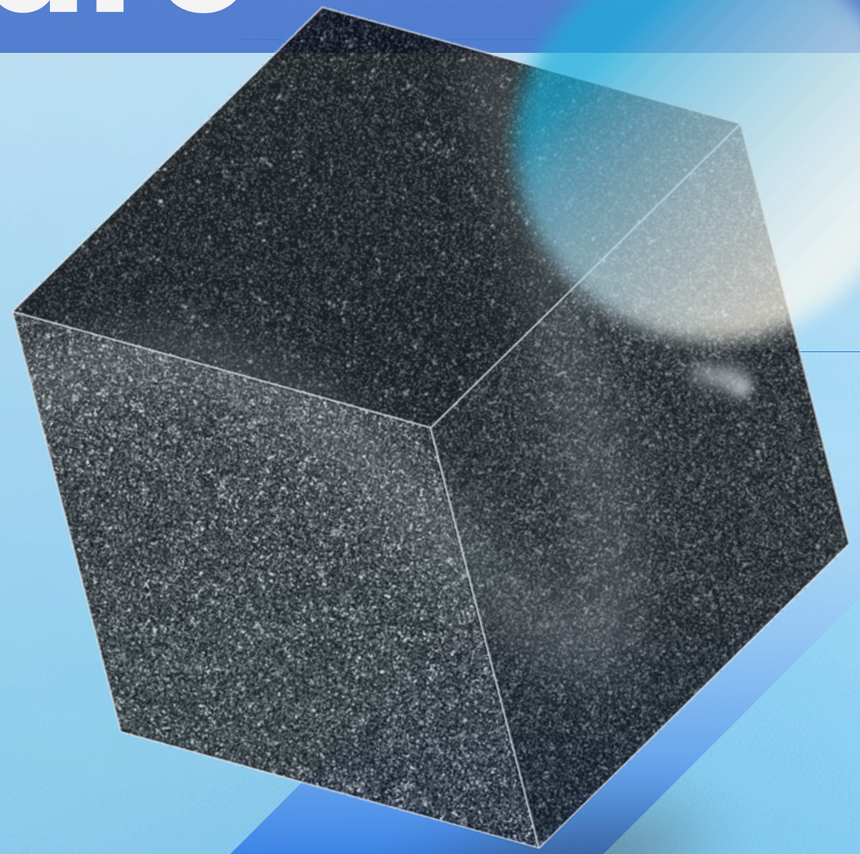
Concrete Architecture Of Void - CISC322



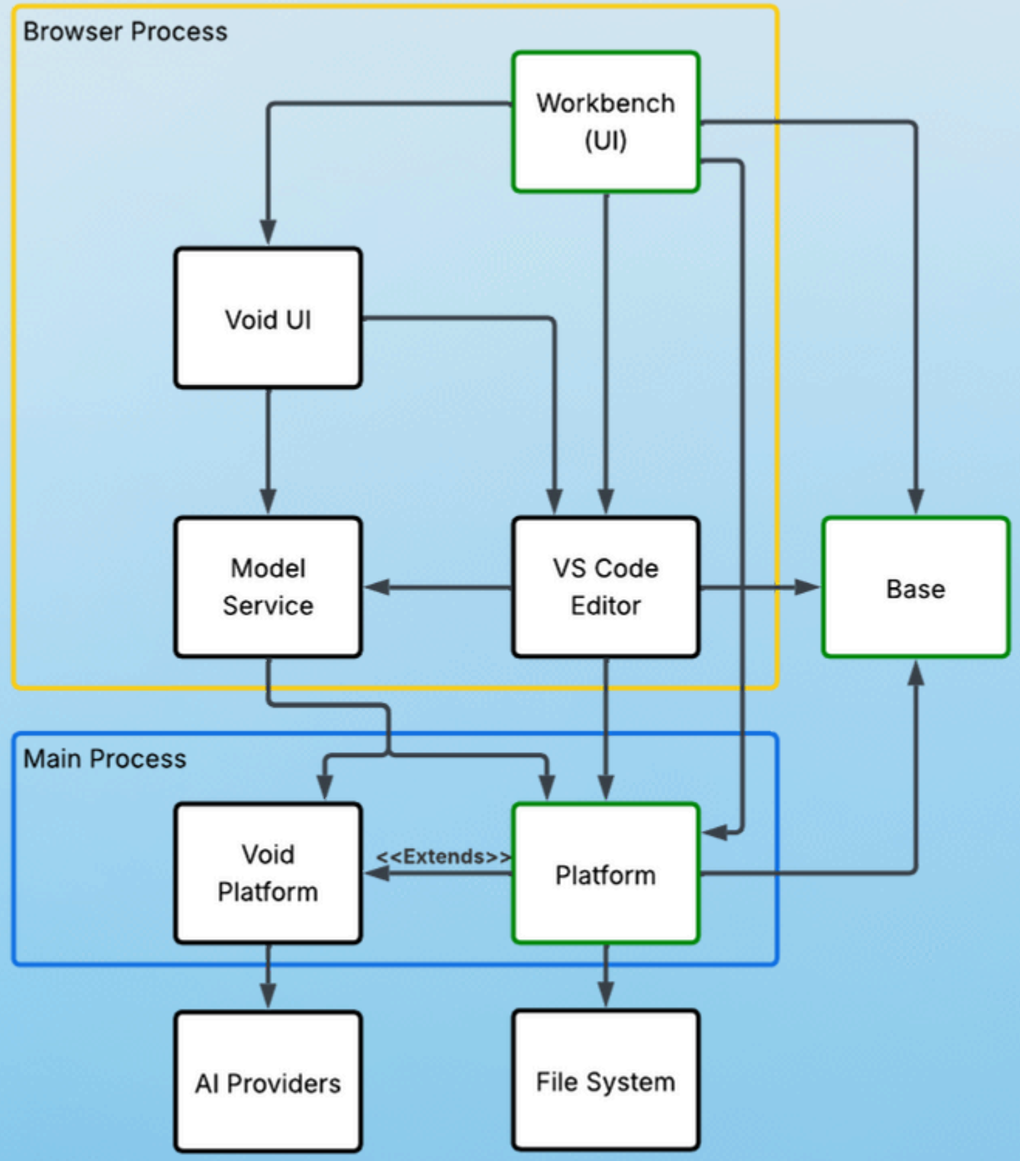
Authors: Matt Dobaj, Shaun Thomas, Babinson Batala (Presenter), Esmé Mirchandani (Group Leader), Sandy Mourad (Presenter), Nihal Kodukula

