Video Link:
https://www.youtube.com/watch?v=13C9nkQ8lVg

VOID IDE

ENHANCEMENT PROPOSAL

Presented By: Emily Cheng, Jeff Hong

GROUP MEMBERS

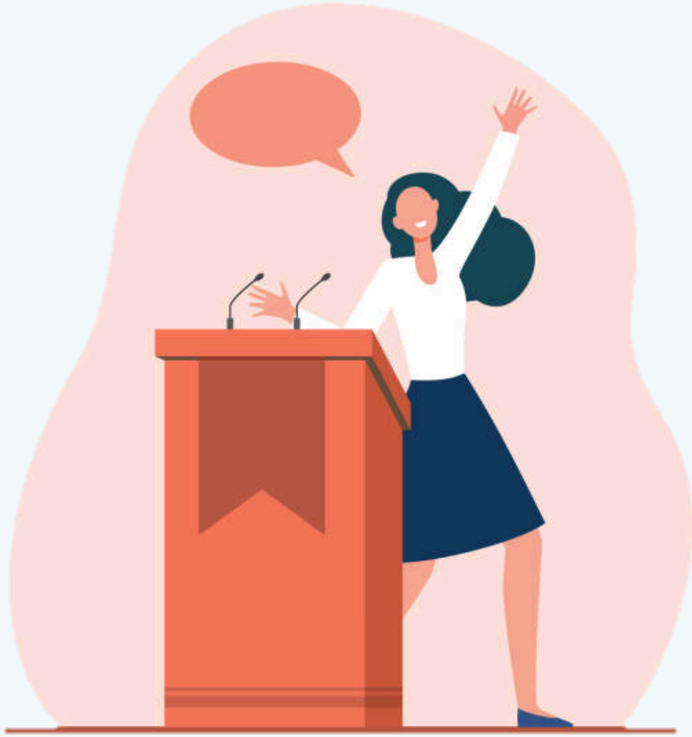
- Jinpeng Deng 22SS117@queensu.ca – (Group Leader) Introduction and Abstract, High-Level Architecture Changes, Risks and Benefits Overview.
- Jeff Hong 22RQ20@queensu.ca – (Presenter) Grammar Review, Static Design, SAAM, Stakeholders, NFRs, Impact Analysis, slides.
- Emily Cheng 22TWS1@queensu.ca – (Presenter) Use Cases, Sequence Diagrams, Box-and-Arrow diagrams, Directories, editing, slides.
- Zijie Gan 21ZG6@queensu.ca – Conclusion, AI report, Lessons Learnt.
- Nathan Daneliak 22TMX2@queensu.ca – Understand, Architecture Effects, NFR Analysis, Risk Categories.
- Zipeng Chen 23QH10@queensu.ca – Real-time Design, SAAM, Stakeholders, NFRs, Impact Analysis.

01

MOTIVATION OF FEATURE

SALES PITCH

- The enhancement lets users talk to a LLM, receive responses, perform quick edits, and use IDE commands with only their voice
- VoiceUI component acts as the interface, SpeechService handles STT and TTS, VoiceSessionController manages voice commands
- Mimics the idea of rubber duck programming to inspire better problem solving
- Returns a vocalized response to provide more ideas instead of simply talking to yourself
- Improves productivity overall, letting users perform tasks in an instant with IDE voice commands, and voice activated quick code edits



02

SAAM ANALYSIS

ALT 1: STATIC SPEECH PROCESSING

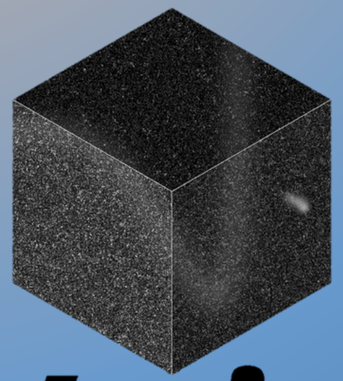
Scenario	Main Impact of Static Processing	Key NFRs Related
Voice Chat with the chosen LLM	• Waits for user to finish speaking, less resources used	Usability, performance.
Voice Quick Edit	• May take longer for edits to take place due to static processing	Usability, reliability.
Network Delays while streaming STT	• Static processing is less resource intensive, thus less affected by latency	Reliability, UX, complexity.
Complex real-time Voice Commands in the IDE.	• Due to shorter voice commands, static processing does not slow down the process considerably	Usability, complexity.

