GROUP 13

THE VOID MANUAL

Video presentation link: https://youtu.be/cPTUHstZqNs

https://the-void-manual.vercel.app/#

GROUP 13

TEAM MEMBERS

Aiden Ramezani
Leader

Divyanshi Agarwal
Presenter

Kanika Poonia
Presenter

Vedansh Bhatt

Bella Xu

Sameer Bandha

https://the-void-manual.vercel.app/#

GOAL AND SCOPE

Purpose

The principal aim of Assignment 2 is to transition from the conceptual architecture of the Void Editor (defined in A1) to its concrete, as-built implementation

Scope

This involves studying the correspondence between design intent and actual code structure, identifying discrepancies, and providing architectural insights.

https://the-void-manual.vercel.app/#

METHODOLOGY & TOOLS

Methodology

The concrete architecture was recovered using SciTools Understand, a static-analysis tool, to analyze code dependencies. The analysis focused on both:

- Top-level integration: How Void interacts with VS Code.
- Subsystem-level: The internal structure of the Void subsystem.

Steps Followed

- Load the pre-built .und project in Understand.
- Generate dependency graphs for all modules.
- Group related modules into subsystems.
- Perform detailed analysis of the Void subsystem.
- Compare results with A1 conceptual architecture, noting alignments and divergences.

https://the-void-manual.vercel.app/#

REPORT ORGANIZATION

- Section 1: Introduction and Process Overview
- Section 2: Top-Level Architecture Analysis
- Section 3: Void Subsystem Decomposition
- Section 4: Architectural Styles and Patterns
- Section 5: Reflexion Analysis (A1 vs A2)
- Section 6: Lessons Learned and Preparation for A3

https://the-void-manual.vercel.app/#

GROUP 13

CONCRETE ARCHITECTURE OVERVIEW

Purpose of Architecture Recovery

- Recovered from source code using SciTools Understand 5.1.
- Provides a code-based perspective of Void Editor's structure.

High-level structure:

- Layered, plugin-oriented ecosystem.
- Core runtime logic in src.
- Supported by build, scripts, test, and resources.

Observation:

- Electron → UI rendering & window management.
- VS Code Extension Host → command registration & dispatch.

Key purpose

- Identify top-level components and subsystems.
- Capture control & data dependencies.

https://the-void-manual.vercel.app/#

GROUP 13

TOP-LEVEL ARCHITECTURE (VS CODE + VOID)

