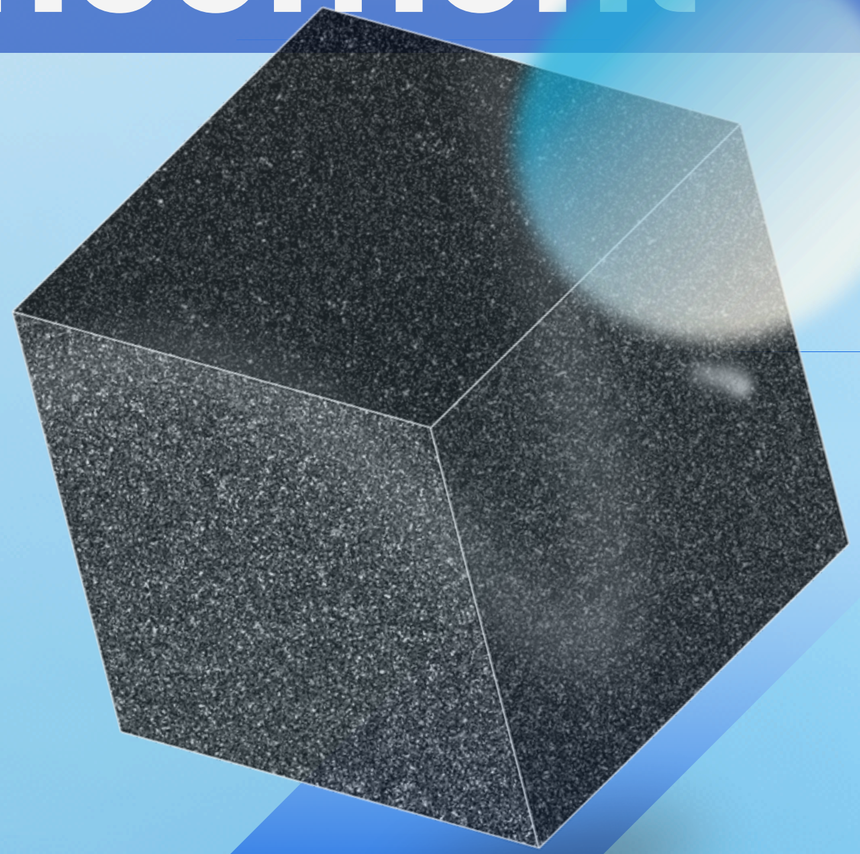
Architectural Enhancement

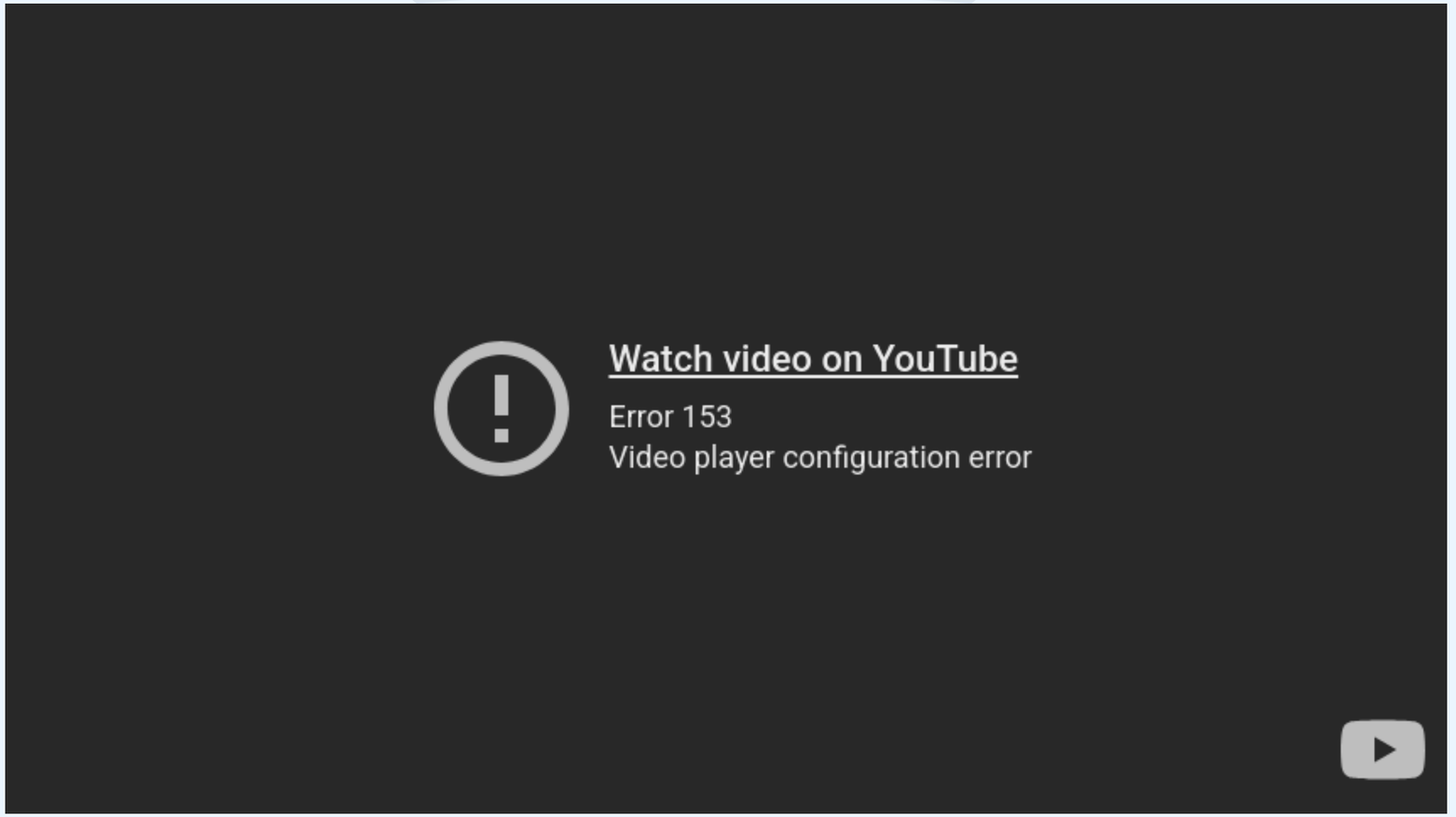
CISC322



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

https://youtu.be/cFKd78S6HL4

sandy

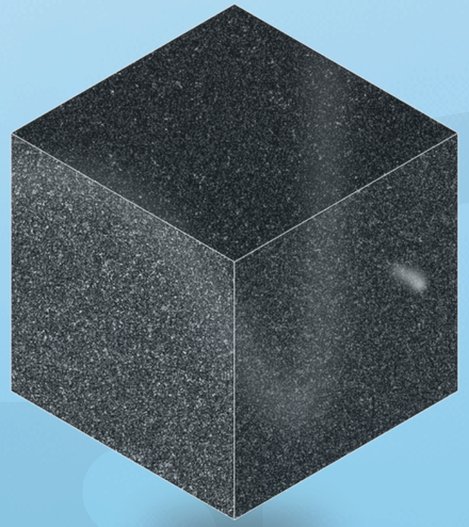


Agenda

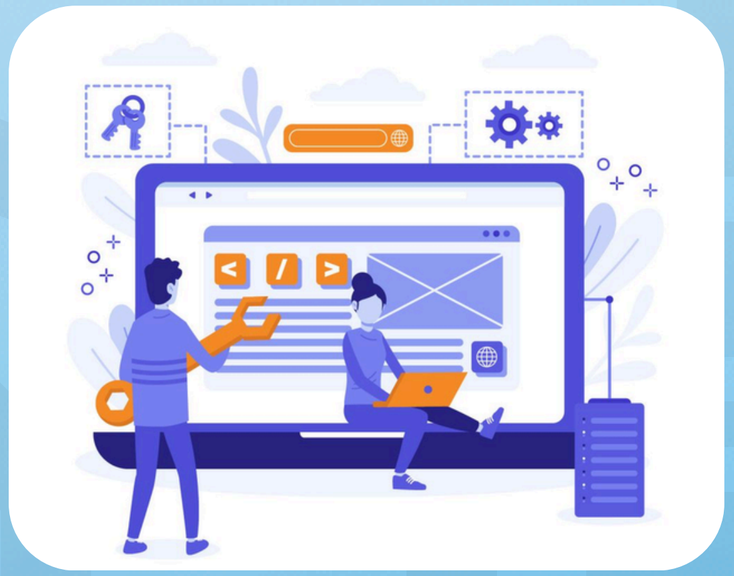
1. Motivation & Problem Definition
    a. (Why Void needs automatic context generation)
2. Architectural Alternatives & SAAM Analysis
    a. (Search vs. RAG + stakeholders + NFRs)
3. Subsystem Impacts & Updated Architecture
    a. (Concurrency, Platform/UI changes, diagrams)
4. Use Cases & Sequence Diagrams
    a. (Inline completion & chat interactions)
5. Risks, Testing Strategy & Architectural Constraints
6. Lessons Learned & AI Teammate Support

Why Void Needs Automatic Context Generation

- Void lacks automatic retrieval of workspace context
- AI responses often inaccurate or incomplete
- Competing editors (Cursor, Claude) already support this feature
- Commented-out context code shows an unfinished Void capability
- Enhancement improves accuracy, relevance, and user workflow



+



Two Architectural Approaches: Search vs RAG

- Search-based retrieval: grep/keyword scanning on demand
- Persistent local RAG: vector store of code embeddings
- Both can integrate with existing layered architecture
- Each affects Platform, UI, and Common components differently

Search

- Platform → grep → VS Code file read → return to Platform

RAG

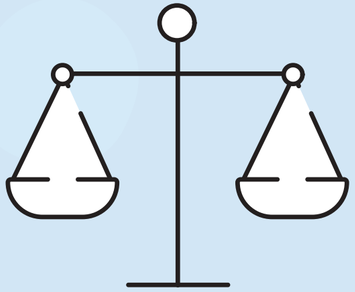
- Void Platform → Context Component
- Context Component → VS Code Platform

Stakeholders & Their Non-Functional Requirements

Stakeholders: developers/maintainers, end users

Users need: accuracy, speed, scalability

Maintainers need: maintainability, testability, evolvability



NFRs drive comparison between the two architectural options

SAAM Evaluation: Tradeoffs Between Search and RAG (Pros & Cons)

