
<i>(G.5) High-level usage scenarios</i>	<i>4</i>
User creates a new note	4
User adds a text box	5
User modifies a text box	5
User adds a shape	5
User modifies a shape/group of shapes	5
User links two shapes in a diagram with an arrow	5
User saves a note	5
<i>(G.6) Limitations and Exclusions</i>	<i>6</i>
<i>(G.7) Stakeholders and requirements sources</i>	<i>6</i>
Direct Stakeholders	6
Indirect Stakeholders	7
Other Requirements Sources	7
(E) Environment	8
<i>(E.1) Glossary</i>	<i>8</i>
<i>(E.2) Components</i>	<i>8</i>
<i>(E.3) Constraints</i>	<i>8</i>
<i>(E.4) Assumptions</i>	<i>9</i>
<i>(E.5) Effects</i>	<i>9</i>
<i>(E.6) Invariants</i>	<i>9</i>
(S) System	10
<i>(S.1) Components</i>	<i>10</i>
<i>(S.2) Functionality</i>	<i>10</i>
<i>(S.3) Interfaces</i>	<i>10</i>
(S.3.1) Editor Interface	10
(S.3.2) Shortcut Command Processor Interface	10
(S.3.3) Rendering Engine Interface	10
(S.3.4) Canvas Manager Interface	11
(S.3.5) Data Management Interface	11
(S.3.6) Configuration Interface	11
(S.3.7) External Interfaces	11
<i>(S.4) Detailed usage scenarios</i>	<i>12</i>
System Diagram	12
<i>(S.5) Prioritization</i>	<i>12</i>
(S.5.1) Functional Requirements Prioritization	12

(S.5.2) Non-Functional Requirements Prioritization	13
(S.5.3) Interface and Scenario Prioritization	14
(S.5.4) Prioritization Summary	14
(S.5.5) Requirement Traceability Summary	15
<i>(S.6) Verification and acceptance criteria</i>	<i>16</i>
Acceptance Criteria	16
(P) Project	18
<i>(P.1) Roles and Personnel</i>	<i>18</i>
Team Lead - Ethan	18
Keynoter - Jeffrey	18
Lead Reviewer - Chengze	18
Team Liaison - Hussain	18
Meeting Chair - Kevin	18
<i>(P.2) Imposed Technical Choices</i>	<i>18</i>
Hardware Binding Technical Choices	19
Hardware Binding Justification	19
Software Binding Technical Choices	19
Software Binding Justification	19
<i>(P.3) Schedule and milestones</i>	<i>19</i>
Milestone 1: Project Proposal & Team Formation (Week 1–4)	20
Milestone 2: Requirements and Analysis (Week 5–6)	21
Milestone 3: System Design, V&V + PoC Plan (Week 7–8)	22
Milestone 4: UI/UX Design and Design Document (Week 9–10)	23
Milestone 5: PoC Demos (Week 11–12)	24
Milestone 6: UI Development + Authentication (Week 13–14)	25
Milestone 7: Feature Implementation + Design Documentation (Week 15–17)	26
Milestone 8: Final Polishing (Week 18–22)	27
Milestone 9: Final Demo and Expo Preparation (Week 23–25)	28
Milestone 10: Project Closure and Capstone Expo (Week 26)	29
<i>(P.4) Tasks and deliverables</i>	<i>29</i>
Milestone 1: Project Proposal & Team Formation (Week 1-4)	30
Milestone 2: Requirements and Analysis (Week 5-6)	31
Milestone 3: System Design, V&V + PoC Plan (Week 7-8)	32
Milestone 4: UI/UX Design and Design Document (Week 9-10)	33
Milestone 5: PoC Demos (Week 11-12)	34
Milestone 6: UI Development + Authentication (Week 13-14)	35
Milestone 7: Feature Implementation + Design Documentation (Week 15-17)	36
Milestone 8: Final Polishing (Week 18-22)	37
Milestone 9: Final Demo and Expo Preparation (Week 23-24)	38
Milestone 10: Project Closure and Capstone Expo (Week 26)	39
<i>(P.5) Required technology elements</i>	<i>39</i>
<i>(P.6) Risk and mitigation analysis</i>	<i>39</i>
<i>(P.7) Requirements process and report</i>	<i>39</i>

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	
	<i>Deadline</i>	<i>Delivered</i>	Developers	Change
V1	Oct 10, 2025	Oct 10, 2025	Hussain, Oliver, Jeff, Kevin, Ethan	Initial release
V2				
V3				

(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 familiar with text editors and IDEs, as well as 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 requirements.
- **SRS** Software Requirements Specification. Document that contains requirements that will be used for the development 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 interact 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 then it is assumed that the user can also install required libraries. 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

🔄 Nothing available at this point.

(S.2) Functionality

🔄 Nothing available at this point.

(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 ($\leq 30\text{ms}$ 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.

(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

 Nothing available at this point.

(P.6) Risk and mitigation analysis

 Nothing available at this point.

(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/>