Integration with VS Code

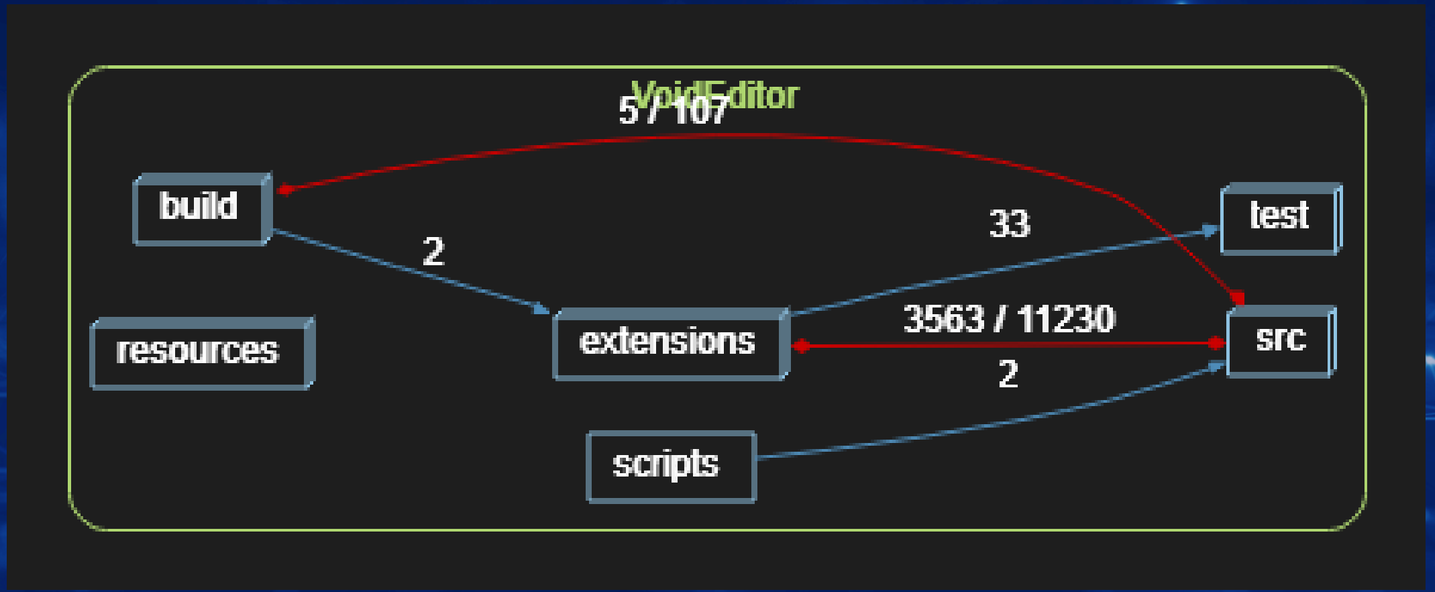
- Void Editor functions as a VS Code extension running in the Extension Host.
- src directory: core runtime logic (command handling, configuration, buffers, syntax parsing).
- extensions directory: integration layer connecting Void features to VS Code APIs.
- Relies on Electron (UI rendering & window management) and Node.js APIs (file I/O, process control).

https://the-void-manual.vercel.app/#

TOP-LEVEL ARCHITECTURE (VS CODE + VOID)

Top-Level Dependency Graph

- Boxes represent major directories / components: src, extensions, build, scripts, test, resources.
- Arrows indicate dependencies:
  - Solid = control (calls, command delegation)
  - Dashed = data (imports, configuration, file I/O)



https://the-void-manual.vercel.app/#

TOP-LEVEL ARCHITECTURE (VS CODE + VOID)

Observations

- Layered + plugin-oriented, multi-layered event-driven architecture.
- Partial circular coupling typical for extensible editors.
- Highlights higher interdependency than conceptual A1 model.

Control & Data Dependencies

- Unidirectional (blue arrows): build → extensions → scripts/test → src (toolchain & automation flow).
- Bidirectional (red arrows): build ↔ src, extensions ↔ src (mutual coupling; src provides & consumes VS Code services).

https://the-void-manual.vercel.app/#

SUBSYSTEM DESCRIPTIONS

Key Top-Level Subsystems

- src – Core runtime logic (command handling, configuration, services).
- extensions – Integrates Void features with VS Code Extension Host.
- build – Automates build, linting, and packaging processes.
- scripts – Utility scripts for testing, launching, and CI automation.
- test – Unit and integration tests validating src functionality.
- resources – Static assets and configuration files used at build/runtime.

