Hazard Analysis
Flow

Team 9, min-cut
Ethan Patterson
Hussain Muhammed
Jeffrey Doan
Kevin Zhu
Chengze Zhao

Table 1: Revision History

Date	**Developer(s)**	**Change**
Date1	Name(s)	Description of changes
Date2	Name(s)	Description of changes
...	...	...

Contents

1 Introduction

Analyzing the hazards associated with a system is an important step in developing a safe and reliable product. It helps identify and evaluate potential risks that could impact the safety or usability of the system. By conducting a hazard analysis before development, we can take proactive measures to mitigate these risks, and design preventative measures to ensure the system operates as intended.

For our note-taking system (Flow), this analysis focuses on hazards that may lead to data loss, perfomance issues, and security vulnerabilities. As Flowaims to procide fast keyword-based editing with real-time diagram rendering, it introduces potential risks in different areas including rendering performance, data integrity, and user experience.

The following sections of the document define the scope and objectives of the hazard analysis, describe the system boundaries and main components of Flow, and outline key assumptions that may influence saftey considerations. The core of the document presents a detailed Failure Mode and Effect Analysis (FMEA) table, which identifies potential hazards, their causes and effects. Finally, the document concludes with a summary of safety and security requirements, and provides a roadmap for implementing these requirements in future development phases.

2 Scope and Purpose of Hazard Analysis

The purpose of this hazard analysis is to identify and evaluate the potential risks that could impact the safety, security, and usability of Flow. Conducting this analysis in the early stages of development allows for us to anticipate possible failure points in the system and implement design strategies to prevent data loss, ensure stable performance, and maintain user confidence.

The text-editing interface, diagram rendering engine, and file storage system all make up the core components of Flow. This analysis will focus on both software related hazards, such as crashes, performance issues, and synchronization errors, as well as user-related risks, including data loss, security vulnerabilities, and unintuitive user interactions.

Losses that may be incurred due as a result of these hazards include:

- **Loss of user data** from failed saves, corruption, or unexpected application closures
- **Loss of productivity** due to lag, unresponsiveness or unintuitive interface or command design
- **Loss of reliability and maintainability** if the application is developed with technical dept or poor coding practices

- **Loss of user trust** due to repeated bugs, confusing features, or inconsistent behavior
- **Loss of security or privacy** if data is not properly protected or stored securely

The hazard analysis focuses on factors in the software domain only. While hardware failures (e.g., device crashes, power loss), as well as operating system issues (e.g., file system corruption, OS crashes) can impact the performance and reliability of Flow, these are outside the scope of this analysis. Flowwill be running in the user space and any potential problems occuring at the kernal level or below are will therefore not be considered. The analysis will assume that the underlying hardware and OS are functioning correctly, and will focus on hazards that can be directly attributed to the design and implementation of Flowitself.

3 System Boundaries and Components

Flowis a note-taking application that allows users to create, edit, and visualize text-based diagrams using only keyboard inputs. The boundaries of the system include all components directly related to the core functionality of the application, including taking user input, rendering diagrams, and file management. The system boundary does not include external dependencies such as the user's operating system, device hardware, or third-party file storage services such as cloud storage providers.

The main components of Flowinclude:

1. **User Interface (UI) Layer**
   - Handling user interactions, keyboard inputs, displaying updates
   - Showing live view of notes and diagrams
   - Providing feedback on actions (e.g., error messages)
2. **Command Processing Engine**
   - Parsing and interpreting user commands
   - Managing application state based on user inputs
   - Sending structured actions to the rendering and storage layers
3. **Diagram Rendering Engine**
   - Converting text-based diagram descriptions into visual formats
   - Handling shape layout, positioning, and connection to the rest of the note content
4. **File Management and Storage Layer**
   - Saving and loading notes and diagrams to/from disk

- Managing file formats and data serialization
- Ensuring data integrity during read/write operations

5. **Error Handling and State Management Module**
    - Detecting, logging, and managing errors across all components
    - Implementing recovery mechanisms and stability after crashes or unexpected shutdowns

In addition to these core components, Flowmay also depend on external libraries for specific functionalities, such as diagram rendering or parsing. To generate and display diagrams, the application may utilize a graphics library to allow for efficient rendering and manipulation of visual elements. For parsing user commands, a parsing library may be employed to simplify the interpretation of complex inputs. State management libraries may also be used to help maintain application state and ensure consistency across different components.

