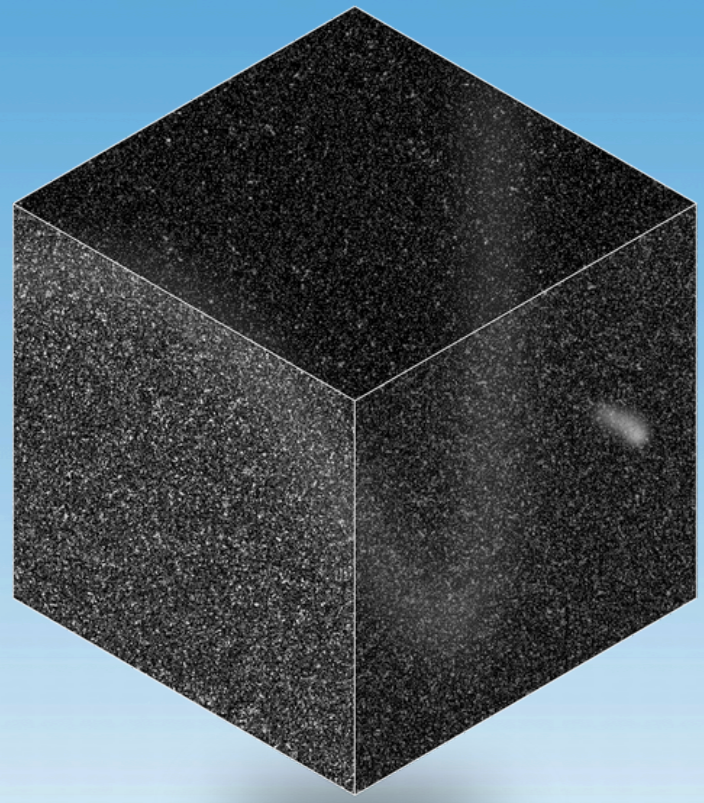
https://www.youtube.com/watch?v=2b3fnQ7STyY

Conceptual Architecture of Void CISC 322

Authors: Matt Dobaj, Shaun Thomas, Babinson Batala (Presenter), Esmé Mirchandani (Group Leader), Sandy Mourad (Presenter), Nihal Kodukula

An investigation into Void's conceptual architecture, noting key features and requirements, and comparing it to its fork source, VS Code.



