Group 25
OK_COMPUTER

Video Link: https://youtu.be/1W1fxXJ_svc

OK_COMPUTER

Thomas Schrappe - Presenter, slides, sequence diagrams

Adrian Yanovich - Presenter, slides, Reflexion Analysis, Updated Conceptual Architecture

Colin Mclaughlin (Group Leader) - Intro, Abstract, AI report

Jaiman Sharma - Second level subsystem analysis

Mantaj Toor - Methodology, high level concrete architecture

Lingwei Huang - Conclusion

A1 Recap

- Void IDE is a **VS Code extension** built on **Electron**
- Adds **AI features** like chat-based code generation and autocomplete
- Architecture is **layered**, using **client–server** and **event-driven** patterns
- Designed for **modularity** and **responsiveness**
- Key parts: **Editor Core**, **Workbench**, **UI**, and **AI Services**
- AI Services connect to **external providers** through **asynchronous IPC**
-

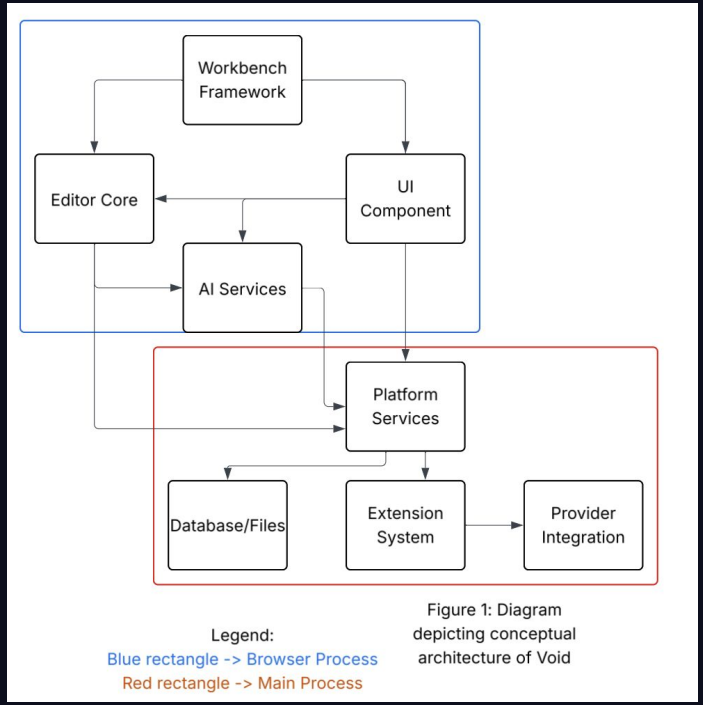
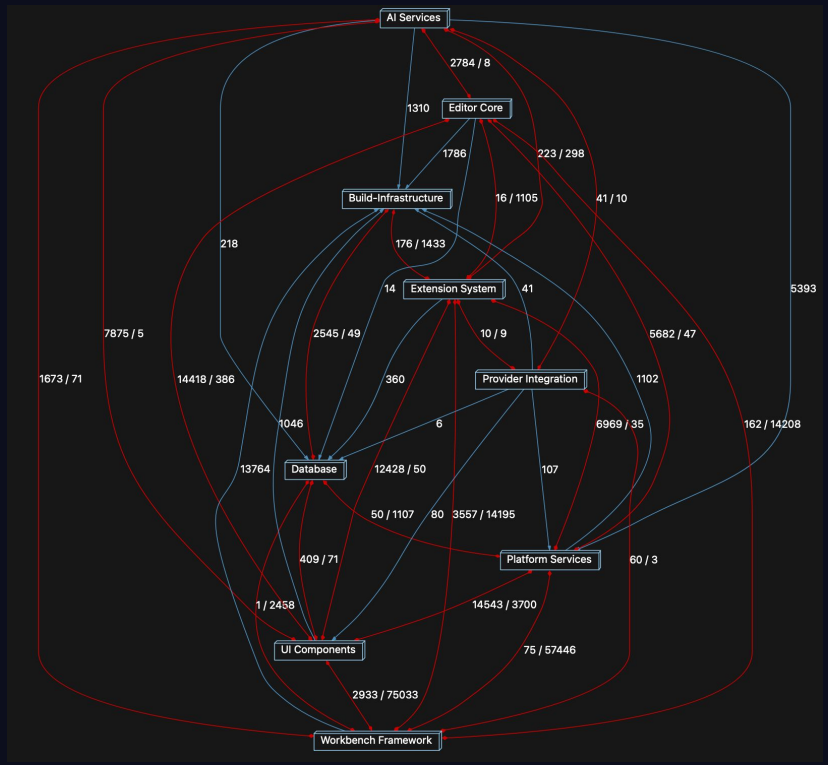