03

SAAM ANALYSIS

ALT 2: REAL-TIME SPEECH PROCESSING

Scenario	Main Impact of Static Processing	Key NFRs Related
Voice Chat with the chosen LLM	• More resource intensive but process concurrently while speaking	Usability, performance.
Voice Quick Edit	• Having text be concurrently processed means the quick edit sequence can start sooner	Usability, reliability.
Network Delays while streaming STT	• Due to higher resource requirements, latency becomes more apparent	Reliability, UX, complexity.
Complex real-time Voice Commands in the IDE.	• The voice driven actions can be performed slightly quicker due to processing finishing quicker	Usability, complexity.



Void

04

STAKEHOLDERS + NFRS OF EACH

USERS

The ones who decide whether Void succeeds by choosing to install it. They care about portability and performance.

CORE DEVELOPERS AND CONTRIBUTORS

Primarily care about the actual code of Void. Need modifiability and maintainability.

AI PROVIDERS

They want Void to be successful so their LLMs can be more popular. So they care about the same NFRs as the users.

PLATFORM STAKEHOLDERS

They want Void itself to be successful. So they care about the same NFRs as the users.

ADVANTAGES/DISADVANTAGES

THE BEST ALTERNATIVE IS STATIC PROCESSING

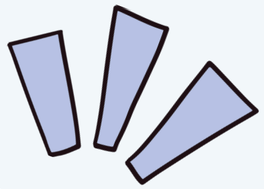
05

Static Processing:

Advantages:
- Simpler implementation.
- Uses less resources.
- More reliable.

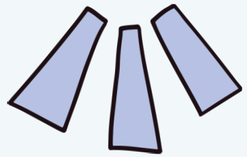
Disadvantages:
- Slower response time.
- User needs to wait for text to finish processing.

Overall:
Slower response speed in exchange for better performance and integration.



Static Processing

- Satisfies the user desire for performance.
- Satisfies developer desire for modifiability and maintainability.
- Is overall more consistent.



Real-Time Processing:

Advantages:
- Faster response time.
- Better overall flow with functions within the IDE.

Disadvantages:
- More affected by latency.
- Complicated implementation.
- Increased processing power.

Overall:
Faster response speed in exchange for being more resource intensive and complex integration.

06 CONCURRENCY & TEAM ISSUES ON ARCHITECTURE

Design and Impact on Void's Development

Void uses a multi-process architecture.