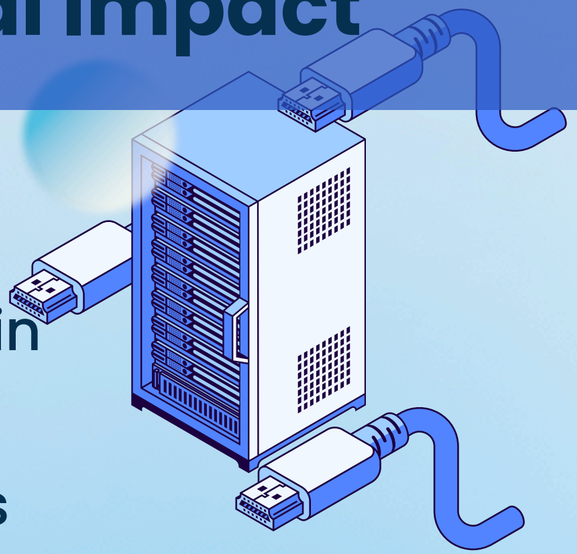
Search	RAG
✔ Higher accuracy	✔ Faster for large repos
✔ Easier to maintain	✔ More scalable
✔ No persistent subsystem	✘ Requires Vector DB + updates
✔ Fewer failure points	✘ Risk of stale embeddings

- SAAM clarifies impacts on NFRs for both alternatives

Why Search-Based Retrieval Is the Best Choice

- Search integrates cleanly with existing Void structure
- No persistent storage or complex new component required
- More maintainable for long-term contributors
- Best overall match for user and developer requirements

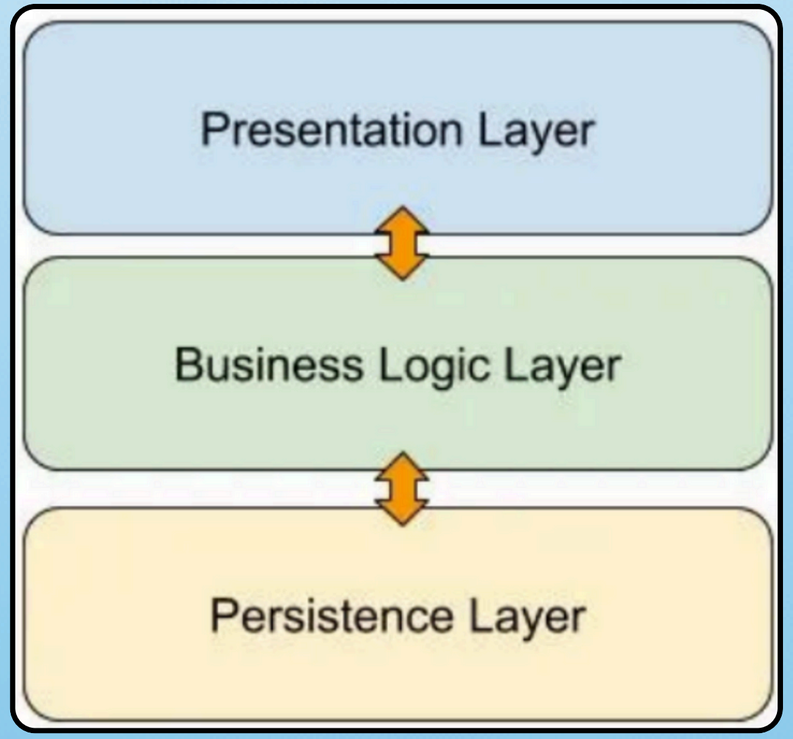
Team Workflow, Concurrency, and Architectural Impact

