Module Interface Specification for Software Engineering

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

November 6, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Software Engineering.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of Software Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Software Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The **Proxi**, is decomposed into a hierarchy of modules following the design principles of information hiding and separation of concerns. Each module corresponds to a well-defined secret and is independently assignable to a developer. The decomposition balances hardware hiding, behaviour hiding, and software decision modules.

Level 1	Level 2 Modules (Secrets / Responsibilities)
Hardware-Hiding	**HH-IO (Audio Adapter)** – manages microphone input and audio output across different platforms. **HH-Auto (System Control)** – performs basic desktop actions such as typing, clicking, or launching applications.
Behaviour-Hiding	**BH-Input (Voice & Text Manager)** – captures user input, converts speech to text, and normalizes text commands. Implements *FUNC.R.1–R.2*. **BH-NLU (Intent Parser)** – interprets text into structured intents and parameters based on defined command patterns. Implements *FUNC.R.3*. **BH-Plan (Task Executor)** – determines which agent or tool should handle a command and coordinates its execution. Implements *FUNC.R.4*. **BH-Safety (Confirmation Gate)** – validates actions that may affect files or system settings and requests confirmation. Implements *FUNC.R.9*. **BH-Session (Context Manager)** – maintains user session data, history, and undo information for continuity. Supports *FUNC.R.4*. **BH-Feedback (Response Manager)** – converts textual responses into spoken or visual feedback for the user. Implements *FUNC.R.5–R.6*. **BH-UI (Proxi Interface)** – presents status updates, confirmations, and results; supports full voice-only interaction. Implements *FUNC.R.8*.
Software-Decision	**SD-Types (Core Structures)** – defines abstract data types for Command, Intent, and Plan. **SD-ToolRegistry (Action Map)** – maintains the mapping between recognized intents and available system actions. **SD-Store (Local Storage)** – handles persistent storage for user preferences, session history, and logs. **SD-STT/TTS Config** – specifies configuration for speech and text synthesis models and supported languages. **SD-Log (Event Logger)** – records system actions and feedback events for debugging and validation.

Table 1: Module Hierarchy for Proxi Voice Assistant

Likely Changes:

- The choice of speech recognition or text-to-speech library (for example, switching from Whisper API to a local model).
- Adjustments to the user interface layout or how voice commands trigger visible feedback or audio playback.
- Fine-tuning thresholds for speech detection and timing between input and response based on user testing.
- Updating supported voice commands or adding new MCP tools as features are expanded.

Unlikely Changes:

- The main processing loop of Input $\rightarrow$ Interpret $\rightarrow$ Plan $\rightarrow$ Execute $\rightarrow$ Feedback.
- The core data structures used for storing Commands, Intents, and Action Plans.
- The communication pattern between modules through the MCP agent interface.

Traceability to SRS:

- **BH-Input** fulfills *FUNC.R.1–R.2*: speech and text input handling with accuracy $\geq 90\%$.
- **BH-NLU** fulfills *FUNC.R.3*: intent recognition accuracy $\geq 90\%$.
- **BH-Plan** fulfills *FUNC.R.4*: agent planning and execution success rate $\geq 85\%$.
- **BH-Feedback** fulfills *FUNC.R.5–R.6*: provides feedback and spoken confirmation within response time $\leq 2\,\mathrm{s}$.
- **BH-UI** fulfills *FUNC.R.8*: supports full hands-free operation for accessibility.
- **BH-Safety** fulfills *FUNC.R.9*: requests confirmation before executing high-risk or destructive actions.
- **Support modules (SD, HH)** enable non-functional goals on latency, reliability, and auditability through structured logging.

