Hazard Analysis
Software Engineering

Team #23, Project Proxi
Savinay Chhabra
Amanbeer Singh Minhas
Gourob Podder
Ajay Singh Grewal

Table 1: Revision History

Date	**Developer(s)**	**Change**
10/10/2025	Amanbeer Minhas	Added Sections 2, 4 and reflection
10/10/2025	Gourob Podder	Added Sections 1, 3 and reflection
10/10/2025	Savinay Chhabra	Added Sections 5 and reflection
10/10/2025	Ajay Singh Grewal	Added Sections 6, 7 and reflection

Contents

1 Introduction

In the context of this project, a hazard is defined as any potential source of harm or adverse effect that could result from the use, misuse, or failure of the AI-powered assistive technology platform. Hazards may arise from interactions between the system and the user, the system and the computing environment, or from internal system errors. They can manifest as physical, cognitive, or informational risks, and may affect user safety, privacy, accessibility, or the integrity of computer operations.

By documenting hazards in a structured and traceable manner, the development team can make informed design and operational decisions, while also providing guidance for future maintenance, updates, and testing of the system.

2 Scope and Purpose of Hazard Analysis

The purpose of this hazard analysis is to identify and evaluate possible hazards linked to the Proxi system, a voice and text driven assistant designed to improve accessibility and productivity for all users. The scope of this analysis includes the software components of Proxi such as the voice and text input handling, MCP integration, system automation features, and interaction with the desktop operating system. It also covers potential issues related to data handling, user commands, and accessibility features. Hardware failures, external network infrastructure, and third-party service outages are outside the scope of this analysis. The main losses that could occur due to these hazards include accidental file deletion or modification, loss or exposure of user data, compromised accessibility for people with disabilities, and reduced user trust caused by incorrect or misleading actions. These issues could also lead to major productivity loss if important commands fail, run incorrectly, or if key accessibility features stop working as expected. Even though Proxi does not control hardware or other safety-critical devices, failures in the software could still cause real harm through data loss, security breaches, inefficiency, or user frustration. The goal of this hazard analysis is to find and understand these risks early in the project so that effective prevention and mitigation strategies can be planned and built into the design.

3 System Boundaries and Components

3.1 System Boundaries

The boundaries of the system are defined by its scope of control and interaction:

- The system accepts user input through **speech** or **text** interfaces.
- It processes these inputs internally using natural language understanding and task planning components.
- It interacts with external computer applications and operating system tools to perform user-requested actions (e.g., sending emails, opening documents, managing files).
- It outputs responses or task results back to the user through **text** or **speech synthesis**.

Elements outside the system boundary include:

- The underlying **operating system** and **hardware** on which the platform runs.
- Third-party applications or APIs that the platform may invoke.
- The user's physical hardware devices such as microphones, speakers, or assistive peripherals.
- Network infrastructure and external AI services (if used for speech recognition or task execution).

3.2 Major System Components

The system can be conceptually divided into the following major components:

1. **User Interface Module (UI):** Handles all communication with the user, including speech-to-text (STT) input, text input, and text-to-speech (TTS) or visual feedback output.
2. **Natural Language Understanding (NLU) Engine:** Interprets user intent from raw speech or text input and converts it into structured commands for the planning module.
3. **Task Planning and Agent Manager:** Responsible for decomposing user intent into executable subtasks, invoking specialized software agents, and monitoring task progress.
4. **Tool and System Integration Layer:** Provides controlled access to the computer's applications and resources (e.g., file manager, web browser, email client) through APIs or system commands.
5. **Knowledge Base and Context Manager:** Maintains context about the user's current session, preferences, and system state to support coherent interactions and task continuity.
6. **Data Storage and Logging Component:** Stores session logs, error reports, and user preferences while ensuring compliance with privacy and data protection requirements.
7. **Security and Permissions Module:** Manages access control, ensures safe execution of actions on behalf of the user, and prevents unauthorized or unsafe operations.

These components collectively form the system boundary for hazard analysis. Hazards may arise from failures, misuse, or unexpected interactions among these components or between the system and its external environment.

4 Critical Assumptions

The hazard analysis is based on a few key assumptions about how Proxi will be used and the environment it will run in:

1. The system will run on supported desktop operating systems like Windows, macOS, or Linux with the needed permissions already set.
2. Users will give clear and intentional commands. Some commands may be misunderstood, but we do not assume the system will be used in a harmful way.

3. The MCP integration layer, system APIs, and other tools Proxi works with are expected to behave as described, even though failures are still possible.

4. Network issues may slow down some features, but basic local actions and automation will still work without an internet connection.

5. A working microphone or similar input device is expected to be available and set up correctly for voice input on the user's device.

These assumptions help us focus on realistic hazards and plan how to handle them. They guide the analysis and reflect how Proxi is expected to work in everyday use.