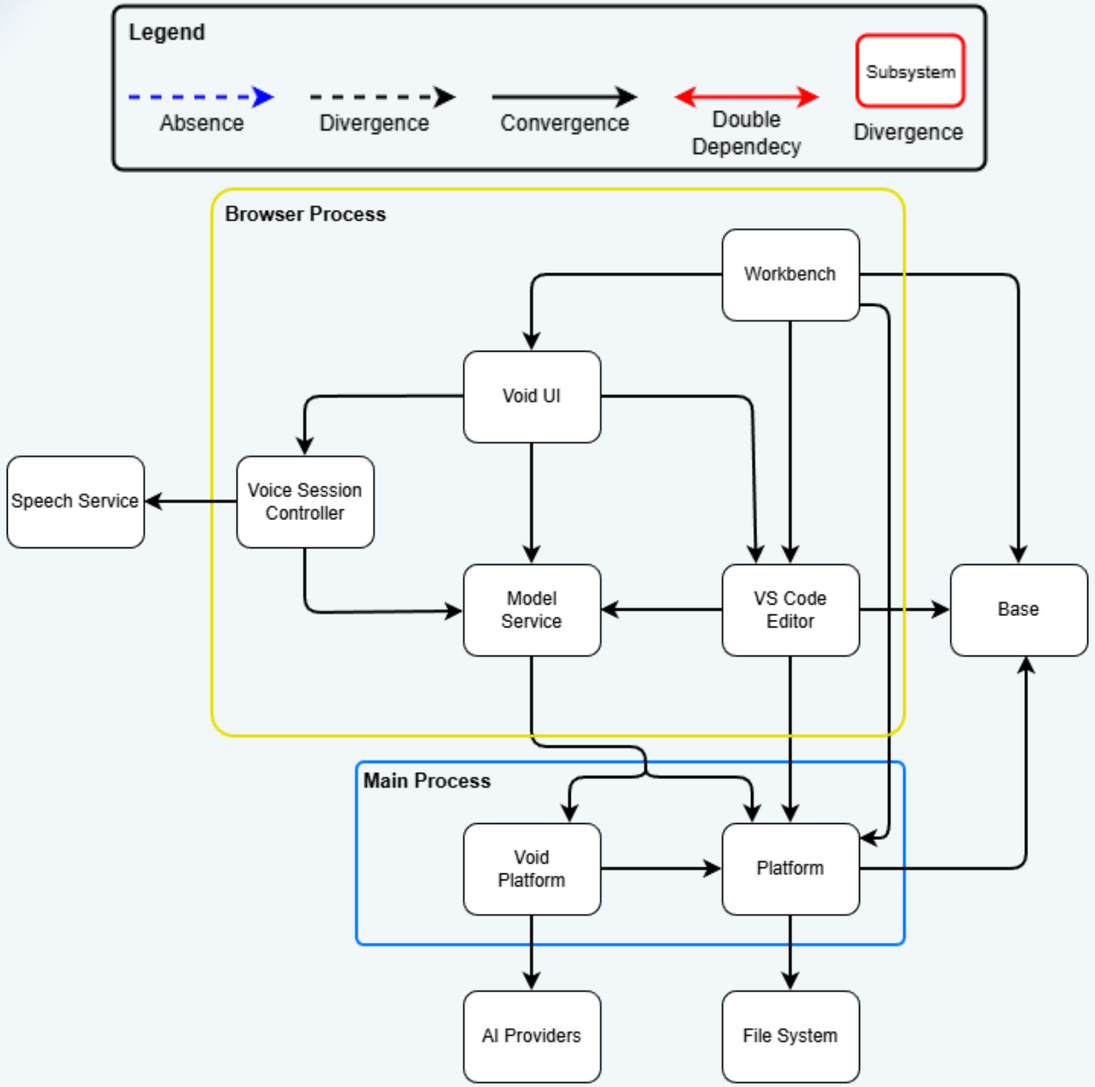
- Communication is asynchronous IPC => all message handling is non-blocking.
- VoiceSessionController and SpeechService must operate without blocking the UI or LLM pipeline.
- Teams split across UI, Extension Host, Platform due to process separation.
- Asynchronous IPC requires coordination on shared message schemas and interface definitions.
- Cross-process features need careful planning to avoid breaking interactions.

Impact on Our A3 Feature and Team Workflow

- Parallel work (use cases, SAAM, subsystems, AI report) need shared terminology.
- Diagrams and subsystems descriptions had to align with the same message pipeline.
- This coordination shaped a design that isolates voice processing and maintains clear dataflow.

07

IMPACTED SUBSYSTEMS



Clear separation of responsibilities is preserved. Changes are isolated to UI and Service layer.

Voice-Processing Above Existing Message Pipeline:
New layer with new components VoiceInputUI, SpeechService, and VoiceSessionController.

Affected Subsystems:
Void UI (chat) and ChatThreadService. Ensures subsystem stability by avoiding deep architectural modifications.

Core subsystems remain unchanged:
Some examples are LLMMessageService, Platform IPC, QuickEdit. Message-processing architecture stays intact.