6 MIS of HH-IO (Audio Adapter)

6.1 Module

HH-IO (Audio Adapter)

6.2 Uses

System audio interface

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	**Input**	**Output**	**Errors**
openMic	N/A	N/A	MicNotFound
closeMic	N/A	N/A	CloseFailed
recordAudio	seconds: real	sound: AudioStream	RecordFailed
playAudio	sound: AudioStream	N/A	PlaybackFailed

6.4 Semantics

6.4.1 State Variables

- micOpen : Boolean
- lastSignal : AudioStream

6.4.2 Environment Variables

- micDevice : physical microphone
- speakerDevice : physical speaker or headset

6.4.3 Assumptions

- At least one working microphone and speaker device exists.
- Only one module controls the microphone at a time.

6.4.4 Access Routine Semantics

openMic():

- transition: micOpen := true if micDevice is available.
- exception: MicNotFound if micDevice is missing or busy.

closeMic():

- transition: micOpen := false if open.
- exception: CloseFailed if device cannot close.

recordAudio(seconds):

- output: returns AudioStream of given duration.
- transition: lastSignal := captured audio.
- exception: RecordFailed if capture fails or micOpen = false.

playAudio(sound):

- transition: lastSignal := sound.
- output: sound played through speakerDevice.
- exception: PlaybackFailed if playback fails.

6.4.5 Local Functions

None.

7 MIS of HH-Auto (System Control)

7.1 Module

HH-Auto (System Control)

7.2 Uses

Operating system automation interface

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Access Programs

Name	Input	Output	Errors
moveCursor	p: ScreenPos	N/A	ActionError
leftClick	N/A	N/A	ActionError
typeText	t: String	N/A	ActionError
openApp	id: AppId	N/A	ActionError

7.4 Semantics

7.4.1 State Variables

- currentPos : ScreenPos

7.4.2 Environment Variables

- desktopEnv : user desktop environment
- keyboardDevice : keyboard input channel
- pointingDevice : mouse or trackpad

7.4.3 Assumptions

- The user session allows simulated input events.
- Screen coordinates are valid for the active display.
- AppId refers to an installed and accessible application.

7.4.4 Access Routine Semantics

moveCursor(p):

- transition: currentPos := p.
- output: N/A.
- exception: ActionError if cursor move fails.

leftClick():

- transition: none.
- output: N/A.
- exception: ActionError if click event fails.

typeText(t):

- transition: none.
- output: N/A.
- exception: ActionError if key input fails.

openApp(id):

- transition: none.
- output: N/A.
- exception: ActionError if app launch fails.

7.4.5 Local Functions

None.

8 MIS of BH-Input (Voice & Text Manager)

8.1 Module

BH-Input (Voice & Text Manager)

8.2 Uses

HH-IO, SD-STT/TTS Config, SD-Types, SD-Log

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	Input	Output	Errors
startCapture	mode: InputMode	N/A	MicUnavailable, AlreadyCapturing
stopCapture	N/A	N/A	NotCapturing
getLastText	N/A	text: String	NoInputAvailable
getStatus	N/A	s: InputStatus	N/A

8.4 Semantics

8.4.1 State Variables

- inputState : InputState
- currentMode : InputMode
- lastText : String
- partialText : String
- lastError : InputError or null

InputState = {Idle, Listening, Processing}
InputMode = {VoiceOnly, TextOnly, Mixed}
InputStatus is a record:

- state : InputState
- hasText : Boolean
- hasError : Boolean

8.4.2 Environment Variables

- micStream : handled by HH-IO for live audio
- sttService : speech-to-text service
- now : system clock for timing

8.4.3 Assumptions

- The microphone and STT component are available when started.
- Only one capture session runs at a time.
- Calling modules handle all exceptions raised.

8.4.4 Access Routine Semantics

startCapture(mode):

- transition: if inputState = Idle and micStream ready then

$$inputState := Listening, \quad currentMode := mode, \quad partialText := ""$$

- exception: MicUnavailable if device fails, AlreadyCapturing if inputState $\neq$ Idle.

stopCapture():

- transition: if inputState $\neq$ Idle then inputState := Idle.
- exception: NotCapturing if inputState = Idle.

getLastText():

- output: returns lastText if not empty.
- exception: NoInputAvailable if lastText is empty.

getStatus():

- output: returns a record s with $s.state = inputState$, $s.hasText = (lastText \neq "")$, $s.hasError = (lastError \neq null)$.