Void's Architectural Foundation: VS Code & Electron

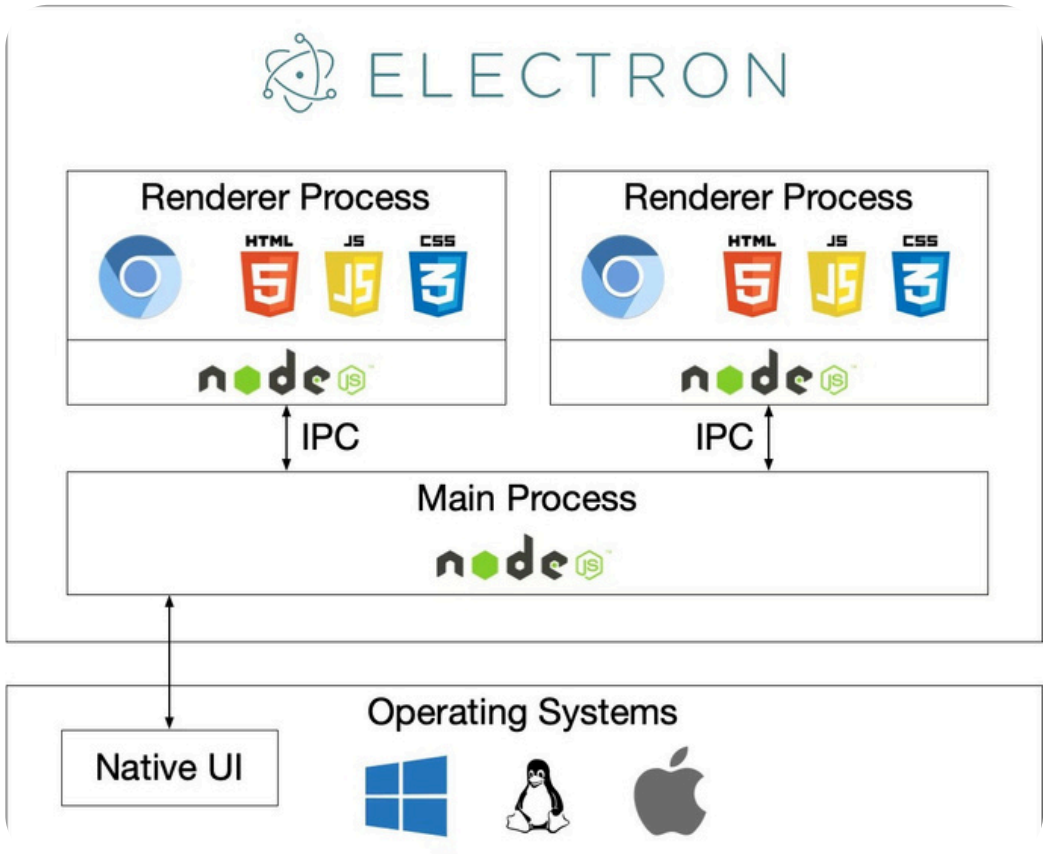
Fork of VS Code

Void is a fork of Microsoft's open-source Visual Studio Code (VS Code), inheriting it’s core structure and extensibility.

Built on Electron

Both projects use the Electron framework, which splits the application into two processes:

- **Browser Process (Client):** Handles the user interface (UI).
- **Main Process (Server):** Manages system logic and file operations.



Void's architecture is very similar to VS Code, primarily utilizing **Layered** and **Client-Server** styles.

Extending the Architecture for AI Capabilities

Void integrates AI capabilities like in line code completion and chat-based code generation into the familiar VS Code environment.

Inline Code Completion

Real-time suggestions while the user types in the editor.

Chat-Based Generation

Natural language prompts generate new code snippets.

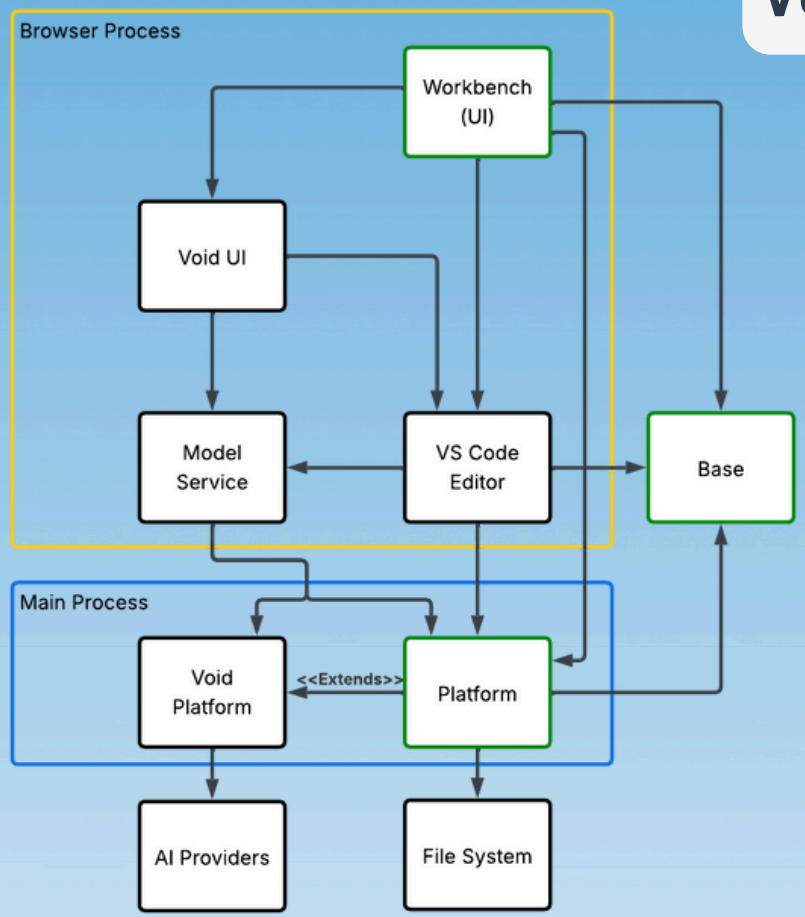
Context Generation

Reads user files to provide better context to external AI Providers.