https://youtu.be/OLB_U2Hxq3Y

Agenda

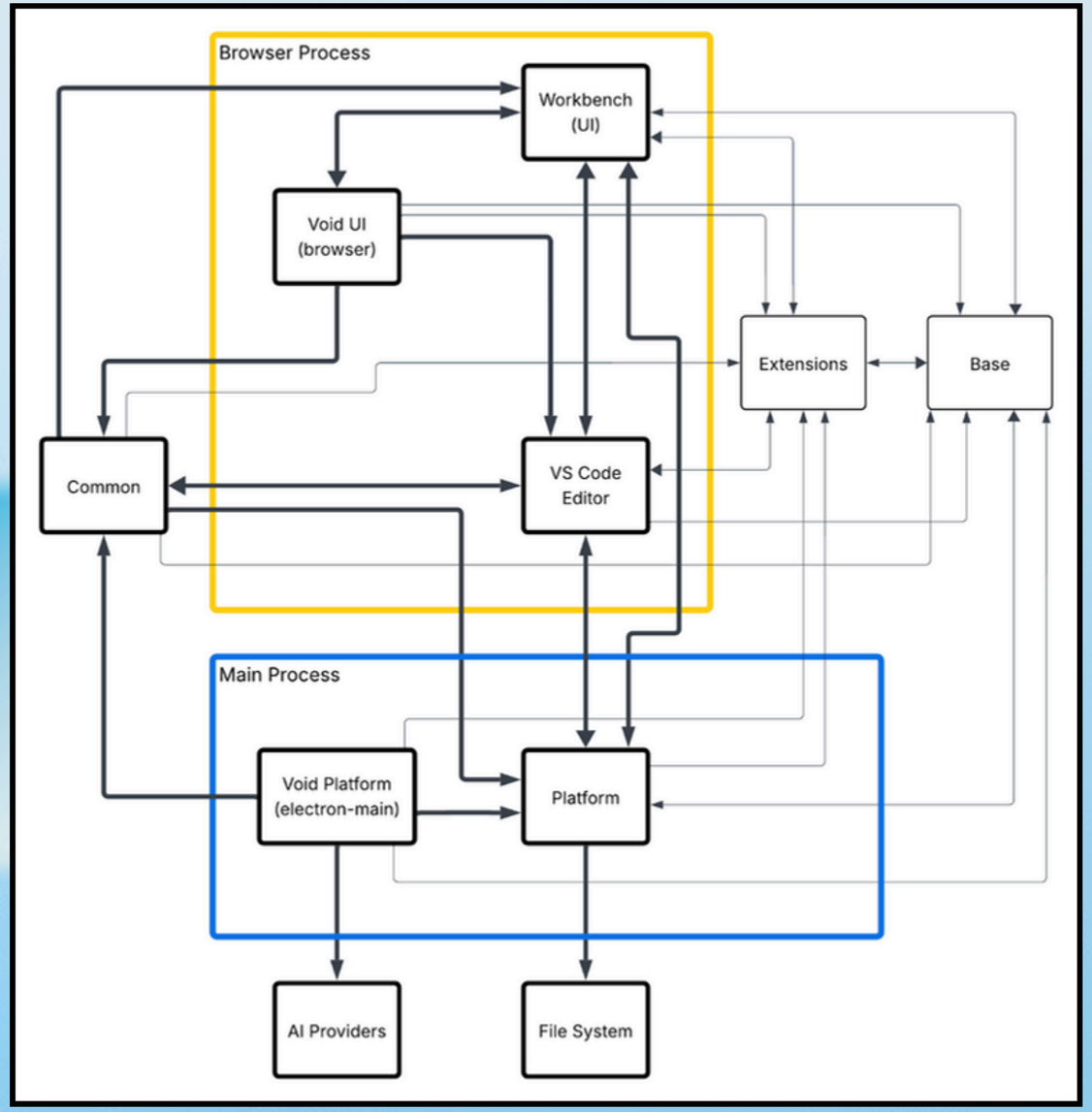
1- Reflexion Analysis of conceptual VS concrete architecture
2- Subsystem breakdowns and use cases
3- Architectural styles and performance
4- Lessons learned and AI support

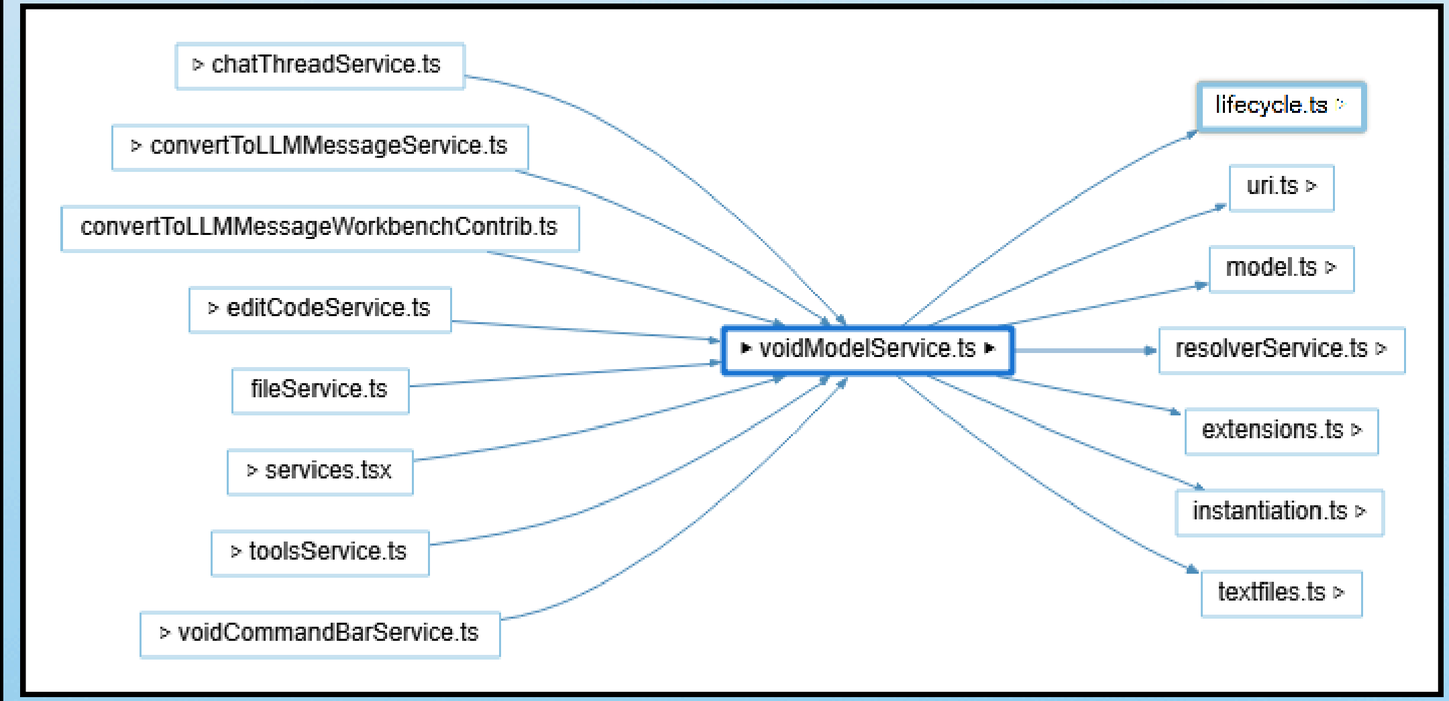
Recap: Conceptual Architecture (A1)