5 Failure Mode and Effect Analysis

Table 2: Failure Modes and Effects Analysis (FMEA) Table

Component	Failure Mode	Effect of Failure	S	Cause of Failure	O	D	Recommended Action	SR Ref.	Ref. ID
Speech-to-Text (STT)	Misinterprets voice commands	Wrong action or no action taken	8	Background noise, accent, poor mic	5	6	Improve model, noise filtering, allow correction	POA.R.1	H1-1
Command Interpreter	Fails to understand user intent	Task not completed or incorrect action	7	Ambiguous phrasing or missing context	4	6	Add clarification prompts, context handling	FUNC.R.3	H2-1
MCP Integration Layer	Wrong tool triggered	Unintended app or function runs	8	Incorrect mapping, version mismatch	4	6	Validate inputs, add fallback defaults	SAF.R.2	H3-1
OS Automation	Command crashes or freezes	User tasks fail or system slows	6	Permission errors, unresponsive apps	5	5	Retry logic, timeouts, graceful error handling	ROFT.R.2	H4-1
Confirmation Layer	Fails to ask for confirmation	Accidental deletion or system changes	9	Logic bug or skipped validation	3	7	Mandatory confirmation before high-risk actions	SAF.R.1	H5-1
Accessibility Layer	Screen reader or assistive tools fail	Users with disabilities cannot use system	9	Missing WCAG compliance, poor ARIA	4	6	Test with AT, comply with WCAG 2.2 and AODA	ACC.R.2	H6-1
Text-to-Speech Feedback	Response delayed or missing	User repeats commands or is confused	5	High load, slow network, TTS errors	5	5	Optimize response, add status indicator	SAL.R.1	H7-1
Update & Deployment	New release breaks features	Commands stop working	6	Lack of regression testing	4	5	Automated tests, rollback support	LON.R.2	H8-1
Logging System	Missing logs or corrupted data	Harder to debug and trace issues	5	Logging disabled or storage full	4	5	Continuous logging, storage monitoring	ROFT.R.1	H9-1
OS Compatibility	Inconsistent performance across OS	Features fail on some platforms	7	API differences, OS permissions	4	5	Multi-OS testing, abstraction layers	EPE.R.1	H10-1

Legend:
S: Severity (1-10)
O: Occurrence (1-10)
D: Detection (1-10)

6 Safety and Security Requirements

6.1 Safety Requirements

Although Proxi doesn't interact with any hardware, a user's files or settings could still be impacted by a poorly written or confusing command. We'll keep things straightforward and cautious to keep people safe:

SAS.R.1 — Confirm before risky changes. For anything that can alter files or sensitive settings we should show a short message asking the user to press the approve button before proceeding.

Why: This prevents any accidental damage from such requests.

How we'll check: In testing, every destructive or high level action should have an approval.

SAS.R.2 — Easy undo for common file actions. When possible, there should be a simple undo option that uses the OS recycle bin or copy-back.

Why: This way there is recovery if Proxi makes a mistake.

How we'll check: Run tests that can restore the original file in at at least 7/10 of cases.

SAS.R.3 — Least privilege tool scopes. MCP tools run inside only folders and apps the user has chosen to allow.

Why: This limits from any bad commands affecting unrelated areas.

How we'll check: Make sure that any attempt to access something out of scope is first requested from the user.

SAS.R.4 — Pause on low confidence. If the confidence is low or there is any confusion from Proxi, it can ask a short question to the user before continuing forward.

Why: Don't guess when unsure it is better to ask if it is safe to proceed.

How we'll check: Can test and experiment with low-confidence inputs that never run without clarification/approval.

6.2 Security Requirements

We also want to avoid common security issues:

SC.R.1 — Validate inputs and use allow lists. Have clean paths, URLs, and arguments and only allow safe protocols. Random directory traversal should be blocked.

Why: This stops not needed injections and any path traversal bugs.

How we'll check: Can run tests that will check to see if there are any injection or traversing issues.

SC.R.2 — Ask before sending data out. Sending personal data needs consent from the user, and we should only send the least amount of data as possible. Only data needed to complete the task should be sent.

Why: This way the user controls what leaves their computer.

How we'll check: Test flows can show an approval step before any data is sent.

SC.R.3 — Encrypt everything in transit. Use TLS/HTTPS for all network traffic to maximize security.

Why: Protects against eavesdropping and man in the middle attacks.

How we'll check: The packet captures will show HTTPS only traffic. Any bad certifications will get simply rejected.

SC.R.4 — Keep dependencies clean. Keep libraries up to date and scan libraries before releasing for any vulnerabilities

Why: This minimizes the risk from any known dependency vulnerabilities.

How we'll check: Release checklist should have 0 major vulnerabilities.

7 Roadmap

What we'll ship during the capstone

These are the highest-impact items for safety and trust.

ID	What we'll build	When
SAS.R.1	Confirmation step for destructive/system actions	Sprint 1
SAS.R.4	Clarify when intent confidence is low	Sprint 1
SC.R.1	Input validation + allow-lists in all adapters	Sprint 1–2
SC.R.2	Consent flow for any external/network calls	Sprint 2
SAS.R.2	One-step undo for file moves/renames/deletes	Sprint 2
SC.R.3	HTTPS-only networking with certificate checks	Sprint 2

Deferred / later work

Still useful, but can land after the demo if time is tight.