8.4.5 Local Functions

Let Event = {StartCmd, StopCmd, Chunk, Error, Timeout}
$nextState : InputState \times Event \rightarrow InputState$

$$nextState(s,e) = \begin{cases} Listening & \text{if } s = Idle \wedge e = StartCmd \\ Idle & \text{if } s = Listening \wedge e = StopCmd \\ Processing & \text{if } s = Listening \wedge e = Chunk \\ Processing & \text{if } s = Processing \wedge e = Chunk \\ Idle & \text{if } e = Error \vee e = Timeout \\ s & \text{otherwise} \end{cases}$$

During execution BH-Input updates

$$inputState := nextState(inputState, e)$$

for each event e. When transcription ends, partialText moves into lastText.

9 MIS of BH-NLU (Intent Parser)

9.1 Module

BH-NLU (Intent Parser)

9.2 Uses

SD-Types, SD-Log

9.3 Syntax

9.3.1 Exported Access Programs

Name	Input	Output	Errors
parseText	text: String	i: Intent	ParseError

9.4 Semantics

9.4.1 State Variables

None.

9.4.2 Environment Variables

None.

9.4.3 Assumptions

- Input text is in English and grammatically valid.
- Command patterns are defined in SD-Types.

9.4.4 Access Routine Semantics

parseText(text):

- output: produces an Intent record with fields

$$i.type = detectCommand(text), \quad i.params = extractParams(text)$$

- exception: ParseError if text cannot be matched to any pattern.

9.4.5 Local Functions

$$detectCommand : String \rightarrow IntentType$$

$$extractParams : String \rightarrow ParamSet$$

10 MIS of BH-Plan (Task Executor)

10.1 Module

BH-Plan (Task Executor)

10.2 Uses

BH-NLU, SD-ToolRegistry, SD-Types, SD-Log, HH-Auto

10.3 Syntax

10.3.1 Exported Constants

ExecStatus = {Pending, Success, Failed}

10.3.2 Exported Access Programs

Name	Input	Output	Errors
planAction	i: Intent	p: Plan	NoToolFound
executePlan	p: Plan	s: ExecStatus	ExecError
cancelPlan	N/A	N/A	NoPendingPlan
getLastStatus	N/A	s: ExecStatus	N/A

10.4 Semantics

10.4.1 State Variables

- currentPlan : Plan or null
- lastStatus : ExecStatus

10.4.2 Environment Variables

- toolSet : accessible system tools or MCP agents
- now : system clock for execution timing

10.4.3 Assumptions

- The input intent has been validated by BH-NLU.
- Each available tool in SD-ToolRegistry exposes a run() routine.
- MCP agents or automation tools are reachable when requested.

10.4.4 Access Routine Semantics

planAction(i):

- output: generates a Plan record p with:

$$p.tool = matchTool(i.type), \quad p.parameters = i.params, \quad p.time = now$$

- transition: currentPlan := p.
- exception: NoToolFound if matchTool fails.

executePlan(p):

- transition:
  - currentPlan := p.
  - s := runTool(p.tool, p.parameters).
- output: returns s of type ExecStatus.
- exception: ExecError if runTool fails.

cancelPlan():

- transition: currentPlan := null, lastStatus := Failed.
- exception: NoPendingPlan if currentPlan = null.

getLastStatus():

- output: returns lastStatus.
- transition: none.

10.4.5 Planning Logic

To model the planning stage, define:

$$matchTool : IntentType \to ToolId$$

$$runTool : ToolId \times ParamSet \to ExecStatus$$

The planning decision can be expressed as:

$$planAction(i) = \begin{cases} p = (matchTool(i.type), i.params, now) & \text{if a tool exists for } i.type \\ \text{NoToolFound error} & \text{otherwise} \end{cases}$$

Execution behaviour follows:

$$executePlan(p) = \begin{cases} Success & \text{if runTool(p.tool, p.parameters) = true} \\ Failed & \text{otherwise} \end{cases}$$

10.4.6 Local Functions

- **matchTool**(t): searches SD-ToolRegistry for a matching tool.
- **runTool**(id, params): calls the linked MCP or system command.