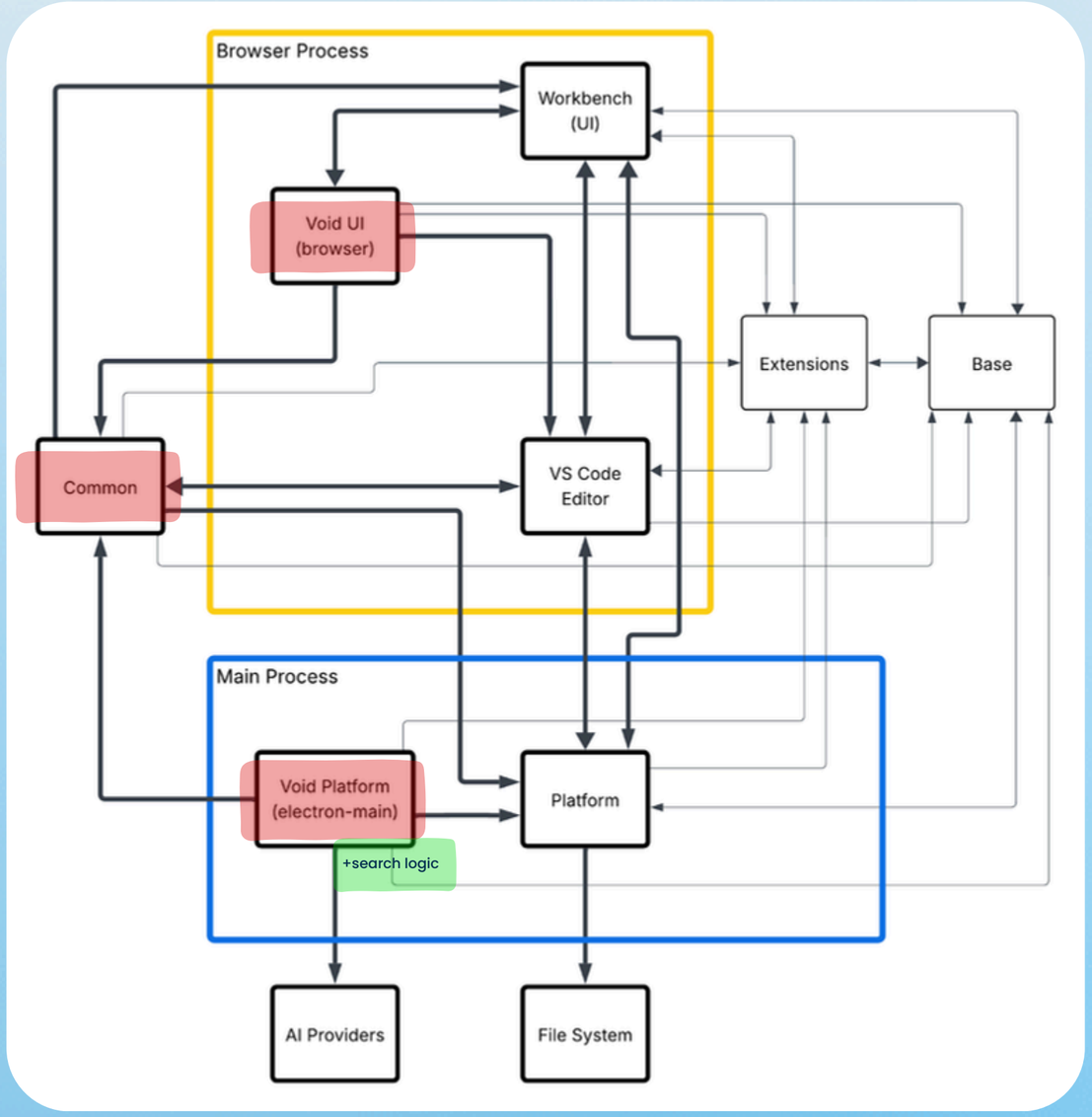


- Search-based approach easy to integrate and maintain
- Minimal long-term maintenance burden on developers
- Testing context-related behaviour may be more challenging
- LLM responses are inconsistent, making tests harder to reproduce
- Changes still require coordination across subsystems

Affected Subsystems & Architectural Style Considerations

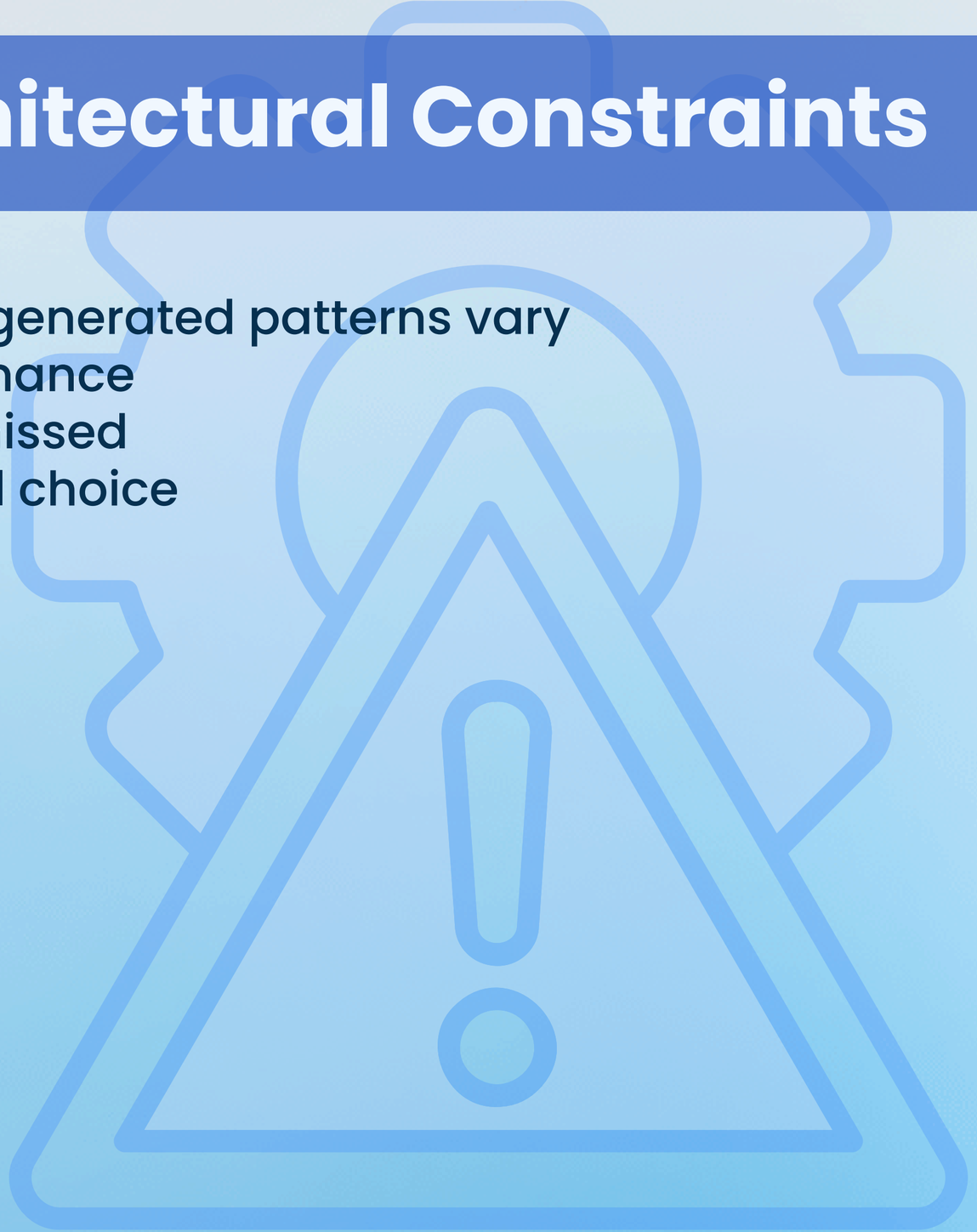
- Affects: Void Platform, Common, Void UI component
- Search logic added to Platform; formatting moved from UI
- RAG option adds an entire Context component
- Enhancement fits Void’s layered architectural style