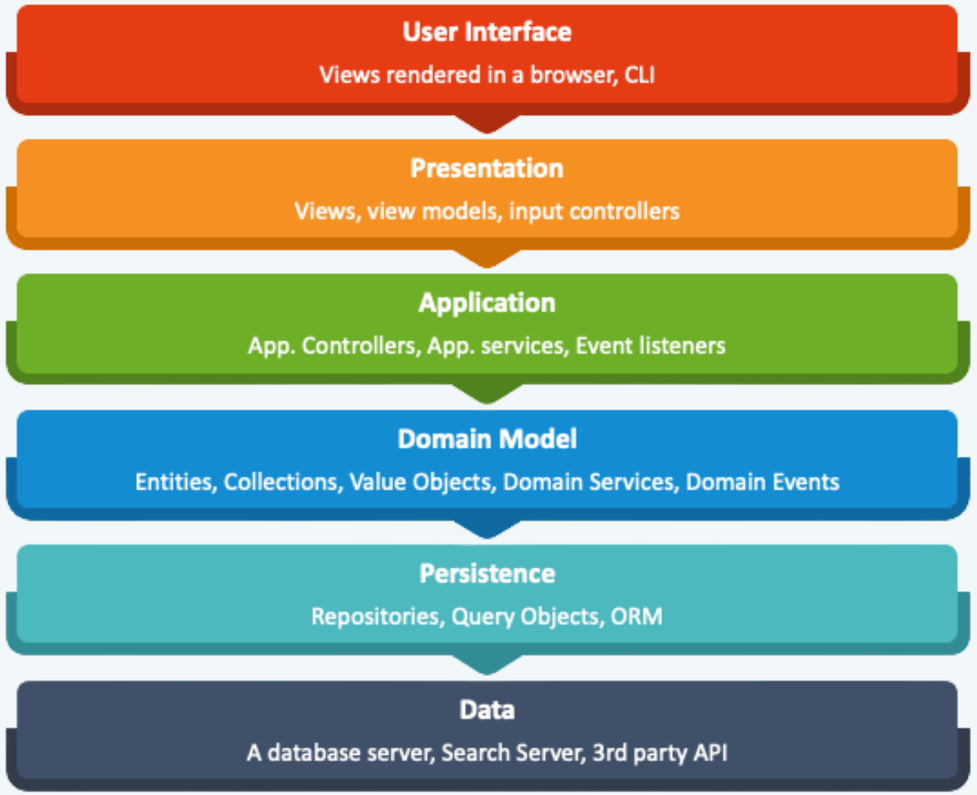
Enhancement fits naturally into existing flow without coupling voice handling to the LLM pipeline. Architectural “kernel” remains unchanged even after adding voice input feature.

01 **Layered Style:** UI forms the outer layer while message handling and IPC form the core.

02 **Microkernel Influence:** Core LLM pipeline stays stable, and the voice layer is added as an outer module.

03 **Pipes-and-Filters Flow:** Audio is transformed step by step (STT → formatted text → LLM), with SpeechService acting as a new filter.

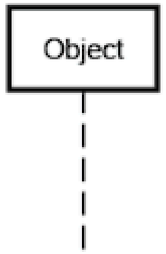
04 **Separation of Concerns:** Responsibilities remain clearly divided, preventing coupling between voice handling and core message processing.



IMPACTED ARCHITECTURE STYLES

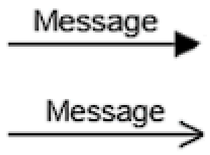
09 SEQUENCE DIAGRAM LEGEND

Actor: The user (developer) who interacts with the Void IDE system and thus its objects, through the interface.



Object: Represents a component or class of Void's concrete architecture.

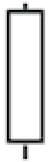
Lifeline: Vertical dashed line from each box, represents existence and activity over the passage of time as it goes downwards.



Message: Interaction or communication between objects, represents a method call, signal, data exchange between components.

- **Synchronous:** Normal method call where the sender waits for a response.
- **Asynchronous:** For background tasks or events, the sender doesn't wait.

Reply: A return message, response sent from the receiver of a message when it has finished processing.