Figure 1: Diagram depicting conceptual architecture of Void

Deriving Concrete Architecture

- Analyzed Void’s source code using **Understand**
- Mapped major directories and dependencies to conceptual subsystems
- Most components matched layers like **Workbench**, **Editor**, **Platform**, and **Base**
- Discovered a **new Build/Infrastructure subsystem** for testing, bootstrapping, and CI/CD
- Structure confirms a **layered architecture** inherited from VS Code
- Upper layers depend on lower ones for services and functionality
- **Client–server** and **event-driven** patterns appear from Electron’s process separation and asynchronous IPC
- Key change: **Build/Infrastructure component** identified only in the concrete architecture



Concrete Architecture

Browser Process (blue)

- Runs the VS Code editor window and Workbench
- Hosts UI components and AI Services that interact with the user

Main Process (red)

- Handles core platform logic and system operations
- Manages provider integration, file I/O, and extension management

Core Components

- **Editor Core:** Text editing, syntax highlighting, buffer control
- **Workbench Framework:** Coordinates UI and extensions
- **UI Components:** Render panels, chat, and autocomplete
- **AI Services:** Connect to external LLMs and manage responses
- **Platform Services:** Provide low-level APIs to extensions
- **Database/Files:** Handle state persistence and workspace data
- **Extension System:** Manage activation and lifecycle of Void
- **Provider Integration:** Interface with external AI endpoints

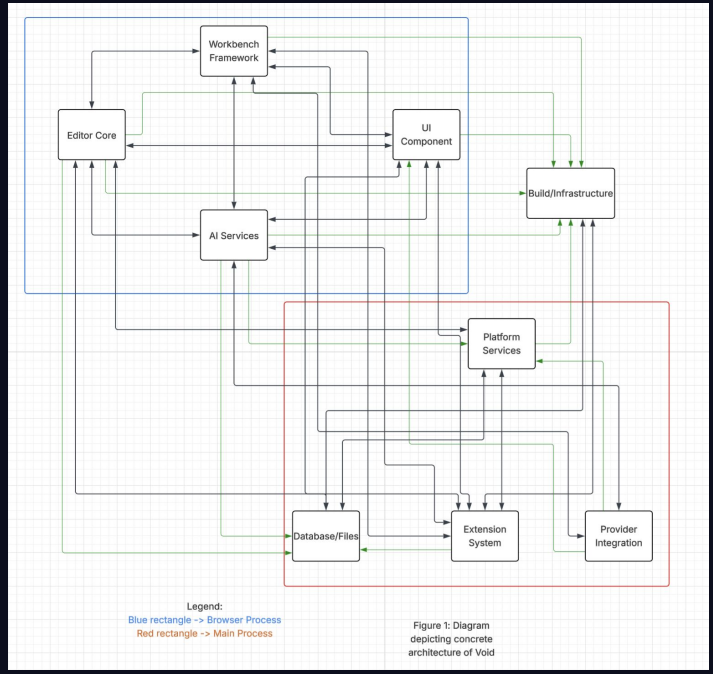


Figure 1: Diagram depicting concrete architecture of Void

Second-Level Subsystem: Void Extension

The Void Extension subsystem implements the IDE's AI-assisted functionality within VS Code's Extension Host using layered and event-driven design.

Its main services—LLMMessageService, chatThreadService, editCodeService, and voidSettingsService-handle AI communication, session state, code editing, and configuration.