- Expected clean layered design
- Components: UI | Model Service | Platform | Editor
- Built on Electron (browser ↔ main)

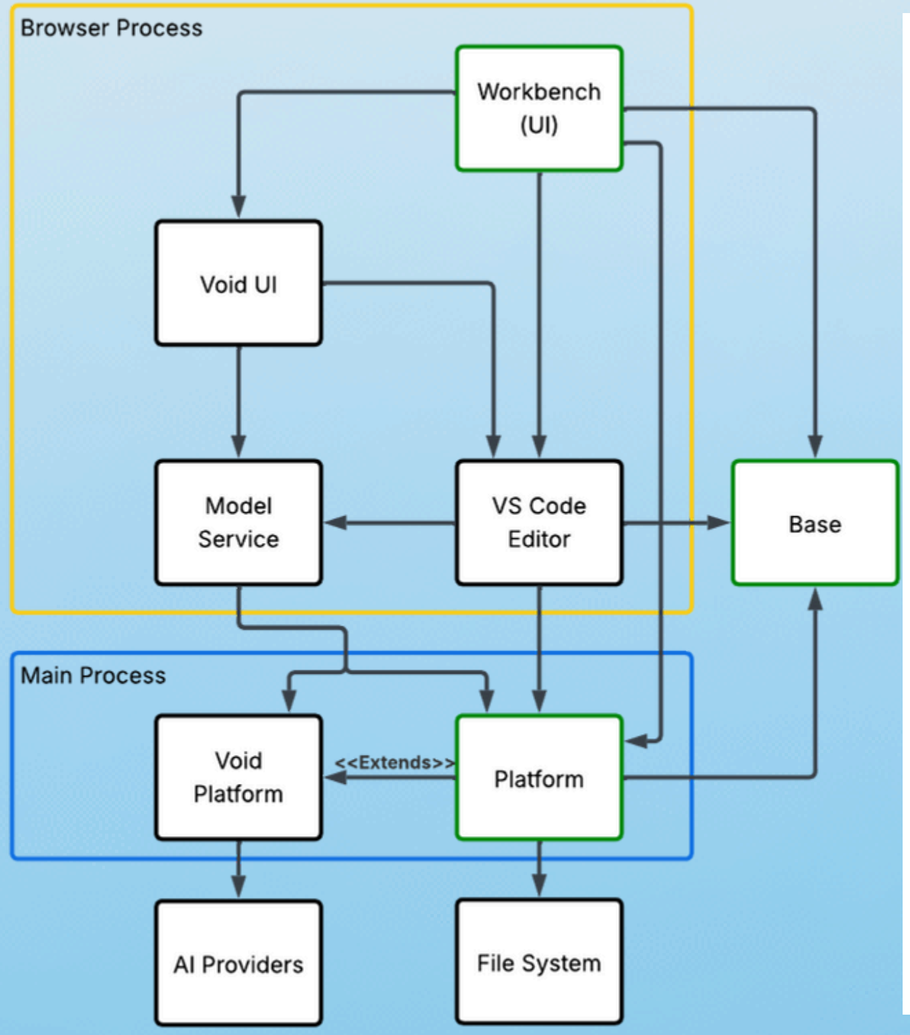
Derivation Methodology (Reflexion + Code tracing)