11 MIS of BH-Safety (Confirmation Gate)

11.1 Module

BH-Safety (Confirmation Gate)

11.2 Uses

BH-UI, SD-Types, SD-Log

11.3 Syntax

11.3.1 Exported Constants

RiskLevel = Low, Medium, High
SafetyDecision = AutoAllow, AskUser, Block
ApprovalResult = Approved, Denied, Cancelled

11.3.2 Exported Access Programs

Name	Input	Output	Errors
classifyAction	a: Action	r: RiskLevel	N/A
decidePolicy	a: Action	d: SafetyDecision	N/A
confirmAction	a: Action	res: ApprovalResult	UserTimeout

11.4 Semantics

11.4.1 State Variables

- pendingAction : Action or null
- lastDecision : SafetyDecision or null

11.4.2 Environment Variables

- uiChannel : connection to BH-UI for user prompts
- now : system clock for time limits

11.4.3 Assumptions

- BH-UI can show a yes/no prompt and return a user response.
- Every Action record includes a defined riskLevel field.
- The system clock is monotonic for timeout checks.

11.4.4 Access Routine Semantics

classifyAction(a):

- output: returns a.riskLevel.
- transition: none.

decidePolicy(a):

- output: returns d of type SafetyDecision, where

$$d = \begin{cases} AutoAllow & \text{if } a.riskLevel = Low \\ AskUser & \text{if } a.riskLevel = Medium \\ Block & \text{if } a.riskLevel = High \land a.isIrreversible \\ AskUser & \text{if } a.riskLevel = High \land \neg a.isIrreversible \end{cases}$$

- transition: lastDecision := d.

confirmAction(a):

- transition: pendingAction := a.
- output:
  - If decidePolicy(a)=AutoAllow then res=Approved.
  - If decidePolicy(a)=Block then res=Denied.
  - If AskUser then BH-UI prompts user; waits for yes/no.
- exception: UserTimeout if no answer before time limit.

11.4.5 Risk Policy Table

Risk	Irreversible?	Decision	Example
Low	N/A	AutoAllow	Open folder, read file
Medium	N/A	AskUser	Rename or move file
High	false	AskUser	Delete to recycle bin
High	true	Block	Permanently delete data

11.4.6 Local Functions

$$policy : RiskLevel \times Bool \rightarrow SafetyDecision$$

$$policy(r, irr) = \begin{cases} AutoAllow & \text{if } r = Low \\ AskUser & \text{if } r = Medium \\ Block & \text{if } r = High \wedge irr = true \\ AskUser & \text{if } r = High \wedge irr = false \end{cases}$$

decidePolicy(a) = policy(a.riskLevel, a.isIrreversible)
This module fulfills *FUNC.R.9* by ensuring confirmation or blocking of high-risk actions, reducing hazards identified in the safety analysis.

12 MIS of BH-Feedback (Response Manager)

12.1 Module

BH-Feedback (Response Manager)

12.2 Uses

HH-IO, SD-STT/TTS Config, SD-Types, SD-Log

12.3 Syntax

12.3.1 Exported Constants

OutputMode = {VoiceOnly, TextOnly, Both}
FeedbackStatus = {Idle, Speaking, Completed, Failed}

12.3.2 Exported Access Programs

Name	Input	Output	Errors
queueMessage	msg: String, m: OutputMode	N/A	QueueFull
speakNow	msg: String, m: OutputMode	s: FeedbackStatus	TtsError
getLastStatus	N/A	s: FeedbackStatus	N/A
cancelAll	N/A	N/A	N/A

12.4 Semantics

12.4.1 State Variables

- outputQueue : sequence of (String, OutputMode)
- lastStatus : FeedbackStatus
- isSpeaking : Boolean

12.4.2 Environment Variables

- audioOut : speaker connection through HH-IO
- ttsService : text-to-speech component

12.4.3 Assumptions

- ttsService can turn any short message into speech in less than the required response time from the SRS.
- HH-IO can play audio without blocking the whole system.
- The output queue has a fixed maximum size.