ID	What	Why later
SAS.R.3	Deeper OS sandboxing beyond path scoping	More engineering effort; too complicated for current scope
SC.R.4	Automated vulnerabilities scanning in CI and signed releases	Better for long-term upkeep; not needed right now

Appendix — Reflection

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?

8 Reflections

8.1 Amanbeer Minhas Reflection

1. What went well while writing this deliverable?
   Writing this deliverable went well because we already had a clear idea of our system from the SRS and FMEA work. Breaking the project into components helped me think about hazards more systematically and kept the analysis organized. I also felt more confident in identifying potential risks and writing about them in a structured way.
2. What pain points did you experience during this deliverable, and how did you resolve them?
   One challenge was figuring out how detailed to make each section without overcomplicating the document. I also found it tricky to write about hazards without repeating the same points as the SRS. I resolved this by focusing on specific examples from our project, like MCP integration and OS automation, and connecting them directly to potential risks.
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?
   We had already thought about risks like incorrect command execution and permission issues earlier in the project. However, risks such as operating system compatibility problems and deployment-related failures came up while writing this hazard analysis. They became more obvious once we broke the system into components and looked deeper into how each part could fail.

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?
   One major risk is data loss or corruption, which can harm user trust and make the system unreliable. Another risk is accessibility failure, where users with limitations might not be able to use the software as intended. Both are important to consider because they affect user confidence, usability, and the overall success of the product.

8.2 Gourob Podder Reflection

1. What went well while writing this deliverable?
   Identifying system boundaries and major components went smoothly because the project architecture was already clear from prior SRS work. Defining modules like the User Interface, Natural Language Understanding, and Agent Manager helped organize potential hazards systematically. Team collaboration also allowed us to quickly align on how risks could manifest in both software and user interactions.

2. What pain points did you experience during this deliverable, and how did you resolve them?
   Determining the appropriate level of detail for the hazard analysis without drifting into system design was challenging. It was sometimes difficult to distinguish between a hazard and a general usability concern. We resolved this by using safety engineering definitions and focusing on sources of harm or adverse outcomes rather than design flaws. Abstract risks such as privacy or misinterpretation of user intent were addressed using scenario-based brainstorming.

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, how did they come about?
   We had already recognized usability and privacy risks, like misinterpreted commands or unauthorized data access. New risks emerged during the hazard analysis, especially related to component interactions, such as potential failures in the Task Planning module or dependency on unreliable external APIs. These became apparent as we examined each subsystem and its interface boundaries.

4. Other than the risk of physical harm, list at least 2 other types of risk in software products. Why are they important to consider?
   Privacy and data security risks, such as improper handling of sensitive user data, can harm user trust and violate legal requirements. Reliability and availability risks, like system downtime or malfunctions, can prevent users from completing essential tasks, especially for assistive technologies. Both are critical for maintaining usability, accessibility, and overall user confidence.

8.3 Savinay Chhabra Reflection

1. What went well while writing this deliverable?
   The provided template helped structure our hazards. Collaboration and work distribution went more smoothly, and we could relate hazards directly to the well-structured SRS document.

2. What pain points did you experience during this deliverable, and how did you resolve them?
   Staying solution-neutral while analyzing hazards was difficult. The line between requirements and design is blurred, and we sometimes discussed solutions prematurely. We resolved this by refocusing on hazards rather than proposed solutions.

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, how did they come about?
   Initially, we focused on privacy and legal risks associated with data breaches. While performing hazard analysis, we discovered additional potential risks and mitigation strategies that were not previously considered.

4. Other than the risk of physical harm, list at least 2 other types of risk in software products. Why are they important to consider?
   Legal risks due to data breaches or violations of user privacy, and compliance risks from accessibility failures in design. Both are important because they affect user trust, legal standing, and the usability of the product.

8.4 Ajay Singh Grewal Reflection

1. What went well while writing this deliverable?
   Turning the hazards into a bunch of short, testable requirements worked well. By adding criteria under each requirement, it made it easier to understand how to verify them later. The FMEA table also helped organize the hazards clearly, and breaking the system into components made it easier to think about specific risks.

2. What pain points did you experience during this deliverable, and how did you resolve them?
   Choosing the criteria that was realistically achievable within our time frame was tricky. We had to balance between being thorough and being practical. We resolved this by focusing on the highest-impact hazards that were most relevant to our project scope and user needs.

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, how did they come about?
   Had already discussed risks like confirmation for destructive actions and permission issues. While writing the hazard analysis, we identified additional risks like OS compatibility and deployment-related failures.

4. Other than the risk of physical harm, list at least 2 other types of risk in software products. Why are they important to consider?
   Firstly, privacy risk associated with sending more data than needed, or sending it without user consent. This breaks user trust and can lead to future disuse. Another less discussed risk is a reliability risk, where if a bad update happens, a specific tools breaks, or if a bug gets introduced, it can lead to the tool being unusable. This can lead to user dissatisfaction and a loss of trust.