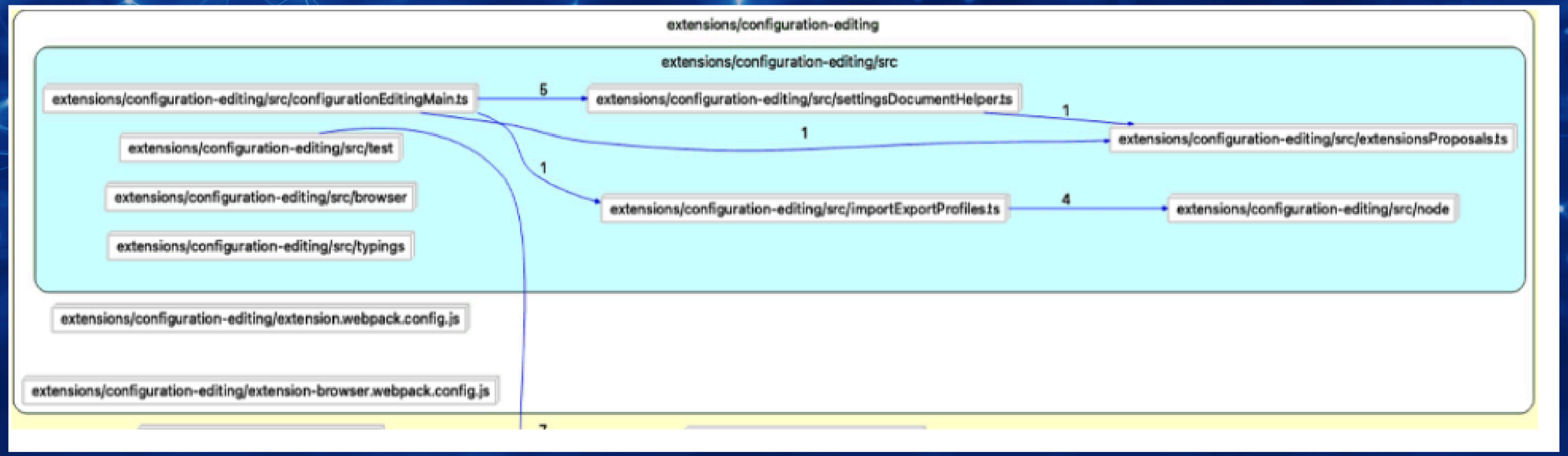
https://the-void-manual.vercel.app/#

GROUP 13

SUBSYSTEM STRUCTURE AND COMPONENTS

Key Modules

configurationEditingMain.ts
browser/
importExportProfiles.ts
node/
extensionsProposals.ts
settingsDocumentHelper.ts



https://the-void-manual.vercel.app/#

CONTROL & DATA FLOW

- Command Flow: User → extensions → src → Electron/VS Code APIs → UI
- Concurrency: Async file I/O, syntax checks, background tasks
- Control vs Data: Solid = commands, Dashed = data/config
- Insight: Extensions layer mediates commands, keeps modules decoupled

https://the-void-manual.vercel.app/#

GROUP 13

ARCHITECTURAL STYLES AND PATTERNS OBSERVED

- Layered: UI → Core → Persistence/API (modularity, separation of concerns)
- Plugin: Void as VS Code extension; commands registered via extensions
- Observer / Publish-Subscribe: Event-driven handling; event listeners in configurationEditingMain.ts
- MVC / Coordinator: configurationEditingMain.ts coordinates helpers & platform modules
- Insight: Ensures modularity, extensibility, maintainability; some higher-than-expected coupling