12.4.4 Access Routine Semantics

queueMessage(msg, m):

- transition:
  - If the queue is not full then append (msg, m) to outputQueue.
- output: N/A.
- exception: QueueFull if appending would exceed the limit.

speakNow(msg, m):

- transition:
  - isSpeaking := true; lastStatus := Speaking.
  - If m = VoiceOnly or m = Both then send msg to ttsService and play through audioOut.
  - If m = TextOnly or m = Both then write msg to the log or UI channel.
- output:
  - If playback and any text output succeed then lastStatus := Completed and s = Completed.
  - Otherwise lastStatus := Failed and s = Failed.
- exception: TtsError if ttsService cannot produce speech.

getLastStatus():

- output: returns lastStatus.
- transition: none.

cancelAll():

- transition: clear outputQueue; isSpeaking := false; lastStatus := Idle.
- output: N/A.
- exception: N/A.

12.4.5 Local Functions

We model the processing of the queue with a helper function:

$$nextMessage : \text{sequence of (String, OutputMode)} \rightarrow (String, OutputMode) \cup \{\text{None}\}$$

$$nextMessage(q) = \begin{cases} \text{first element of } q & \text{if } q \neq [] \\ \text{None} & \text{if } q = [] \end{cases}$$

BH-Feedback periodically checks outputQueue. If nextMessage returns a pair (msg, m), it behaves as in speakNow(msg, m) and then removes that entry from the queue. If None, it leaves the state unchanged.
This module fulfills *FUNC.R.5–R.6* by providing timely spoken and visual feedback to the user, and by reporting a clear status that can be logged or shown in the interface.

13 MIS of BH-UI (Proxi Interface)

13.1 Module

BH-UI (Proxi Interface)

13.2 Uses

BH-Feedback, BH-Safety, BH-Input, SD-Log, SD-Types

13.3 Syntax

13.3.1 Exported Constants

UIState = {Idle, Listening, Waiting, Displaying, Error}

13.3.2 Exported Access Programs

Name	Input	Output	Errors
updateView	s: UIState	N/A	RenderError
showMessage	msg: String	N/A	RenderError
promptUser	q: String	ans: Bool	Timeout
showStatus	st: FeedbackStatus	N/A	N/A
clearScreen	N/A	N/A	N/A

13.4 Semantics

13.4.1 State Variables

- uiState : UIState
- lastMsg : String
- lastError : String or null

13.4.2 Environment Variables

- display : visual interface (screen or console)
- micLED : visual cue showing listening status

13.4.3 Assumptions

- Display device is available and writable.
- Voice cues or LEDs can toggle quickly without delay.
- Text is short enough to fit within screen limits.

13.4.4 Access Routine Semantics

updateView(s):

- transition: uiState := s.
- output: updates micLED or text based on state.
- exception: RenderError if update fails.

showMessage(msg):

- transition: lastMsg := msg; uiState := Displaying.
- output: renders msg visually or as voice through BH-Feedback.
- exception: RenderError if display or output fails.

promptUser(q):

- transition: uiState := Waiting.
- output: shows q; waits for yes/no reply by voice or keypress.
- exception: Timeout if no input received in time limit.

showStatus(st):

- output: shows latest FeedbackStatus or progress.
- transition: none.

clearScreen():

- transition: uiState := Idle; lastMsg := ””.
- output: clears visual area.