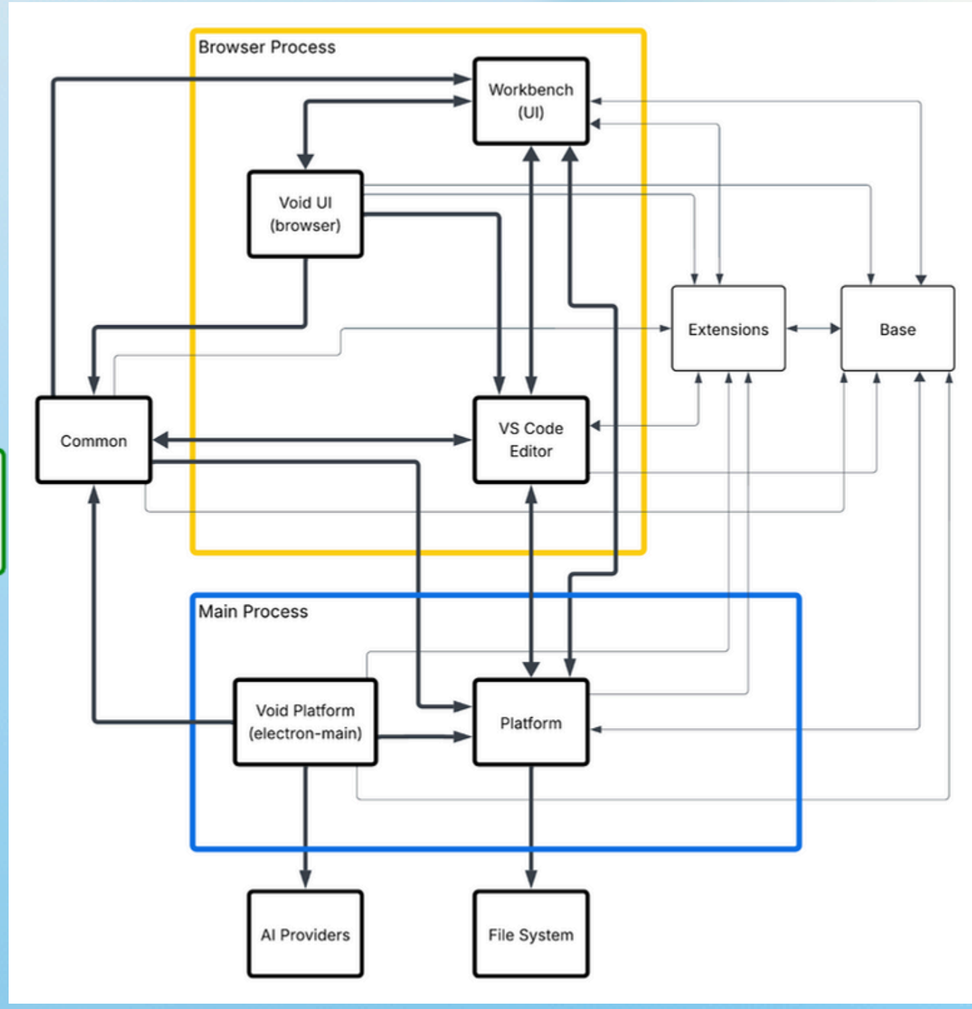


Divergences - What changed and why

- A1 model: 3-layer design → Void UI, Model Service, Void Platform
- Assumed: Editor depends on Model Service for inline completions
- Understand analysis revealed many new dependencies
- Bidirectional relationships between components
- Model Service → Common (shared logic across UI & Platform)
- All components depend on Base (VS Code core utilities)
- Added Extensions layer after repeated code references
- New diagram reflects real, interconnected architecture



A1 - Old



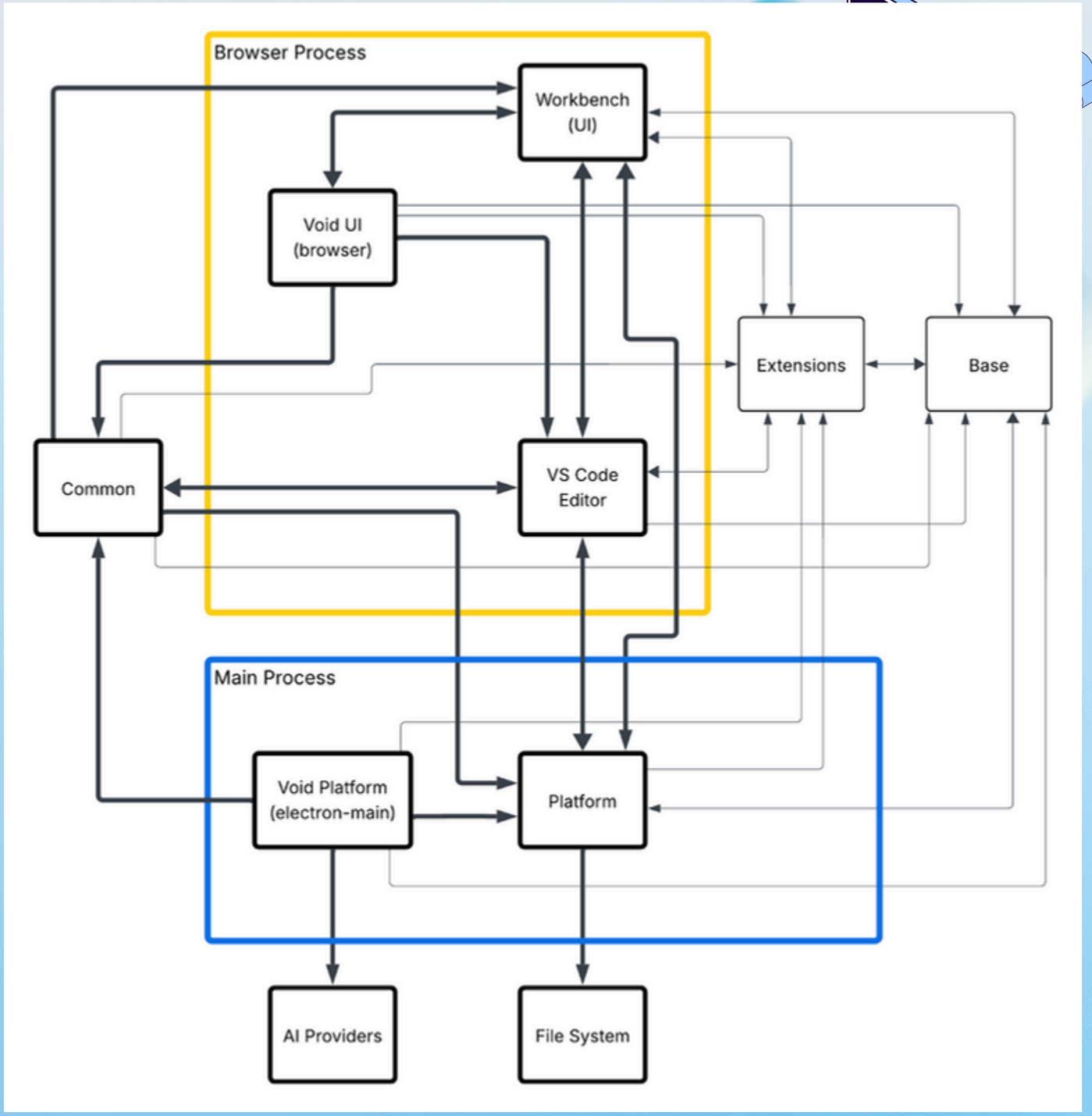
A2 - New

Concrete arch overview - Component Breakdown

1. Five VS Code components:
   a. Base, Workbench, Editor, Platform, Extensions
2. Five Void-specific components:
   a. Void UI, Common, Void Platform (+ shared logic)
3. Each plays a distinct role within the Layered + Pub-Sub system
4. All components depend on Base, and most on Extensions
5. Supports modular, scalable, AI-integrated design

