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 non-intuitive 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 non-intuitive 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 2: 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	SR-RENDER-1
Date1	Name(s)	Description of changes	...	...	...	...
Date1	Name(s)	Description of changes	...	...	...	...
Date1	Name(s)	Description of changes	...	...	...	...
Date1	Name(s)	Description of changes	...	...	...	...
Date1	Name(s)	Description of changes	...	...	...	...
Date1	Name(s)	Description of changes	...	...	...	...
Date1	Name(s)	Description of changes	...	...	...	...
Date1	Name(s)	Description of changes	... 5	...	...	...
Date1	Name(s)	Description of changes	...	...	...	...

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?
2. What pain points did you experience during this deliverable, and how did you resolve them?
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?
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?