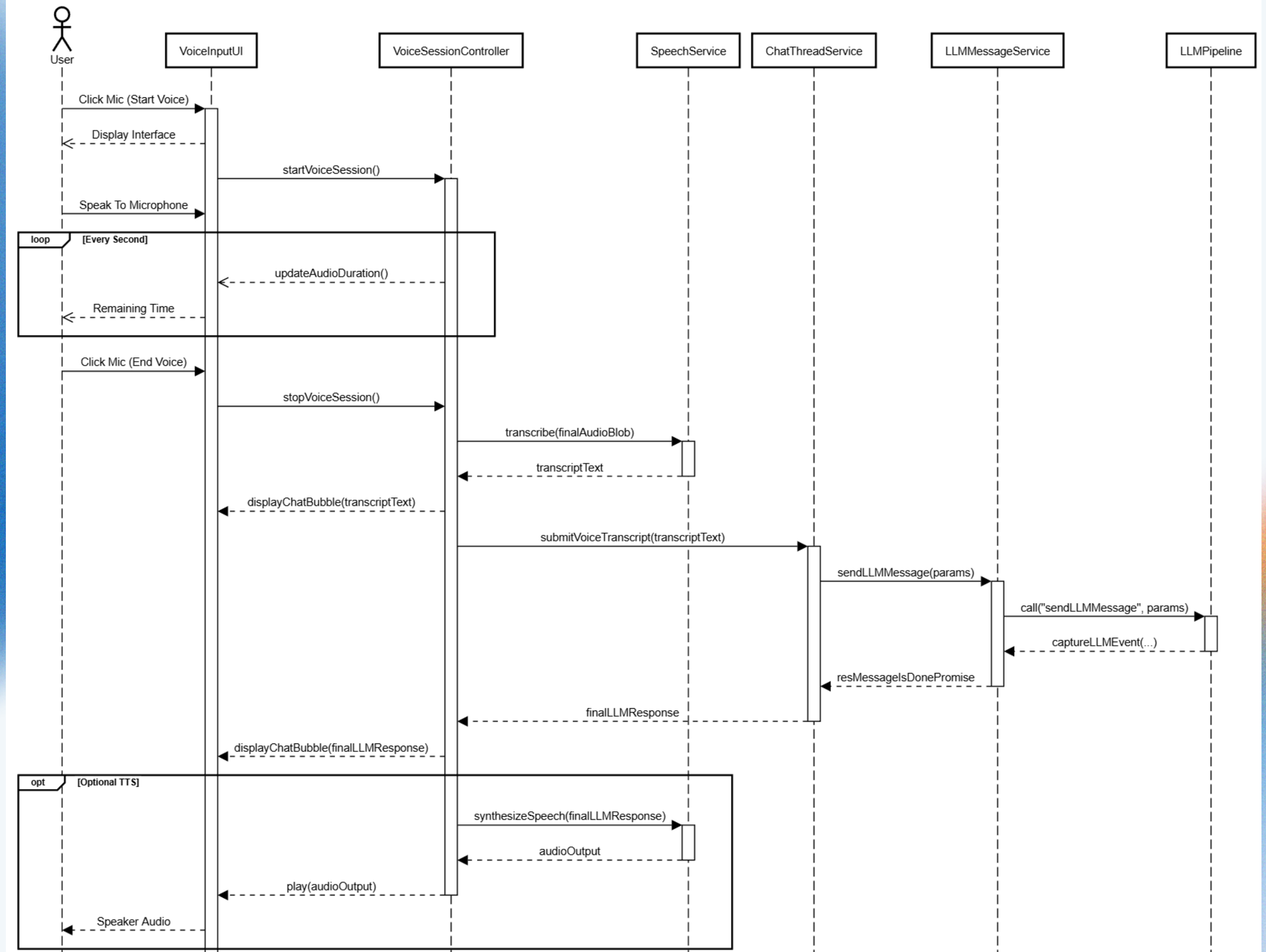


Activation Box: Rectangular box along an object's timeline that represents the time it takes for a task to be completed.