Risks, Limitations, and Architectural Constraints

- Search risks: slower on large codebases; LLM-generated patterns vary
- RAG risks: stale embeddings, complex maintenance
- Persistent storage may desync if file-events missed
- Risk analysis informs sustainable architectural choice



Testing Strategy for Context Generation & System Interactions

Verify search-pattern generation for chat

Test grep timing + file-read accuracy

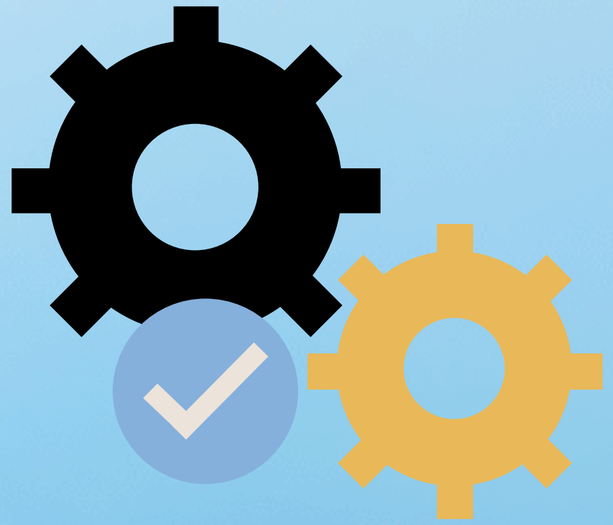
Check message-formatting consistency

Inline completion: measure response delays

Validate LLM responses with mock codebase

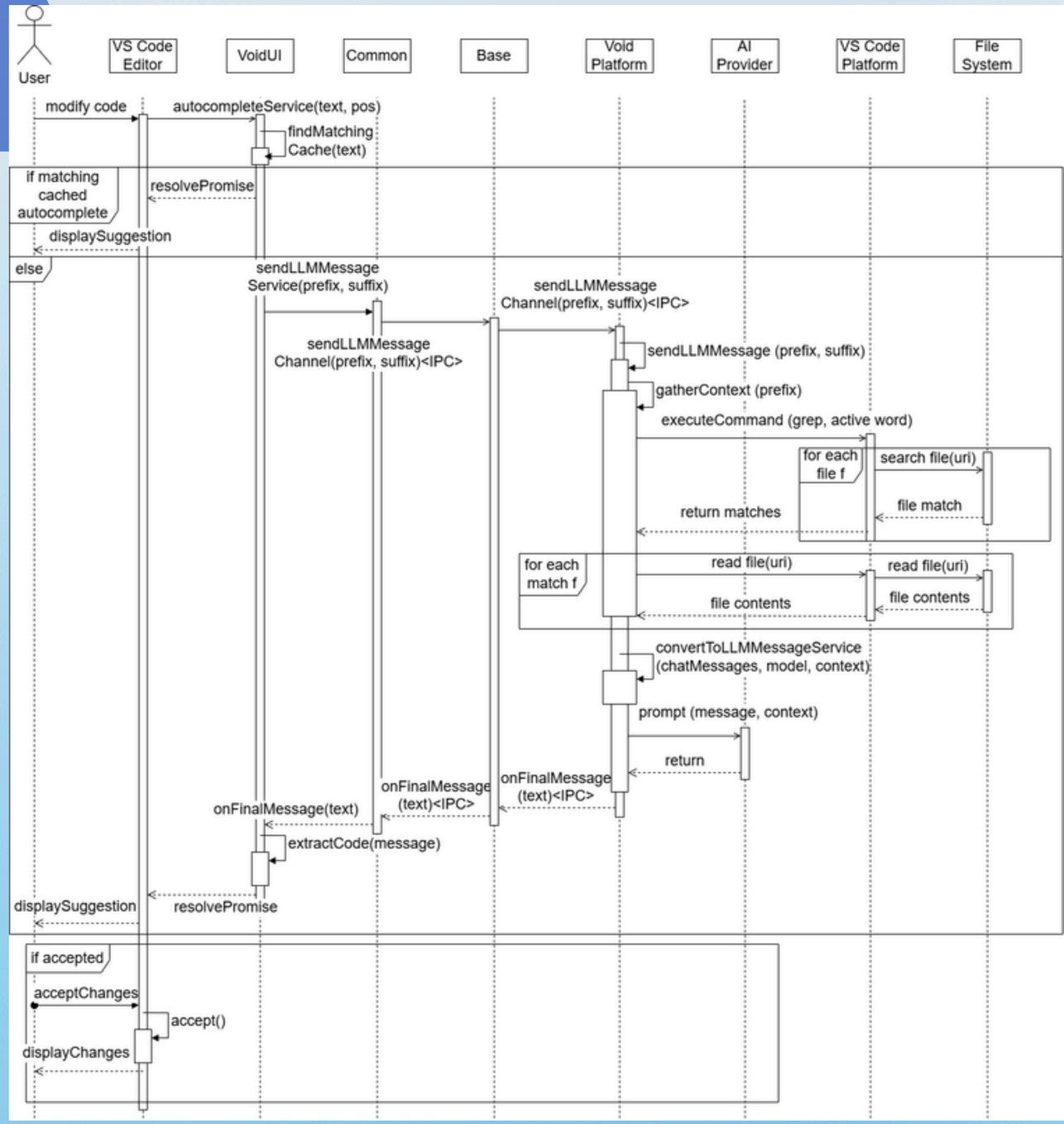
Ensure new IPC messages don't break Browser - Main flow