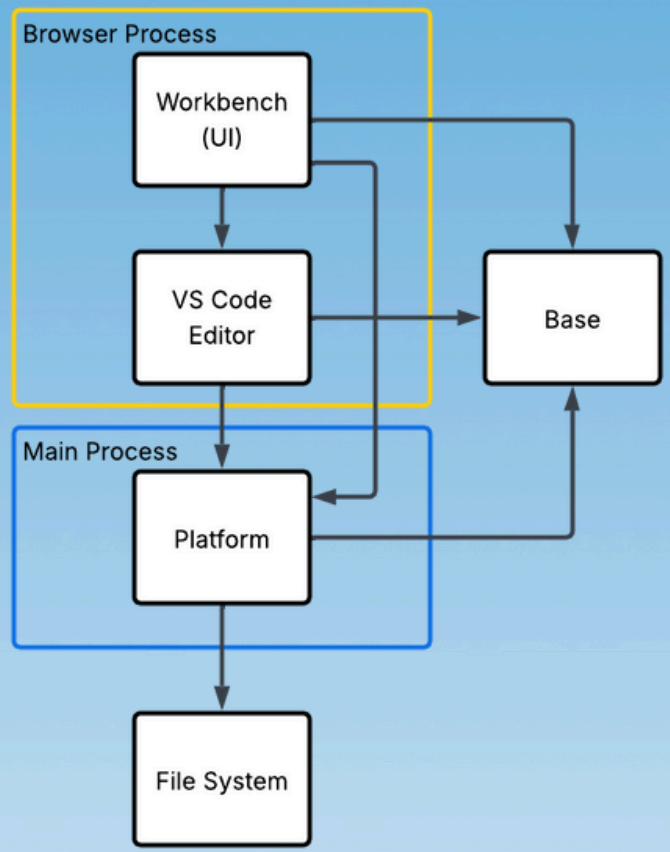
This analysis focuses on how Void introduced AI-driven functionality while maintaining VS Code's stability and extensibility.

Void's Conceptual Components

Void-Enhanced Architecture



Baseline VS Code Architecture



Void uses all of VSCode's components but adds or modifies four key components to support AI functionality.

Void UI

Handles all user-facing elements: inline completion, prompt page, settings, and model selection.

VS Code Editor (Modified)

Contains the Monaco editor logic, extended to communicate with the Model Service for AI completion.

Model Service

Formats and prepares AI messages, gathering context before sending to the AI Provider via the Platform.

Void Platform

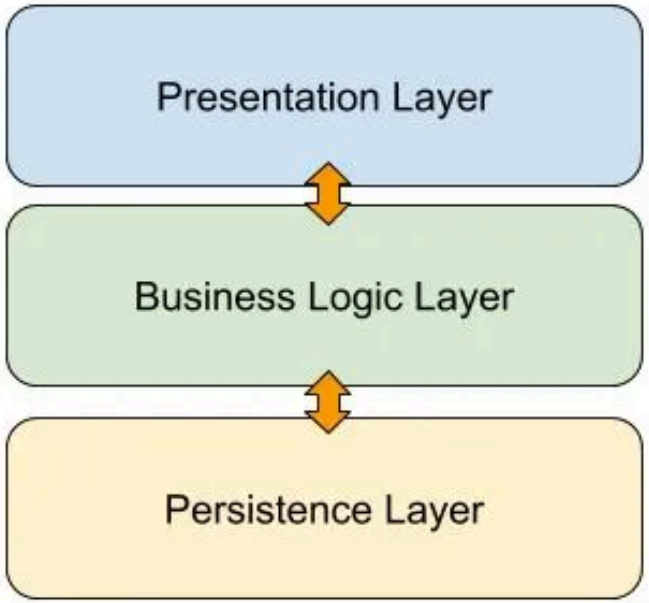
Extends VS Code's platform to directly prompt various external AI providers.

Architectural Styles: Layered & Client-Server

Layered Architecture

Void uses components of ascending abstraction, where each layer provides services to the one above it and uses services from the one below.

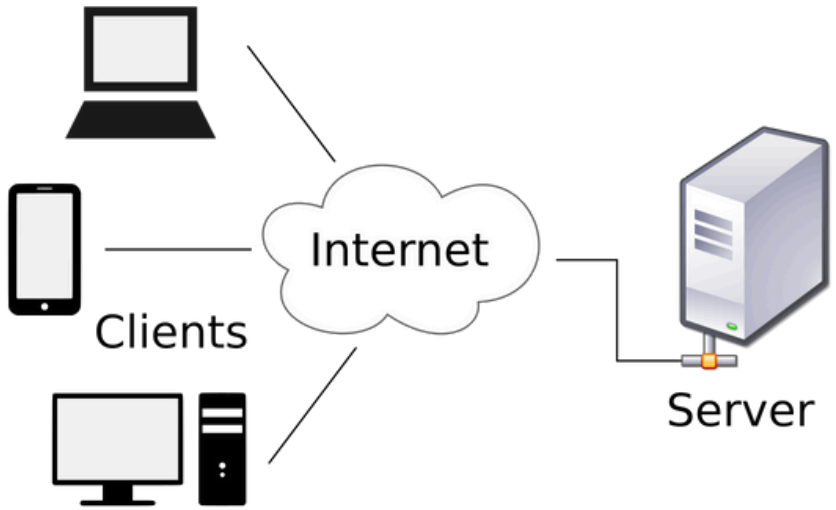
- **Logical Hierarchy:** UI handles interactions, Model Service handles prompt processing, and Platform handles external services.
- **Benefit:** Enhances modularity and reuse; changes to one layer (e.g., UI framework) largely avoid modifying others.