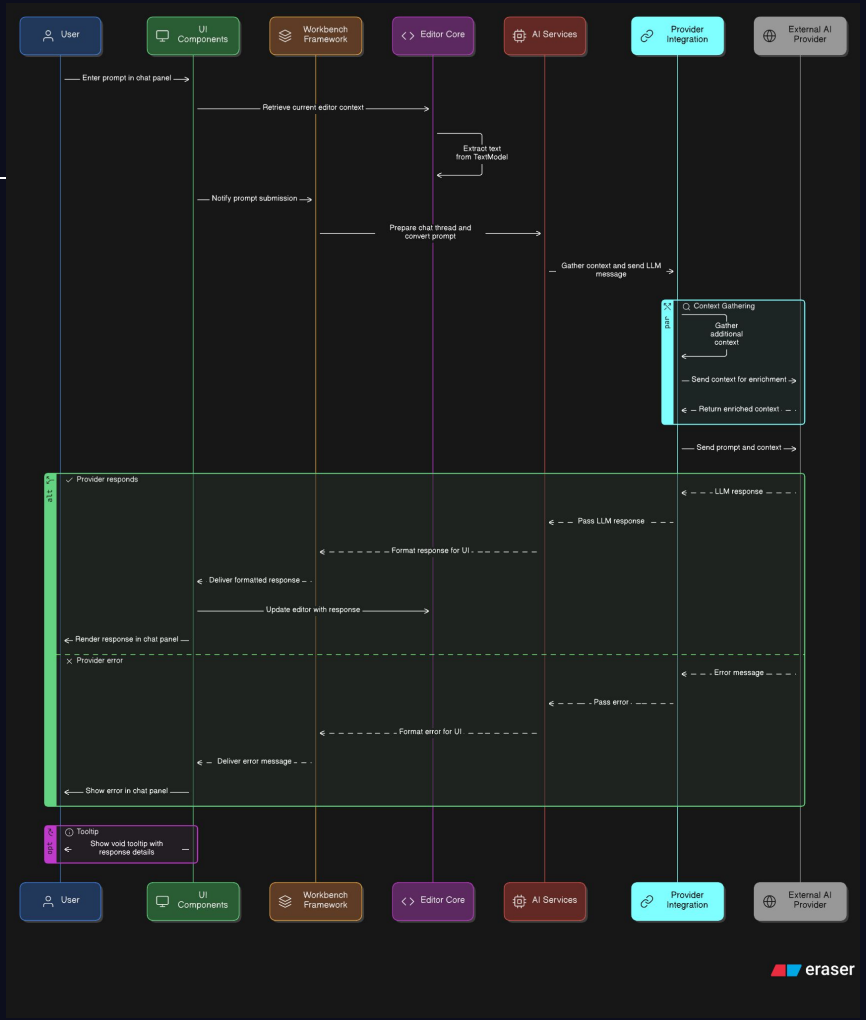
These services interact with VS Code Core asynchronously through message passing, maintaining isolation and modularity.

This design enables concurrent AI requests and real-time updates without blocking the editor's responsiveness.

Sequence Diagram

Use Case: Prompt request to response render.

The user submits a prompt through the UI, which the Workbench Framework captures and enriches with current editor context from Editor Core. AI Services formats this into a structured message and sends it through Provider Integration to the external AI provider. The response streams back in chunks through the same path, with each chunk flowing from Provider Integration -> AI Services -> Workbench Framework. Finally, UI Components render the streaming response in real-time as it arrives.



Reflexion Analysis

- Analysis on divergences between Conceptual and Concrete Architecture
- AI Services Divergence
  - Planned: UI -> AI Services -> Provider Integration
  - Found: UI files talk directly to provider logic
  - Evidence: chatPanel.tsx, editCodeService.ts use sendLLMMessageService.ts and modelCapabilities.ts
-
- Build/Infrastructure Divergence
  - Planned: Build handled inside Platform Services
  - Found: Separate build/ folder, webpack.config.js, and CI pipeline

Limitations & Alternatives

Alternatives Considered

- Considered a **pipe-and-filter architecture** since the AI layer processes data in stages (input → inference → response)
- Rejected because it fits **linear, batch workflows**, not interactive systems
- Void is **event-driven and concurrent**, handling multiple AI queries asynchronously

Chosen Approach

- Kept a **hybrid layered, client–server, and event-driven** structure
- Better reflects real data flow
- Supports **modularity** and **responsiveness**

Limitations

- Hard to find direct **developer communications** on GitHub
- Had to **infer interactions** by reading source code
- Relied on **documentation** to understand IPC (inter-process communication) flow

Concurrency & Project Team Issues

The multi-process architecture separates UI rendering from AI inference, allowing frontend and backend teams to develop in parallel without blocking each other.

Layered structure creates natural ownership boundaries where different teams can own the UI layer, service layer, and platform layer with clear responsibilities.

Event-driven patterns reduce coordination overhead since components communicate through loose event subscriptions rather than tight coupling. Modular services like chatThreadService and editCodeService enable independent feature development as long as teams respect shared interface contracts.

AI Collaboration

- Used GPT-5 (Oct 2025) for structured writing, rubric alignment, and editing consistency.
- Used Claude Sonnet 4.5 (via Cursor) for file-system parsing and interpreting data from SciTools Understand and the Void GitHub repository.
  AI usage increased from Assignment 1 due to deeper code and dependency analysis needs.
- Other models (Gemini 2.5, Grok 4, 03) were tested but found less reliable for large-context code analysis.
- Finally, Eraser was used to convert a description of our use case diagram into a cleaner, more polished looking diagram.

AI mostly worked with technical details and touch ups, making clean diagrams, touching up text, and interpreting data.
Overall we believe AI contributed to around 35% of A2.

Lessons Learned & Wrap-Up

The recovered architecture confirmed that Void's layered design is modular and consistent with its conceptual model, with Electron, VS Code Core, and the Void extension forming a clear service-oriented structure.

The Extension Host and its async IPC were validated as key to responsiveness and process isolation. Using SciTools Understand proved useful for dependency mapping but required manual interpretation to avoid misleading results.

Overall, the project highlighted how architectural recovery bridges design and implementation, emphasizing the need for clear documentation and disciplined validation.