Run full end-to-end chat + autocomplete tests



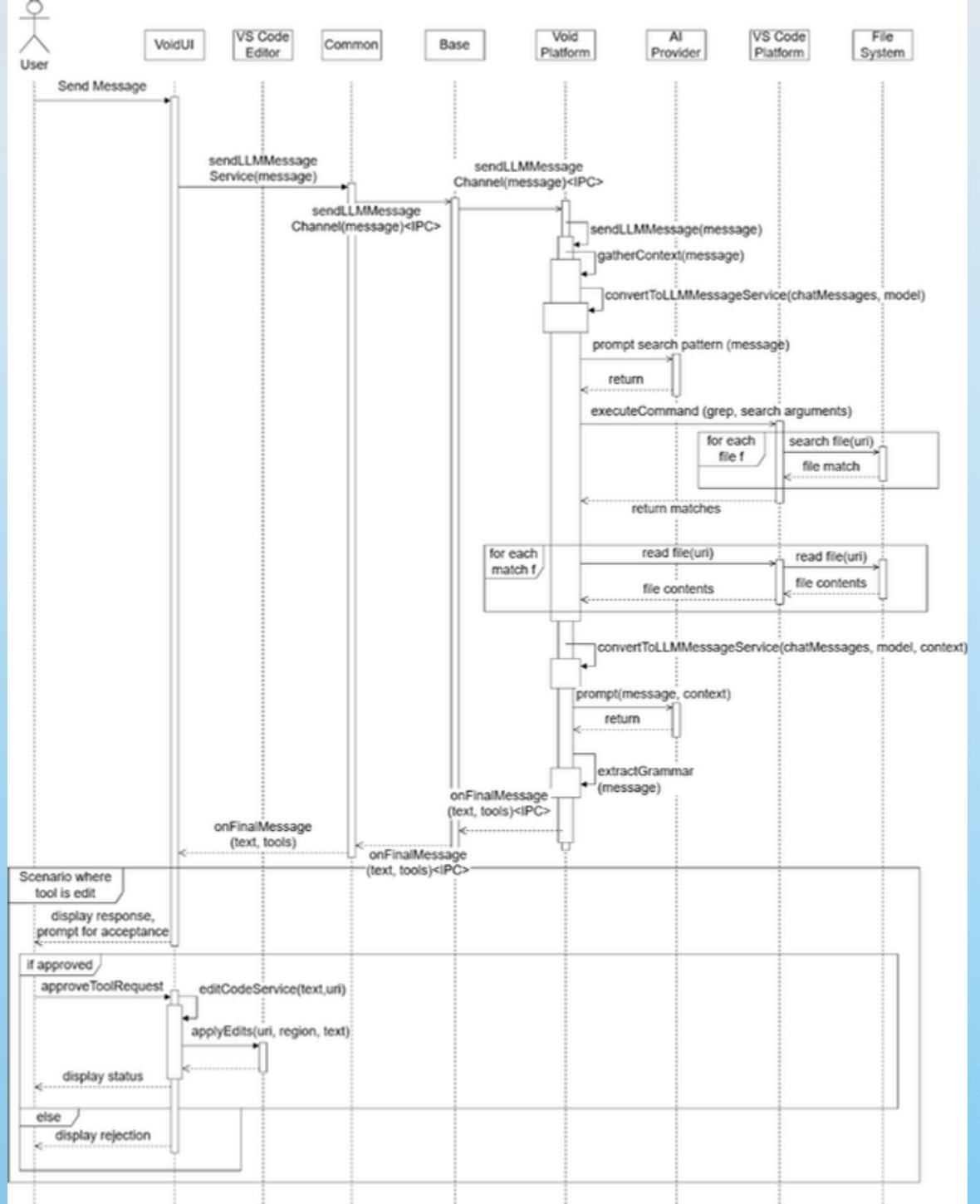
Sequence Diagram: Inline Completion (Use Case 1)