Client-Server Architecture

The system islogicallysplitinto Electron's Browser (Client) and Main (Server) processes.

- **Client (Browser Process):** UI and Editor components request services.
- **Server (Main Process):** Provides services like file system access, command execution, and AI provider interfacing.
- **Performance:** Avoids network latency by using Electron's Inter-Process Communication (IPC) for local communication.



Quality Attributes: Performance & Extendability

Concurrency

The multi process architecture allows the UI (Browser) and backend tasks (Main) to run concurrently. This ensures the application remains responsive during heavy server tasks like AI operations or file I/O.

Performance Requirements

Void requires low response time for critical functions like inline code completion to prevent a mismatch between user input and suggestions. The chat prompt, while valuing speed, is less critical.

High Extendability

Inheriting from VSCode, Void provides an extensive Extension API. Its layered structure means new functionality (e.g., a new AI model) can be added by modifying only the relevant components (e.g., Void Platform) with minimal disruption to others.

External Interfaces: Data Flow

Void exchanges information through four key interfaces, connecting user input, AI backends, and system resources.

Graphical User Interfaces (GUIs)

User input (prompts, commands, config) sent via Electron IPC. Receives streamed AI responses and contextual updates.

AI Provider Interfaces

Main Process manages communication with external LLMs. Sends structured prompts and receives streamed model responses and metadata.

File and Platform Interfaces

Handles OS and file-level communication (open, save, metadata). Uses IPC locally or remote protocols for remote development.

Extension and Data Interfaces

Manages third-party plugins, persistence, and analytics via the VS Code Extension API, ensuring modular integration.

Use Case 1: Inline Code Completion

The system automatically suggests an inline code completion as the user types.

User Action

User types code, triggering a codeComplete() event in the Editor.

Context Retrieval

Void Platform queries the File System for project context.

Prompt Formation

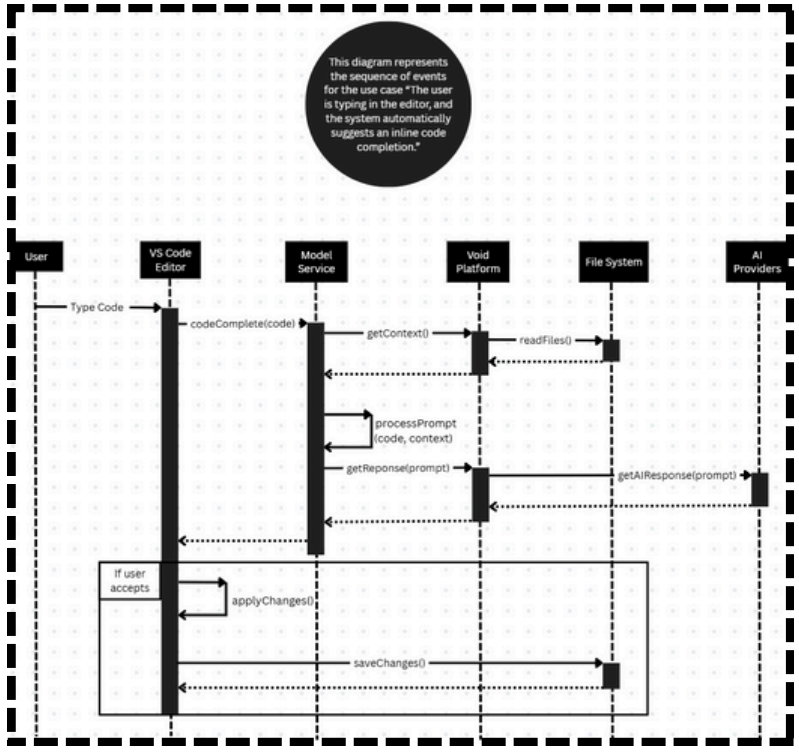
Model Service processes input + context to form a prompt for the AI Provider.