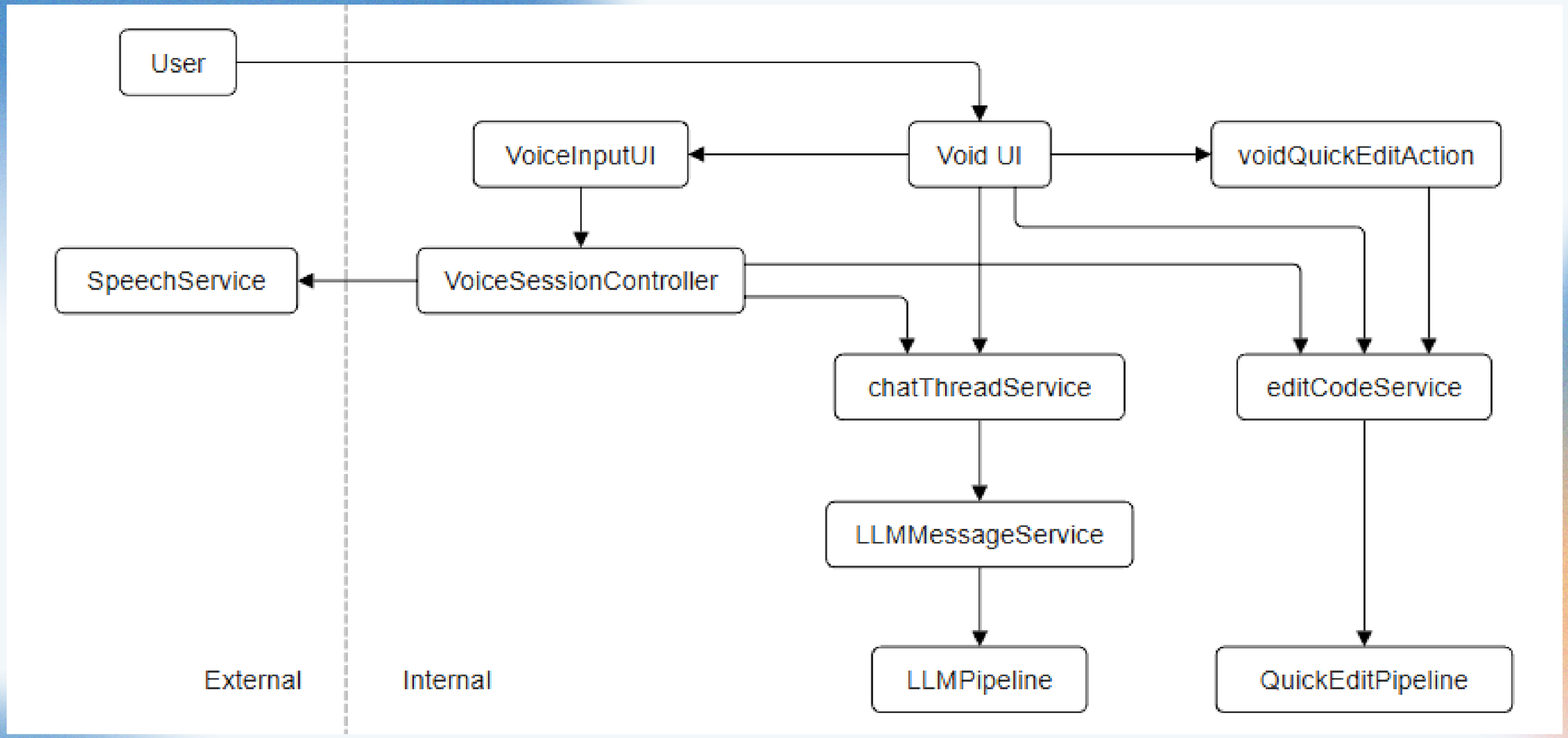
10

SEQUENCE DIAGRAM

VOICE-TO-CHAT MESSAGE



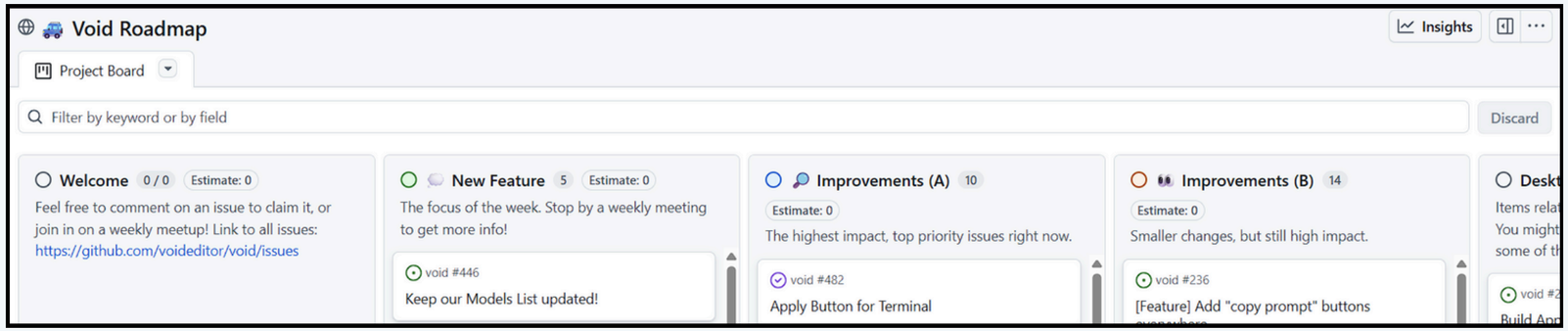
BOX-AND-ARROW DIAGRAM



LIMITATIONS OF FINDINGS

ANALYSIS IS BASED ON STATIC REASONING, NOT PROTOTYPE TESTING.

- Methodological Limits: STT latency, accent handling, and error rates were evaluated qualitatively.
- Diagram Abstraction: LLMPipeline and QuickEditPipeline are shown as single blocks. Internal message transformations and IPC timing behaviour not fully traced.
- NFR Evaluation Constraints: Concurrency cases (multi-session, rapid start/stop) reasoned about but not simulated. Findings reflect architectural expectations.
- Scope Restrictions: Integration points are prioritized over long-term maintenance issues. Limited evaluation of external provider versioning and API changes.
- Future Work: Empirical latency profiling and concurrency testing. Deeper IPC tracing and evaluation of voice command conflicts.



RISKS, LIMITATIONS, AND TESTING PLANS OF ELEMENTS

Risks:

- Interaction with more third party components. creates more reliability risks due to possible downtime.
- Passing audio files to a third party platform may pose some security risks for the user.

Limitations:

- Performance and response time of voice related features depend heavily on user hardware.
- Depending on the speech-to-text model used, certain accents may not be transcribed correctly.

Tests:

Accuracy of STT Transcription:
Must perform unit tests to ensure proper handling of accents, and various errors.

Testing Other Components' Interactions:
We must test to make sure the components connected have no errors in the process.

Testing of VoiceSessionController:
We must ensure the VoiceSessionController can accurately perform the associated commands.

LESSONS LEARNED & RATIONALE

14

ARCHITECTURAL INVARIANTS

Kept voice layer separate from A2 subsystems. Prevented architectural erosion and kept reasoning manageable

VALUE OF ALTERNATIVE ANALYSIS

SAAM comparison clarified subsystem impacts. Static processing aligned better with maintainability and reliability goals.