- VS Code triggers inlineSuggestion → UI handler
- UI checks for cached autocomplete
- UI calls sendLLMMessage → Common → Base → Platform
- Platform runs gatherContext (lexical search for active word)
- Reads matching files via VS Code Platform
- Bundles context using moved convertToLLMMessageService
- Platform prompts LLM and returns completion
- UI extracts result and applies suggestion



Sequence Diagram: Chat Interaction (Use Case 2)

- UI calls chat_userMessageContent → Common → Base → Platform
- Platform runs gatherContext
  - LLM generates search pattern
  - Grep finds relevant files
- Files appended using convertToLLMMessageService
- Platform prompts LLM → uses extractGrammar
- UI displays result via onFinalMessage
- User approves edits → VS Code Editor tentatively applies changes



Lessons Learned: Architecture, SAAM, & Collaboration

Designing enhancements requires deep architectural understandingc

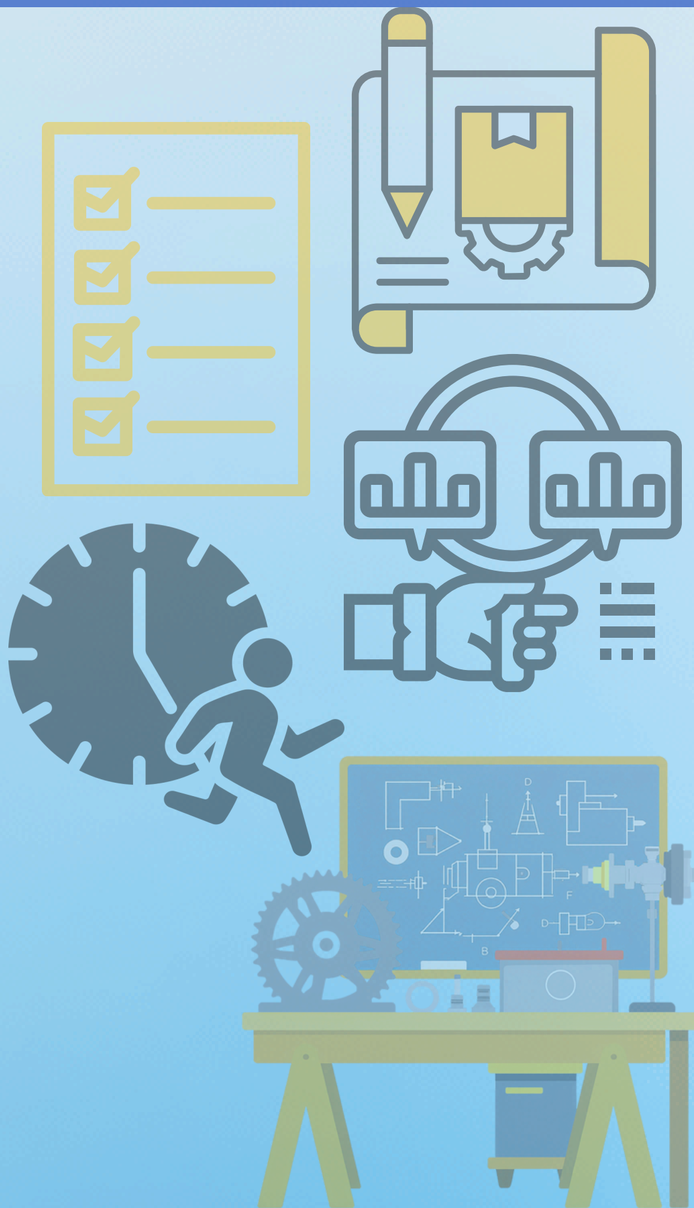
Small changes (e.g., moving one file) affect multiple layers

SAAM reveals tradeoffs more clearly than intuition

Collaboration essential across UI, Platform, Common groups

Limitations of Our Findings & Future Work

- LLM response variability makes behaviour harder to test
- NFR analysis was conceptual; no performance benchmarking
- Future work needed to validate real runtime behaviour
- SAAM comparison was conceptual, not empirically tested
- No prototype to measure real performance or accuracy



Architecture & Sequence Diagrams Supporting the Analysis

Figure 1: Current architecture

- Browser (yellow) VS Main (blue) processes
- Bold arrows = major dependencies
- context generation is manual & limited

Why enhancemnent fits: MCP already supports file access; automatic context improves accuracy

Search approach:

- On-demand grep search
- Move convertToLLMMessageService → Platform
- Add gatherContext.ts + update IPC channels

RAG approach:

- New context component (Vector DB + file-change events)