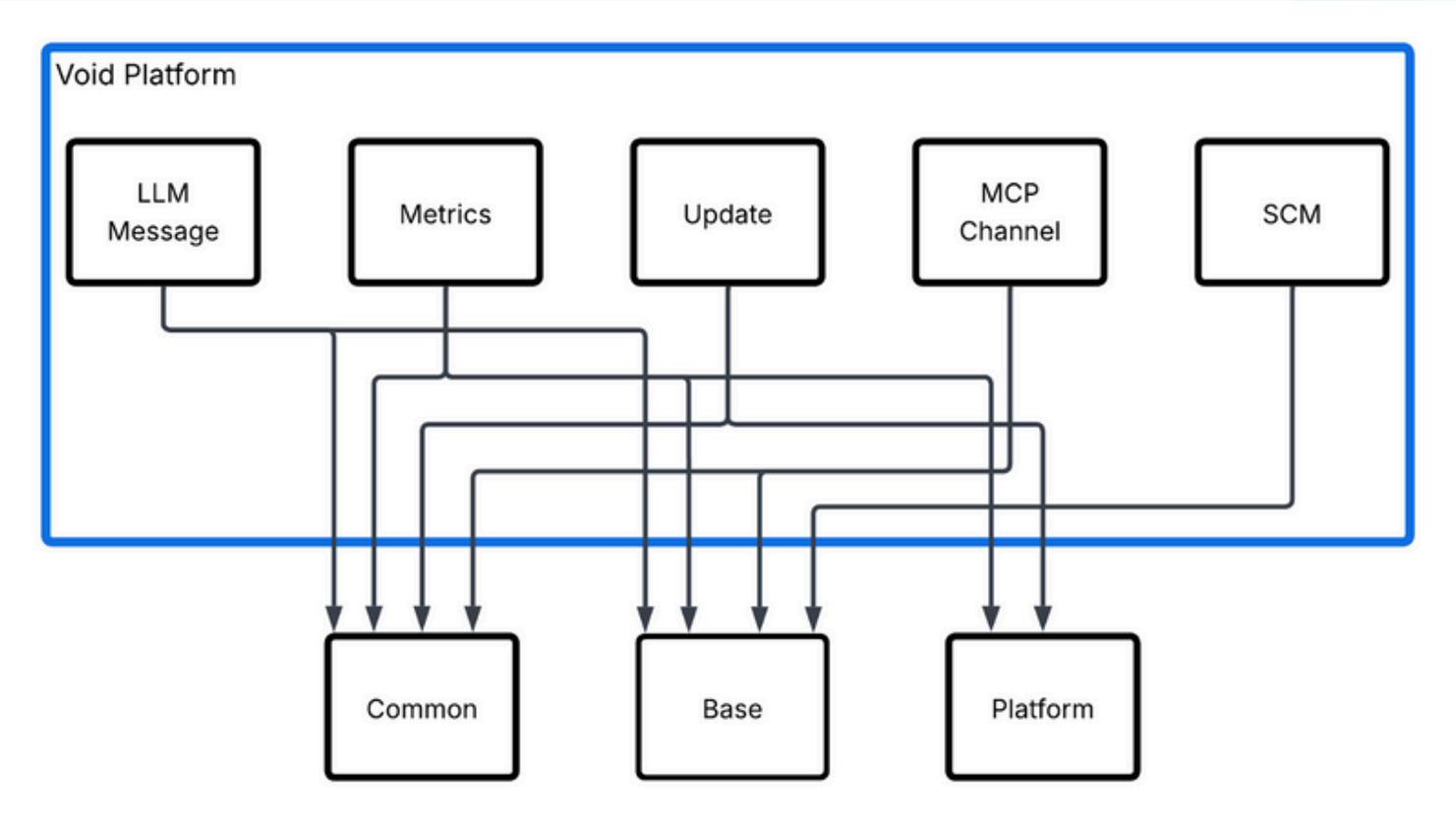
Motivations for additions:

1. Common: eliminate duplication; run in Browser or Main; unify types & services
2. Extensions: consolidate integration points; reduce ripple effects; plugin ecosystem
3. Outcomes: ↓ coupling, ↑ cohesion; faster feature onboarding; safer provider swaps



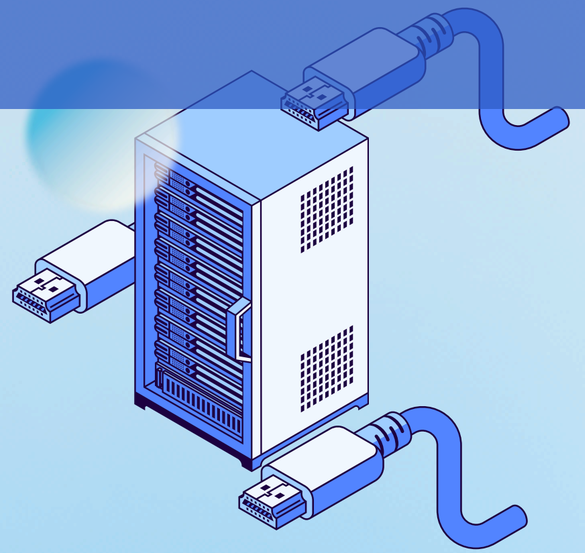
Subsystem: Void platform

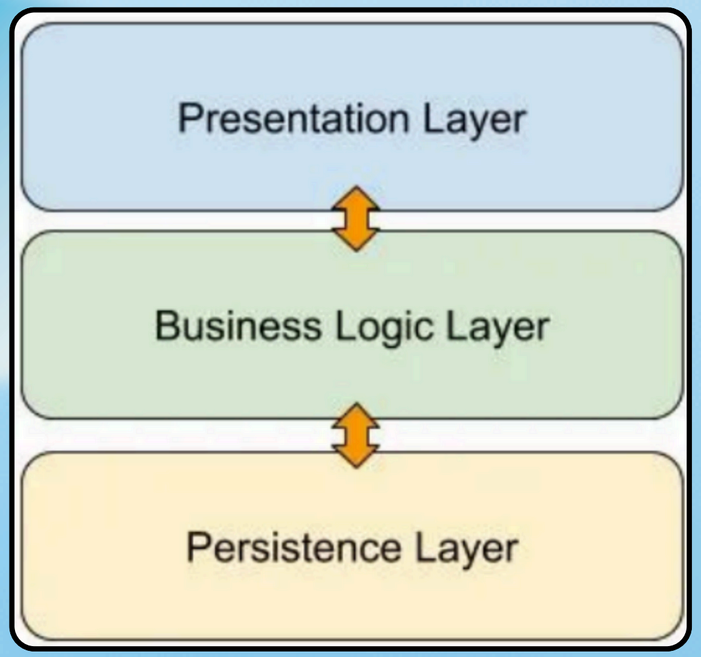
1. LLM Message: receive IPC from Common → call provider → post-process (code/MCP)
2. MCP Channel: handle remaining MCP ops on Main; UI handles primary ones
3. SCM: read repo state for commit-message gen (via Base/IPC)
4. Metrics: aggregate via IPC → external metric service
5. Update: wrap VS Code updater → target Void binaries