REPRESENTATION CONSISTENCY

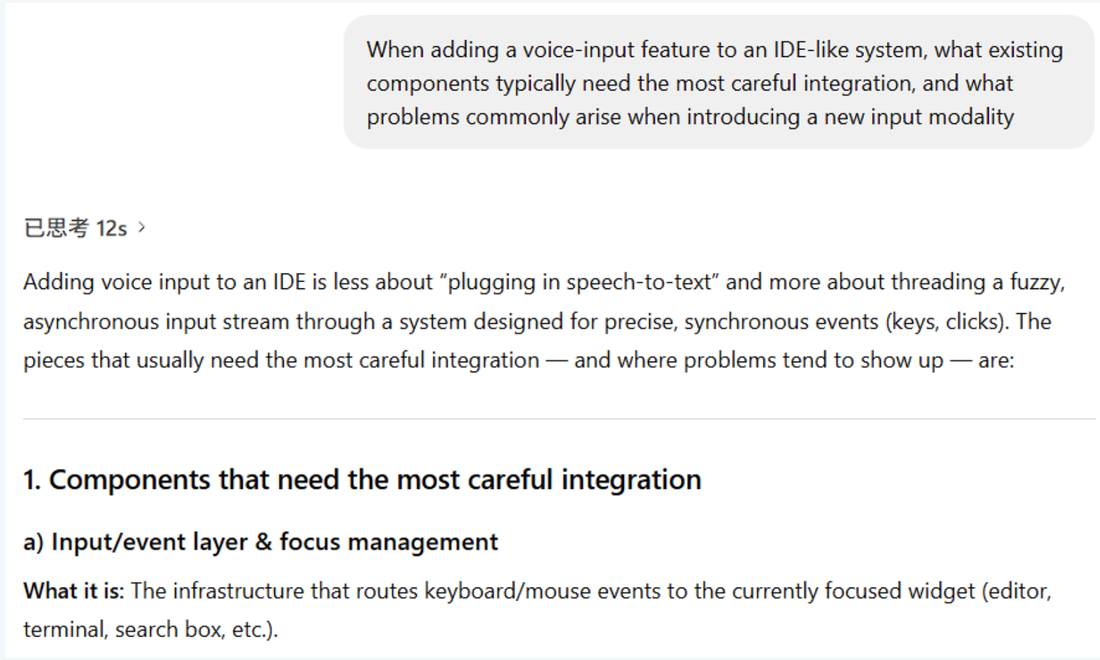
Unified sequence, box-and-line diagrams, and subsystem descriptions. Avoided the confusion experienced in assignment 2.

EXTERNAL SERVICE IMPLICATIONS

Integrating SpeechService introduced new reliability and privacy considerations. Required evaluating risks at cross-system interactions, not only inside components

AI REPORT

15



When adding a voice-input feature to an IDE-like system, what existing components typically need the most careful integration, and what problems commonly arise when introducing a new input modality

已思考 12s ›

Adding voice input to an IDE is less about "plugging in speech-to-text" and more about threading a fuzzy, asynchronous input stream through a system designed for precise, synchronous events (keys, clicks). The pieces that usually need the most careful integration — and where problems tend to show up — are:

---

1. Components that need the most careful integration

a) Input/event layer & focus management

What it is: The infrastructure that routes keyboard/mouse events to the currently focused widget (editor, terminal, search box, etc.).

ChatGPT-5.1 Model Chosen:

- For architecture clarification and terminology checks.
- Assisted with component boundaries and reasoning about alternatives.

Tasks Given:

- Each member generated questions individually, replies documented and reviewed.
- One member compiled the final AI report, prevents over-reliance and kept architecture distinct.

Prompting Strategy/Quality Control:

- Constraints applied (static-processing context, "do not rewrite meaning").
- Architectural context always provided before questions, for consistent and reproducible responses.
- Checked output with documentation, codebase, A1/A2 architecture to maintain unified writing tone.

Impact/Limitations:

- Support role, estimated contribution: ~9%. Early planning, risk comparison, clarify responsibilities.
- Incorrect suggestions corrected with group discussion.
- Confidently inaccurate claims, struggled with details (file dependencies, timing semantics).

THANK YOU FOR LISTENING!

REFERENCES

Arık, Sercan Ö., et al. "Deep voice: Real-time neural text-to-speech." International conference on machine learning. PMLR, 2017. Deep Voice: Real-time Neural Text-to-Speech.

A. Kumar, “Void IDE: The Comprehensive Guide to the Open-Source Cursor Alternative,” Medium, Mar. 24, 2025.

Microsoft Azure, “Azure AI Speech,” Azure AI Foundry, 2025.

Void Editor, “Void Official Website,” Void Editor. Available: voideditor.com.

“Void: The Open Source Cursor Alternative.” Y Combinator.

voideditor, “void/VOID_CODEBASE_GUIDE.md at main · voideditor/void,” GitHub, 2024.

Zread.ai, “LLM Message Pipeline Architecture | Voideditor/Void,” Zread, Jun. 25, 2025.

Zread.ai, “Quick Start | Voideditor/Void,” Zread, Jun. 25, 2025.