13.4.5 Local Functions

We define a helper that maps states to display text:

$$stateText : UIState \rightarrow String$$

$$stateText(s) = \begin{cases} ''Listening...'' & \text{if } s = Listening \\ ''Waiting for input...'' & \text{if } s = Waiting \\ ''Processing...'' & \text{if } s = Displaying \\ ''Idle'' & \text{if } s = Idle \\ ''Error'' & \text{if } s = Error \end{cases}$$

This mapping helps BH-Feedback and BH-Safety show consistent notifications through both visual and voice channels. This module fulfills *FUNC.R.8* by ensuring full hands-free interaction and clear accessibility feedback for all system states.

14 MIS of [Module Name —SS]

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

14.1 Module

[Short name for the module —SS]

14.2 Uses

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

14.4 Semantics

14.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

14.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

14.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

14.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

14.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

15 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

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 design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?
4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?
5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)
6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)

Amanbeer Minhas Reflection

1. **What went well while writing this deliverable?**

   Once I had the SRS, VnV Plan, and Hazard Analysis in place, this design doc felt much more manageable. Breaking the system into BH-Input, BH-Plan, BH-Safety, BH-Feedback, and BH-UI made it easier to see how everything fit together. Courses like COMPSCI/SFWRENG 3RA3 (Requirements) helped me think in terms of clear responsibilities and traceable requirements.My QA co-op experience also helped because I naturally thought about how each module would be tested while I was specifying it.

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

   The main pain point was finding the right level of abstraction. At first, I wrote the MIS in terms of specific tools and APIs, which made the design feel tied to one implementation. After revisiting the lecture notes and examples, I rewrote the modules to focus on behaviour and information hiding instead. Another challenge was formal notation for state and transitions. I drew on ideas from SFWRENG 2DM3 (Discret Maths) to think in terms of states, events, and transitions, then simplified that down to the most important pieces so the document would still be readable.

3. **Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. peers, stakeholders, users)?**

   Most of our client feedback came from our team mates parents/relatives trying early voice-based prototypes. Some people wanted fewer confirmations so the system felt faster, while others were worried about accidental destructive actions. This made us directly include the BH-Safety module: we introduced risk levels and different behaviours for low, medium, and high risk actions. Ideas around accessibility and clear feedback were reinforced by discussions in ENGINEER 4A03, where we talked about inclusivity, ethics, and duty of care. Where we did not have direct user input, I leaned on course examples and the capstone lectures for reference.

4. **While creating the design doc, what parts of your other documents (e.g., requirements, hazard analysis, etc), if any, needed to be changed, and why?**

   While writing the MIS, I noticed that some requirements in the SRS were too vague to map cleanly to modules. We will refine a few of them to have clearer success conditions (for example, accuracy targets and timing bounds), so they matched the behaviour of BH-Input and BH-Feedback. The Hazard Analysis also needs change: some risks originally assigned to "the system" were moved specifically to BH-Safety, since that is where confirmation and blocking actually happen. The VnV Plan needs change to add module-level tests for voice accuracy, plan execution success, and safety prompts, reflecting what I learned about test design in my QA co-op and SFWRENG 3S03(Software testing).

5. **What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)**

   Right now, the design assumes relatively simple commands and a limited amount of context. The NLU is closer to pattern matching than full natural language understanding. The system also depends on reliable speech models and may not perform as well in noisy environments or for all accents. With more time and resources, we would explore more robust language models, better noise handling, and wider accessibility testing with real users. We would also add more automation around logging and replaying user sessions for regression testing, drawing from ideas in my QA work and SFWRENG 3S03 and STATS 3Y03 for analyzing failure patterns.

6. **Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)**

   We considered three main designs. The first was a single monolithic voice assistant where input, planning, and safety were all handled in one loop. This would have been easier to code quickly but very hard to test or change without breaking things. The second option relied mostly on an external cloud assistant, which might give better speech accuracy but raises privacy and reliability concerns. The third, which we chose, is the modular design in this document, with separate BH modules and clear interfaces. It fits the ideas from SFWRENG 2OP3, 2C03, 3RA3, and 3XB3 about modularity, testing, and requirements traceability. We chose it because it balances what we can realistically implement as a student with the level of structure and safety we have learned to aim for in software engineering.