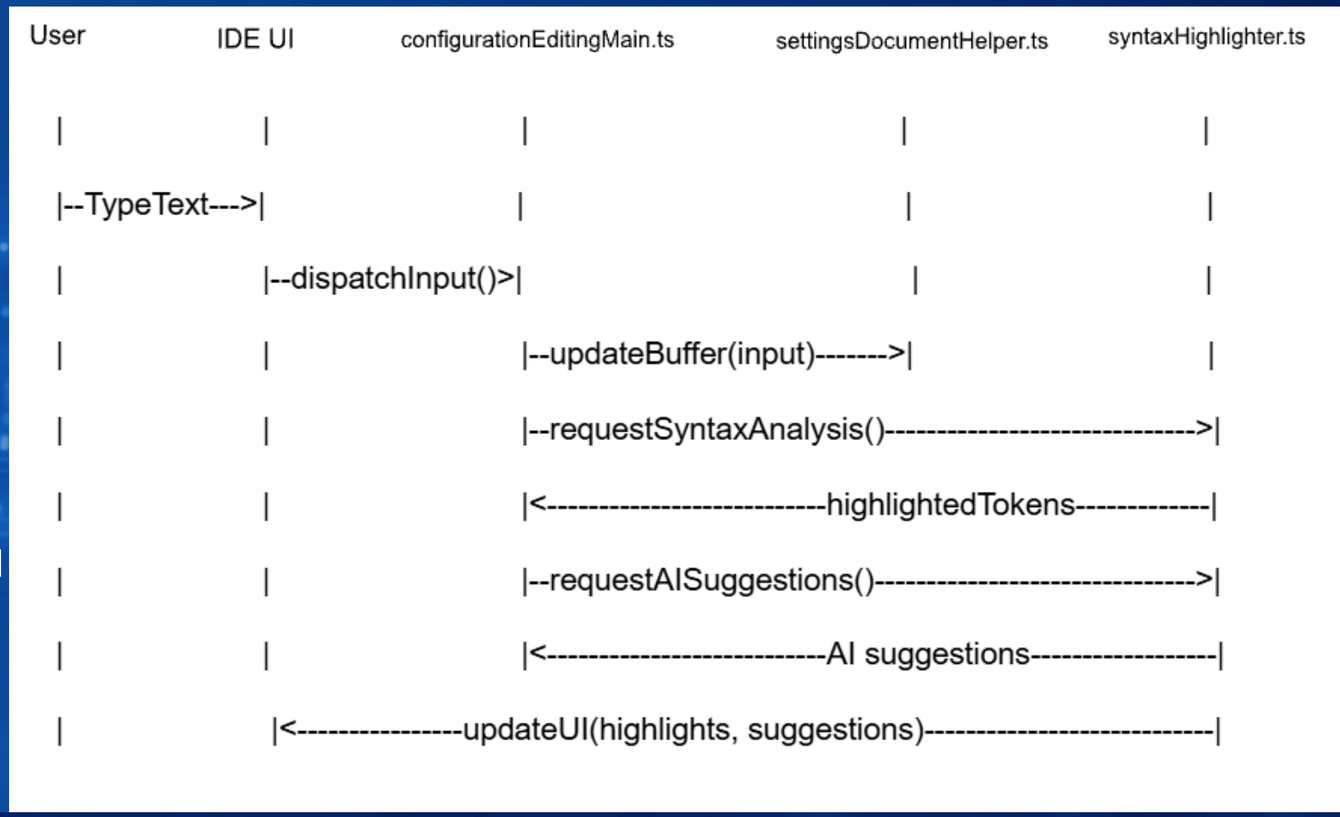
https://the-void-manual.vercel.app/#

GROUP 13

USE CASE WALKTHROUGH

Flow Summary

- User Input: Developer types → IDE UI
- Event Handling: UI →configurationEditingMain.ts
- Buffer Update: Coordinator → settingsDocumentHelper.ts
- Syntax Analysis: settingsDocumentHelper.ts → syntaxHighlighter.ts
- AI Suggestions: Coordinator → aiSuggestionEngine.ts
- UI Update: Coordinator updates IDE UI with highlights & AI suggestions



https://the-void-manual.vercel.app/#

MAPPING CONCEPTUAL COMPONENTS TO CODE

- Start with A1 components: Editing Core, Syntax Service, Plugin Framework, Persistence Layer, Configuration
- Identify responsibilities for each component
- Use Understand to locate code matching responsibilities (keywords: editor, tokens, files, storage, configuration, extension)

Map directories to components:

- src/vs/editor → Editing Core
- src/vs/editor/common/languages, tokens → Syntax Service
- extensions → Plugin Framework
- src/vs/platform/files, storage → Persistence Layer
- src/vs/platform/configuration → Configuration
- Classify each mapping: Match / Partial Match / Divergence

GROUP 13

HIGH-LEVEL DIVERGENCE

- Void as a VS Code extension
- Rationale: A1 modeled Void as a standalone editor; actual implementation plugs into VS Code extension host.

- Core Engine & Platform Services partially reused
- Rationale: A1 assigned dedicated core/persistence layers to Void; implementation uses VS Code’s existing services.

https://the-void-manual.vercel.app/#

GROUP 13

VOID SUBSYSTEM DIVERGENCES

- Plugin Framework → Partial Match
- Rationale: Coupled with VS Code extension host instead of standalone plugin layer.