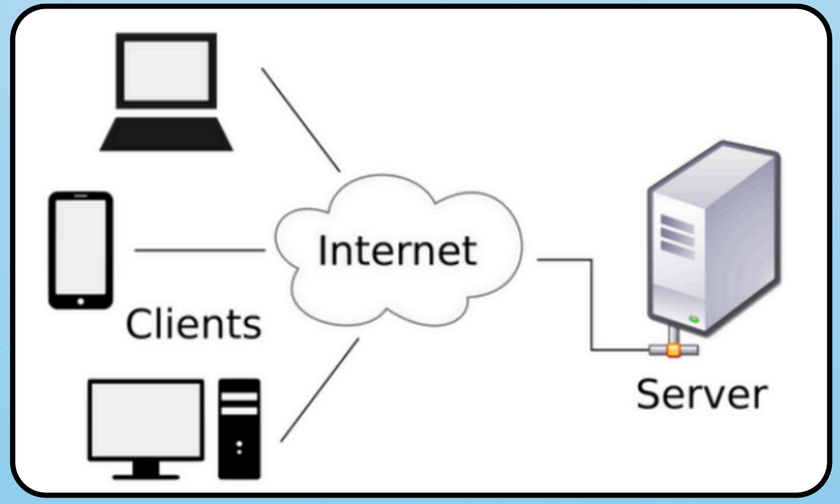


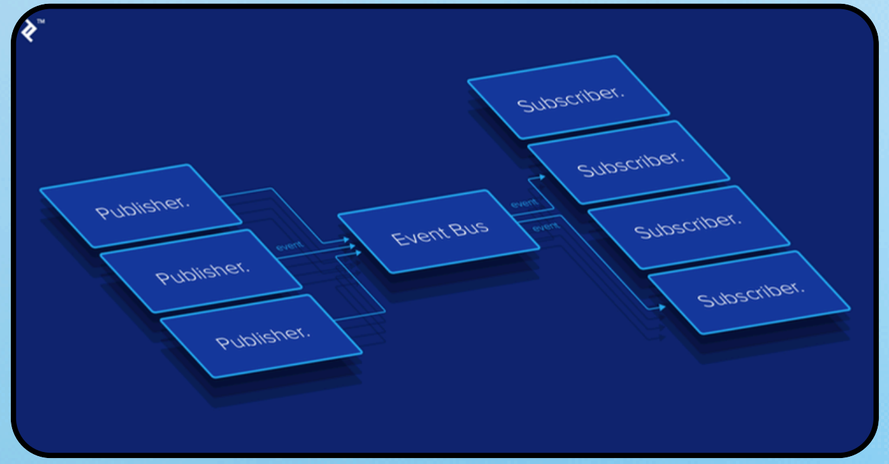
Concrete Architecture styles

1. Chosen:
    a. Layered (UI/Common/Platform/Base) → abstraction clarity
    b. Client-Server (Browser client ↔ Main server) → process isolation
    c. Publish-Subscribe (IPC listeners/channels) → loose coupling, extensibility
2. Considered / Rejected:
    a. Pipe-and-Filter: non-standardized I/O; shared state violates invariants