Figure 2: Updated architecture with new responsibilities & Context component

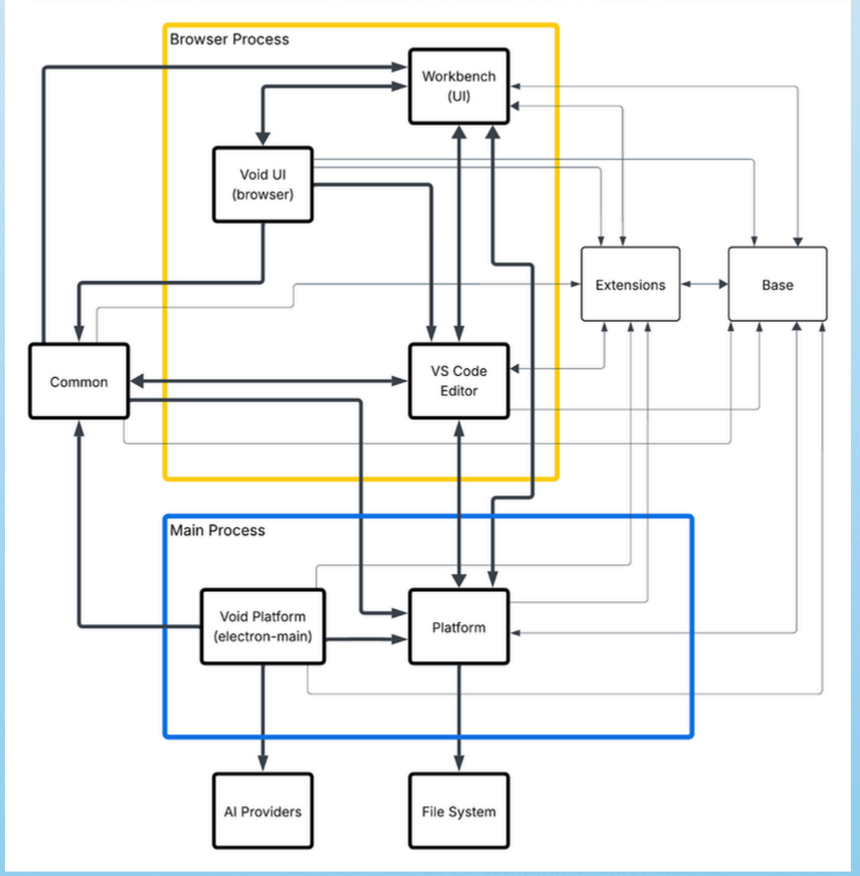


Figure 1

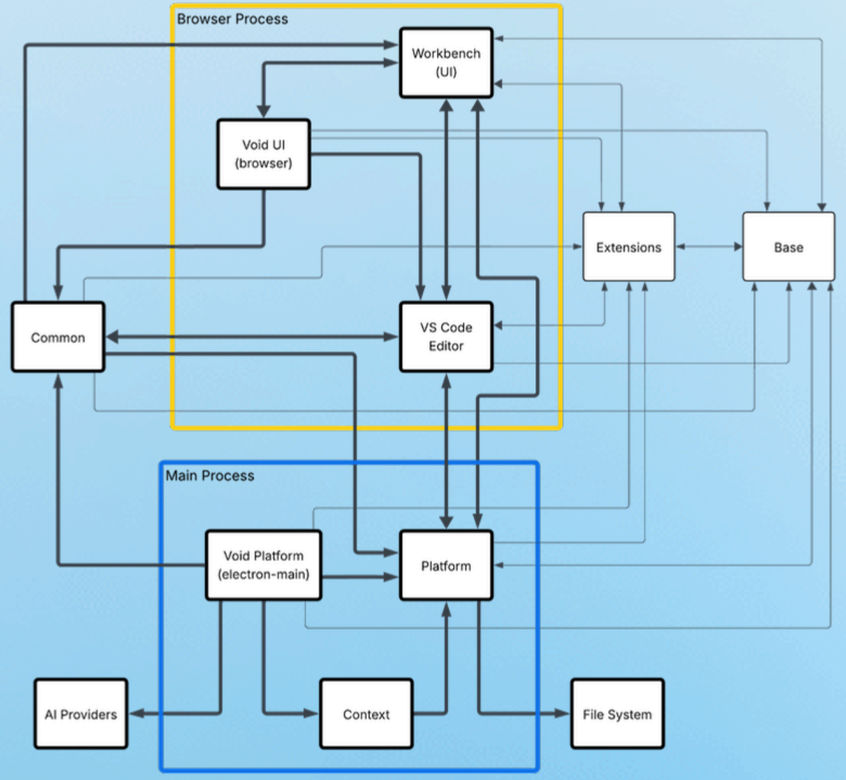


Figure 2

AI Teammate

Used chatGPT as a structured AI teammate

Assigned limited, well-defined tasks

- SAAM Scenario drafting
- NFR wording refinement
- Organizational support

Followed a clear prompting strategy

Kept AI output separate and isolated for review

Strict quality control:

- >= 2 team members reviewed ai outputs
- Cross-checked with architecture notes & external sources
- Rewrote unclear or inaccurate content

AI contributed ~15% (wording + scenario options)

All architecture, diagrams, SAAM decisions done manually

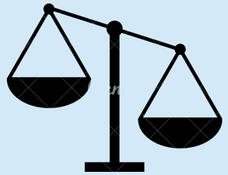
Ensured transparent and responsible human-AI collaboration

Conclusion

Clarified Void's concrete architecture & constraints

Identified missing automatic context generation

Integrates cleanly & improves accuracy without added complexity



Compared Search vs RAG using SAAM

Search-based approach chosen

Thank You!