- Persistence Layer → Divergence
- Rationale: Shared VS Code file/storage APIs used instead of a dedicated Void persistence module.

- Editing Core → Partial Match
- Rationale: Combines editing, view wiring, and service integration, unlike the separated conceptual components in A1.

https://the-void-manual.vercel.app/#

EDITING CORE PARTIAL MATCH

- Editing Core: Partial Match, combines editing, view logic, and service wiring.
- Plugin Framework: Partial match, tightly integrated with VS Code extension host.
- Persistence Layer: Divergence, uses shared VS Code platform services instead of Void-only layer.
- Syntax Service & Configuration: Strong match with A1 design.
- Insight: Implementation favors reuse of VS Code infrastructure, showing architectural drift.
- Evidence: Git logs confirm most file I/O and config work occurs in VS Code modules.
- Future: A3 may add façade layers or refactor to separate Void from host platform.

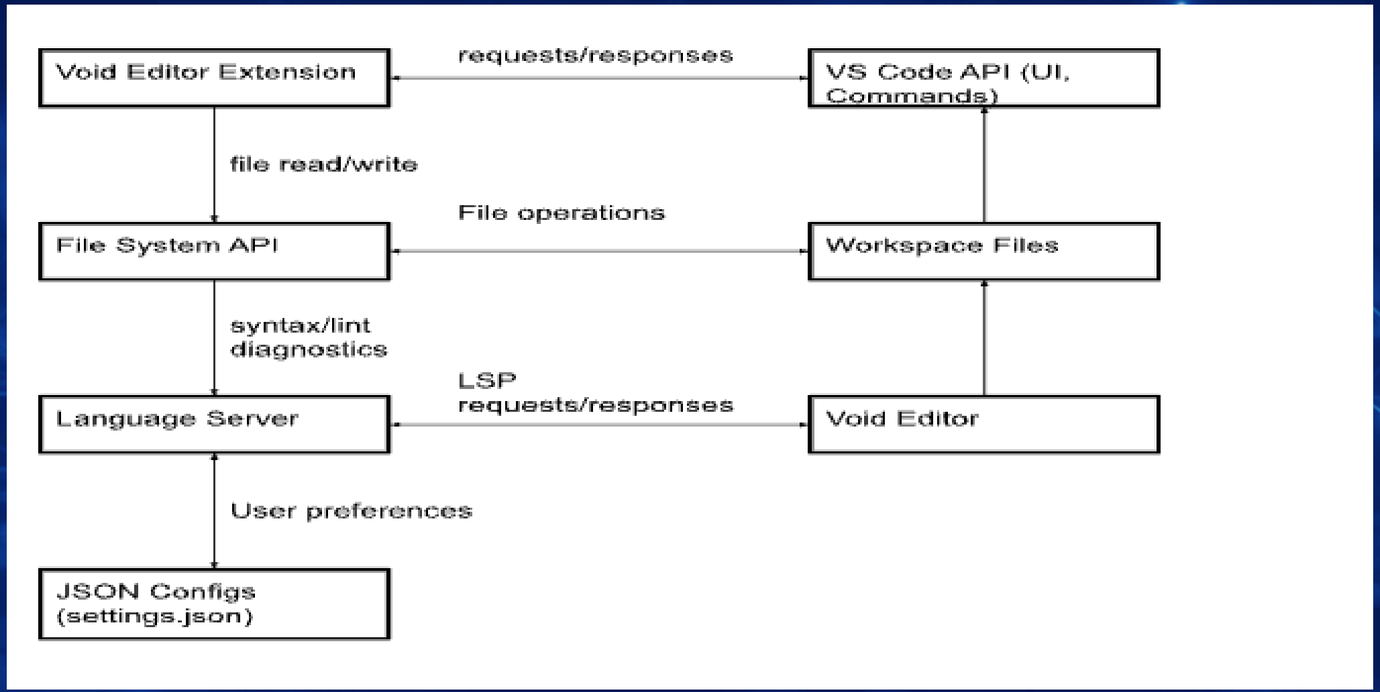
https://the-void-manual.vercel.app/#

GROUP 13

EXTERNAL INTERFACES

Key Integrations:

- VS Code Extension API: activation, commands, UI events.
- File System APIs: async file read/write, cross-platform.
- Language Servers (LSP): syntax, linting, suggestions.
- User Config (JSON): settings, theme, formatting.



https://the-void-manual.vercel.app/#

CONCLUSIONS AND LESSONS LEARNED

Architecture Insights:

- Void = VS Code extension, layered & event-driven.
- Coordinator (configurationEditingMain.ts) manages syntax, AI, and config helpers.
- Matches: Syntax Service & Configuration;
- Partial: Editing Core & Plugin Framework;
- Divergence: Persistence uses VS Code services.

https://the-void-manual.vercel.app/#

CONCLUSIONS AND LESSONS LEARNED

Lessons:

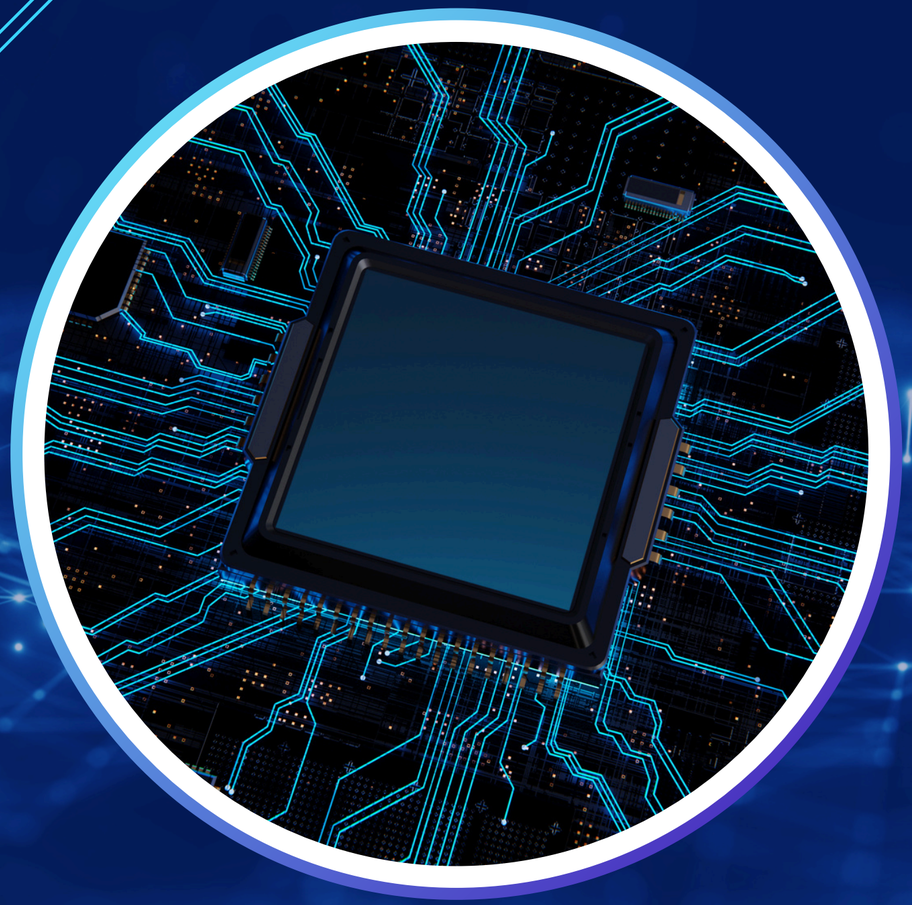
- Reuse platform services → reduces duplication but increases coupling.
- Layered, modular design → simplifies async handling & maintenance.
- Reflexion analysis identifies gaps vs A1.

A3 Implications:

- Enhance coordinator & helpers.
- Maintain modularity for async flows.
- Consider façades to manage VS Code coupling.

https://the-void-manual.vercel.app/#

GROUP 13



THANK YOU

https://the-void-manual.vercel.app/#