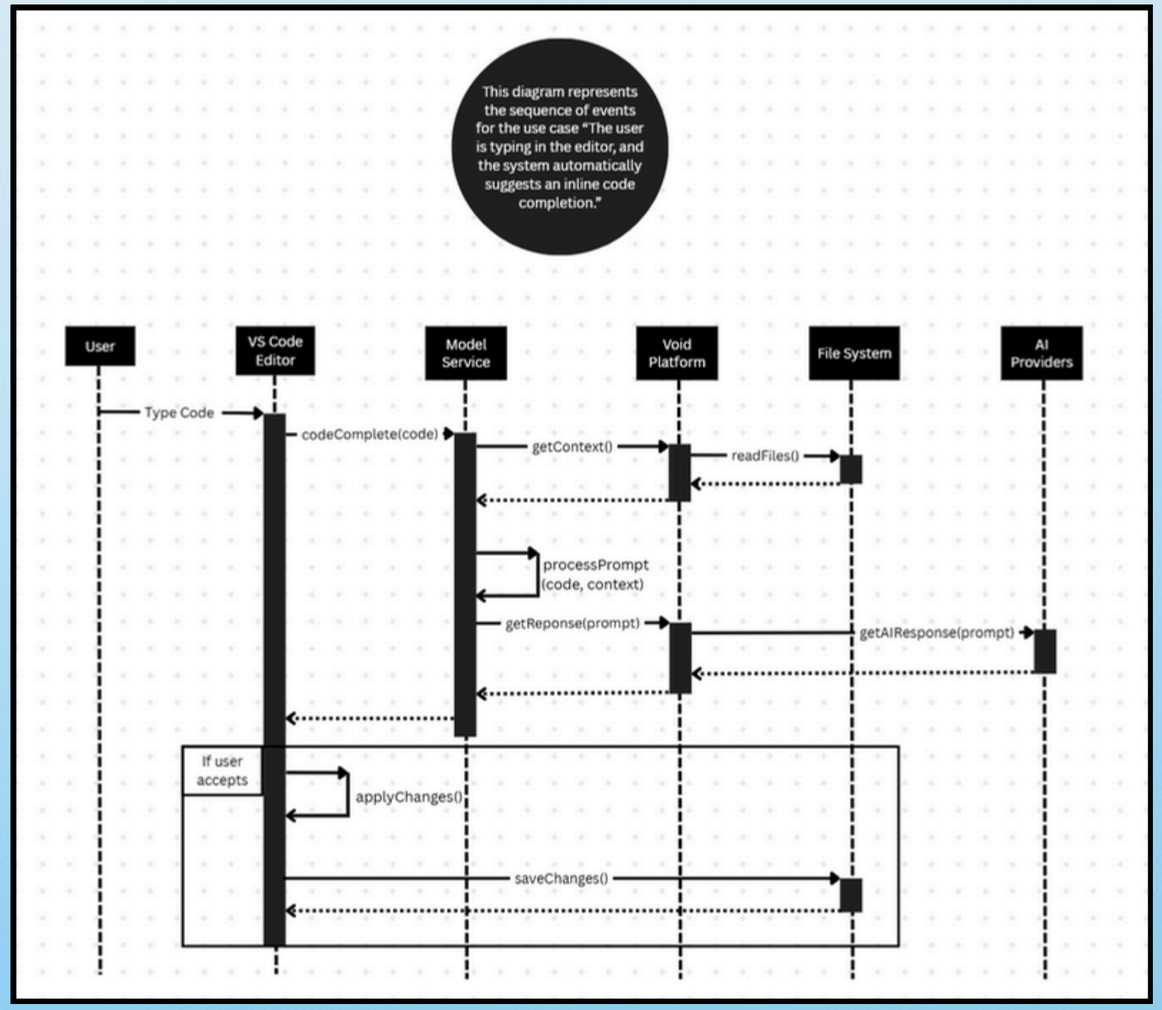


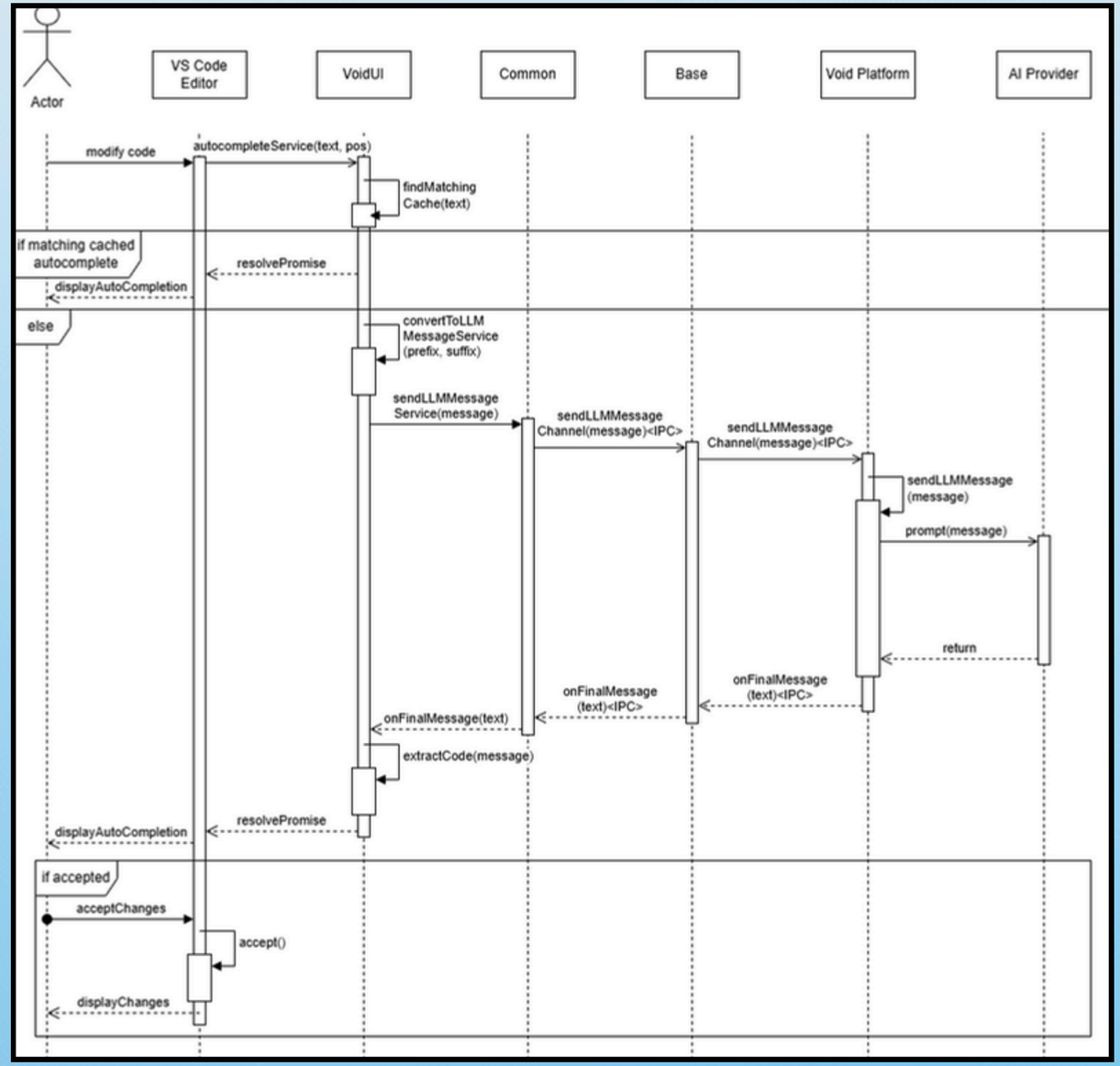




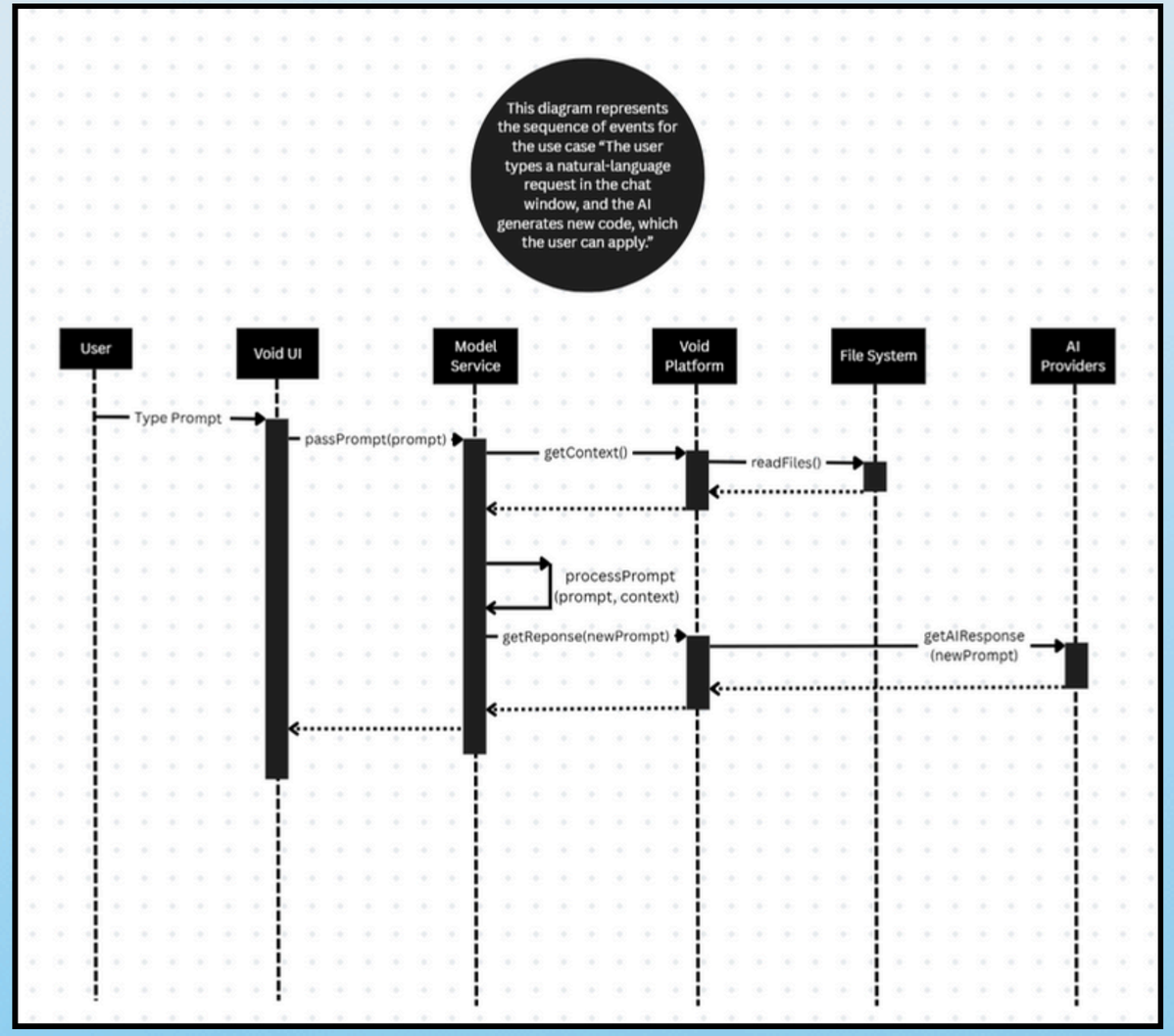


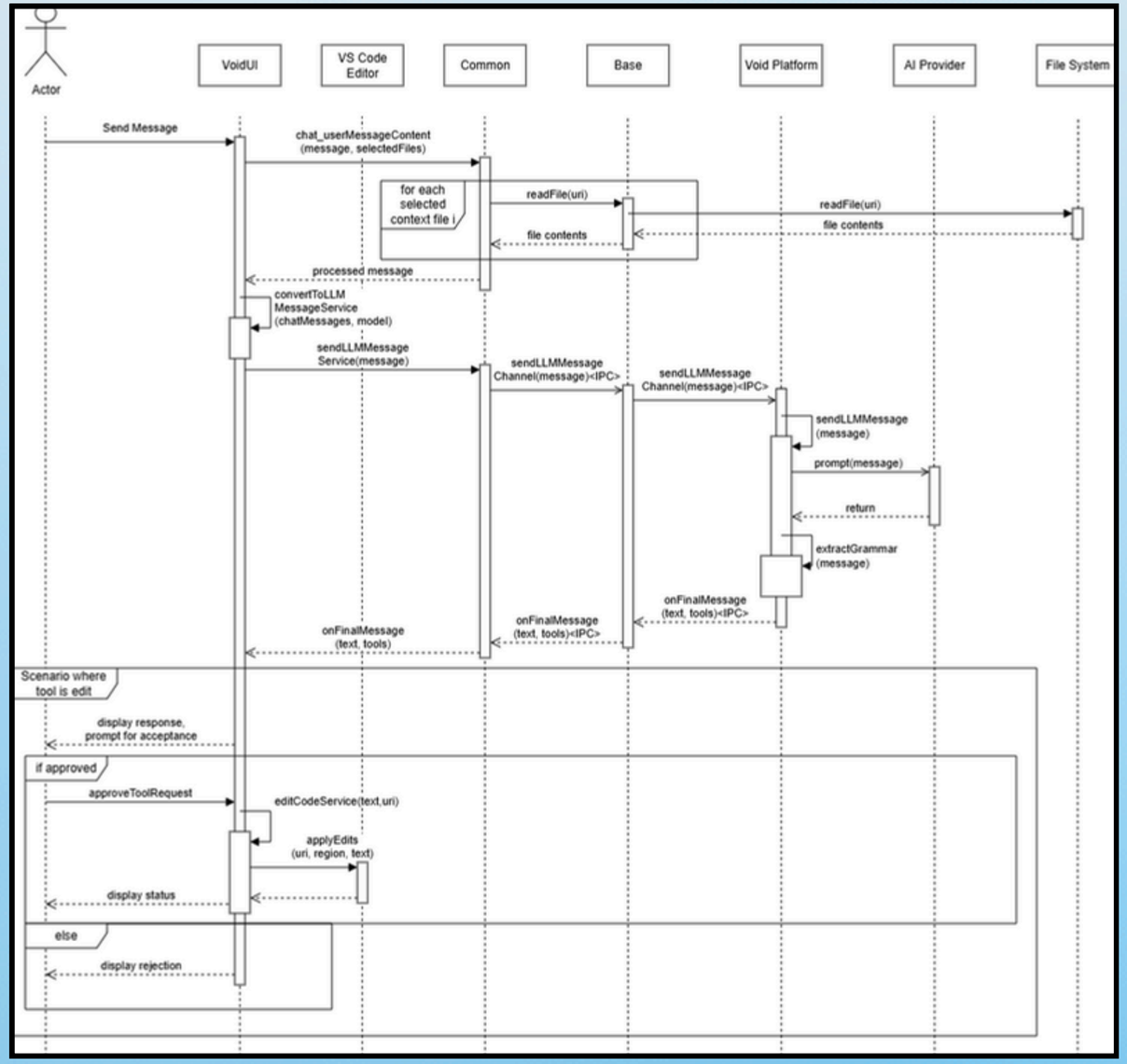
Sequence diagram - Use Case 1





Sequence diagram - Use Case 2





Lessons Learned

Concept vs. code: assumptions rarely hold

Broad → detailed view improved accuracy

VS Code is more connected than expected

Team coordination sped up analysis

Understand was key for visualizing real dependencies

AI Teammate

Used ChatGPT (GPT-5, Aug 2025) for code clarification & verification

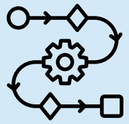
Helped interpret TypeScript files, dependencies, subsystem roles

Assigned structured, support-only tasks (no decision making)

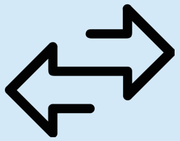
Followed strict cross-checking & peer-review for accuracy

≈ 15 % contribution - improved understanding & efficiency

Conclusion



Used Understand to analyze key subsystems and dependencies



Found major change: Model Service → Common (shared UI–Platform layer)

Design is modular, coordinated, and built for scalability

Extensions now central for AI features & third-party integration

Architecture remains Layered & Client–Server, with event-driven IPC

Thank You!