Suggestion Delivery

AI Provider generates a completion. Model Service sends it back to the Void Platform, which inserts the suggestion inline.

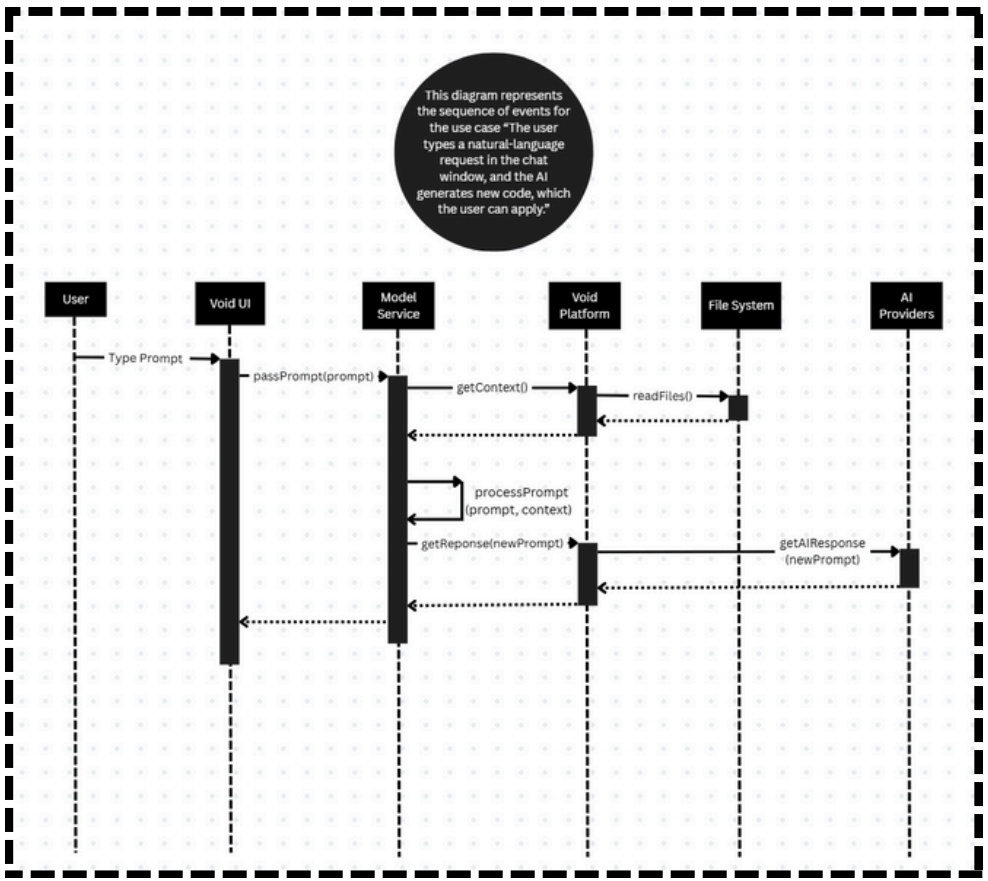
Acceptance

If accepted, the editor applies and saves changes in the File System.



Use Case 2: Chat-Based Code Generation

The user types a natural-language request, and the AI generates new code for the user to apply.



User Prompt

User types a natural-language prompt into the Void UI chat window.

Context Request

Model Service requests relevant code context from the Void Platform.

File Retrieval

Void Platform retrieves necessary files from the File System.

Data Processing

Model Service processes the combined data (prompt + context) and forwards it to the AI Provider.

Display Suggestions

AI Provider returns generated code. Void UI displays these suggestions for the user to apply.

Conclusion & Lessons Learned

Key Architectural Findings

- Void is a robust evolution of VSCode, using Layered and Client-Server styles.
- New components (Void UI, Model Service, Void Platform) enable seamless AI integration.
- Electron IPC reduces network dependency, ensuring responsiveness during AI interactions.
- The design supports third-party extensions and future AI model integration.

Future Measures

Clarify the Model Service's ambiguous placement, optimize performance via caching, and formalize security/privacy measures for data handling.

Reflection on Teamwork

The team learned the importance of more frequent, shorter check-ins to coordinate progress and ensure flow between sections. Sharing drafts sooner was crucial for continuous integration and feedback.

AI Collaboration (ChatGPT)

ChatGPT served as a supplementary reviewer (approx. 10% contribution) for verification, clarification, and formatting. All critical analysis and architectural decisions were strictly human-made.

AI tools are most effective when used as assistants, not decision makers.