4 Critical Assumptions

Certain key assumptions will be made regarding the user and operating environment during the development and use of Flow. These assumptions include:

1. **User Competency:** It is assumed that users have a basic level of computer literacy and is familiar with computer opertations such as using a keyboard and mouse. Users are expected to follow reasonable usage patterns (e.g. not shutting down the application abruptly or attempting to manipulate internal files directly).
2. **File Access and Storage:** It is assumed that the file system used for storing notes and diagrams is reliable and free from corruption. Users are expected to have sufficient permissions to read/write files in the designated storage locations.
3. **Operating Environment:** It is assumed that Flowwill be run on a supported operating system without unexpected interruptions such as forced shutdowns, crashes, or power failures. The system is expected to have adequate disk space and memory to run the application smoothly.
4. **Software Dependencies:** It is assumed that any third-party libraries or frameworks used by Floware stable, compatible, and function as intended on the target operating systems.

5 Failure Mode and Effect Analysis

Table 3: Failure Mode and Effect Analysis (FMEA) for Flow

Design Function	**Failure Modes**	**Effect of failure**	**Causes for failure**	**Detection**	**Recommended actions**	**SR**
Diagram rendering and layout	Overlapping shapes, unreadable layout, slow rendering	Reduced usability, user spending more time rearranging rather than note taking	Poor algorithm layout, too large graphics	Creation shows long frame times, UI freezes, user complaints, perfomance test threshold time limit exceeded	Have Unit tests performance tests for frame time; Use worker threads for rendering; Set rendering performance limits (e.g. ¡100ms per update); Implement partial rendering (e.g. rendering visual portion first); Potentially user to pin diagram positions	(F232, NF231)
Command parsing	Parser misinterprets or fails to parse user input	Unexpected or no diagram output, lost note, user confusion/frustration	Unintuitive grammar, improper user input, unhandled exceptions	Parser exception and logs, failed rendering, regression tests failing on inputs	Define simple, well-documented grammar and implement intuitive parser with proper error message handling; Provide immediate syntax validation (e.g. highlight errors); Fuzz unit tests to handle any and all errors; On parse error, preserve raw text and do not delete it	(F221, NF221)
Load/Open note	Note fails to load, note partially loads, note produces wrong rendering	User cannot access previous work, user confusion/frustration, loss of user trust	Incompatible or invalid file format, deserialization error	Load exceptions, visual regression tests, user reports	File format version tag with schema validation during load; Save raw text of error log, tests for loading after saving	(F241, F243, F244, NF241)
Save note	File not saved, save interrupted	File missing or corrupted, Loss of notes or diagrams, decrease in user productivity, loss of user trust in application	Power loss mid-save, app crash during write, full file storage	Exceptions logged, file hash value (or checksum) mismatch, unexpected file size, user reported missing file	Implement autosaving or atomic saving (save to temp file and rename it); Verify with checksumes on save and load; Inform user of insufficient disk space before save; Unit tests for save routines	(F241, F243, F244, NF241)

Table 3: Failure Mode and Effect Analysis (FMEA) for Flow

Design Function	**Failure Modes**	**Effect of failure**	**Causes for failure**	**Detection**	**Recommended actions**	**SR**
Keybinding Commands	Key conflicts, lost keystrokes (commands done unexpectedly)	Inability to create note, user frustration, potential data loss from unintended commands done (e.g. Alt+F4)	Global shortcut overlapping (e.g. Ctrl+C natively means copy), race conditions in input handling	Unit tests for commands, integration tests under different environments, user reports	Allow for remappable keybinding (with safe default set); Expose preview of keybinds; Provide undo command / option	(F222, NF223)
Interacting with shapes or diagrams (Add Text, Move, etc)	Edit commands behave incorrectly, loss of shape selection, text addition is inconsistent/undesired	Wrong edits made, user frustration, possible data corruption if partial edits made or deletes of inccorect item	Text and visual model not in sync, race conditions in input handling, undo stack not properly managed	Unit tests for edit commands and sequences of commands, user reports	Implement robust undo/redo stack; Integration testing simulating keyboard macros; Sync checks for text and visual model; Provide visual feedback on selection and edits;	(F216, NF212)
Custom Geometry creation (user-defined shapes)	Custom shapes fail to render, Shapes cannot be reusedd reliably	User time wasted recreating shapes, loss of user trust in customization features	Improper serialization or deserialization, ambiguous grammar for custom shapes, render limitations	Save or load of custom shapes fails, unit tests for custom shape rendering and serialization, user reports	Define clear grammar for custom shapes; Test creating, saving, loading, and rendering custom shapes; Initially limit complexity of custom shapes (e.g. no nested shapes);	(F215)
Save custom geometry as reusable commands	Command fails to save, Command working incorrectly	Loss of user productivity, Command produces wrong output, parse crashes, user frustration	Poor macro serialization, syntax collisions, improper parsing	Macro creation errors logged, regression tests for macros, user reports	Isolate macros syntax from normal command syntax; Provide clear error messages on macro creation; Unit tests for macro creation, saving, loading, and execution;	(F251)
Overall performance / responsiveness vs alternatives	Easier and more intuitive to take notes on other apps, app feels slower than existing tools (e.g. laggy rendering)	Users revert to other tools, poor adoption of app	Inefficient algorithms, full rendering on every change, blocking main thread	User performance testing, user feedback, collecting performance metrics	Incrementally render (diagram first, then text); If document becomes large, only render visible portion; Use worker threads for rendering; Performance budgets for each operation (e.g. ¡100ms for rendering update); Measure against alternative apps in user tests	(NF 211, NF212)

6 Safety and Security Requirements

[Newly discovered requirements. These should also be added to the SRS. (A rationale design process how and why to fake it.) —SS]

7 Roadmap

[Which safety requirements will be implemented as part of the capstone timeline? Which requirements will be implemented in the future? —SS]

Appendix — Reflection

[Not required for CAS 741 —SS]

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?
    - Chengze - I found that our team collaborated effectively in dividing the work and keeping our analysis consistent across sections. We communicated clearly about potential hazards and quickly agreed on the scope and structure of the document. Using LaTeX templates also helped maintain a clean and professional format throughout.
    - Ethan -
    - Hussain -
    - Jeffrey -
    - Kevin -
2. What pain points did you experience during this deliverable, and how did you resolve them?
    - Chengze - A main difficulty was completing some parts of the hazard analysis without direct communication among team members, since it is closely related to the SRS content. Some sections depended on finalized requirements or component details that were still being developed. We resolved this by holding short sync meetings and clarifying assumptions so our analysis stayed consistent with the SRS.
    - Ethan -
    - Hussain -
    - Jeffrey -
    - Kevin -
3. Which of your listed risks had your team thought of before this deliverable, and which did you think of while doing this deliverable? For the latter ones (ones you thought of while doing the Hazard Analysis), how did they come about?

- Our team had already identified general risks such as data loss and performance issues early in the SRS phase. However, while doing this deliverable, we recognized new risks related to user input handling and rendering consistency in the canvas editor. These came up as we analyzed possible failure modes and realized how improper error handling or unexpected user actions could lead to crashes or corrupted note files.

4. Other than the risk of physical harm (some projects may not have any appreciable risks of this form), list at least 2 other types of risk in software products. Why are they important to consider?

    - Some important types of risk in software products are data security risks and usability risks. Data security risks, such as unauthorized access or data corruption, can lead to loss of user trust and legal issues. Usability risks, like confusing interfaces or unclear workflows, can reduce user satisfaction and prevent adoption of the software. Considering these risks early helps ensure both system reliability and a positive user experience. Another risk may be financial risk, where project delays or cost overruns could impact the viability of the product. If the project runs out of budget or takes too long to deliver, it may not be feasible to continue development or support.