Explains design and software architecture

Subhash Malaviya (2023CS10749)
Soumodeep Chatterjee (2023CS50299)
Shashank Kumar (2023CS10020)

April 2025

Team Members

SUBHASH MALAVIYA (2023CS10749)
SOUMODEEP CHATTERJEE (2023CS50299)
SHASHANK KUMAR (2023CS10020)

GitHub Repository

The complete source code and documentation are available at:
`https://github.com/Pacify-14/Rust-Lab-COP-290`

1 Why Proposed Extension Could Not Be Fully Implemented

While the core Vim-like terminal and GUI-based spreadsheet functionality was successfully implemented, several proposed extensions could not be realized due to time and complexity constraints. In particular:

- **Chart and Data Visualization** features like statistical plots, time series, and correlation graphs were not implemented.
- **Custom mathematical functions**, hierarchical expression parsing, and dynamic formula libraries were left out.
- **Smart data management features** such as filtering by Boolean conditions or dynamic query execution were not developed.
- **Performance optimizations** like parallel formula evaluation and lazy evaluation were planned but not implemented.
- **Automated testing frameworks** including property-based testing were proposed but not incorporated.

2 Could We Implement Extra Extensions Over and Above the Proposal?

Yes, some enhancements were implemented that went beyond the originally proposed scope:

- A fully-functional **GUI-based interface** using the `egui` framework in addition to the terminal interface.
- **Visual selection support** in the GUI and TUI for range operations like copying and pasting ranges.
- **File format flexibility**, including support for CSV, TSV, and a custom SS format with formula preservation.

3 Primary Data Structures Used

- `Vec<Vec<cell>>` – A 2D vector used as the spreadsheet grid.
- `cell` struct – Stores individual cell state including value, formula, and error flag.
- `EditorState` – Maintains UI mode, cursor position, selection range, and buffer states.
- `ClipboardContent` enum – Encapsulates cell, row, column, and range data for copy-paste operations.
- `DAGNode` and `Node` – Represent the dependency graph for formula evaluation.

4 Interfaces Between Software Modules

The architecture is modularized as follows:

- **core logic** (formula parsing, evaluation) in `main.rs`
- **editor state and modes** in `editor.rs`
- **terminal interface** in `ui.rs`
- **GUI interface** in `egui_ui.rs`
- **command execution** in `commands.rs`

5 Approaches for Encapsulation

Encapsulation was maintained by:

- Defining stateful structs like `EditorState` with controlled access to internal fields.
- Using enums like `Mode` and `ClipboardContent` for explicit mode/state management.
- Separating interface logic from core logic and data manipulation routines.

6 Justification of Design

This design enables:

- **Robust terminal and GUI interaction** via mode-specific input handling.
- **High extensibility**, allowing new commands and UI features to be integrated with minimal change.
- **User efficiency**, leveraging familiar Vim-like controls.
- **Clean separation of concerns**, which improves code maintainability and testability.

7 Modifications to the Initial Design

The following deviations were made from the original proposal:

- Inclusion of the `egui`-based graphical interface, which was not originally planned.
- Reduction of scope in terms of visualization and advanced querying features to prioritize core spreadsheet stability.
- Extended copy-paste functionality to support entire rows and columns, beyond individual cells.

8 Code Architecture and Workflow

8.1 Overview of Functionality

The Vim-like spreadsheet is a terminal and GUI-based editor developed in Rust. It supports multiple editing modes, efficient keyboard navigation, expression evaluation, and formula-based cell dependencies. The interface mimics `vim` behavior, offering modes like `Normal`, `Insert`, `Visual`, and `Command` for power users.

8.2 Core Module Responsibilities

- `main.rs`: Entry point and core logic. It handles formula parsing, topological evaluation using a dependency graph (DAG), scroll commands, and command-line input.
- `editor.rs`: Manages editor state and mode transitions. It defines the `EditorState` struct and modes (Insert, Normal, Visual, Command). Also handles edit buffer and clipboard.
- `ui.rs`: Implements the terminal-based UI using the `crossterm` crate. It renders the spreadsheet grid, cursor, and status bar, and processes keyboard input based on the current mode.
- `egui_ui.rs`: Provides an optional GUI interface using `eframe` and `egui`. It supports mouse and keyboard events, colored mode banners, and cell-based interactions.
- `commands.rs`: Parses and executes `:commands` in Command mode, including `:w`, `:q`, `:e`, search/replace, and batch operations.
- `mod.rs`: Central reexport module that links together the `commands`, `editor`, `ui`, and `egui_ui` modules under the `vim_mode` namespace.

8.3 Data Flow and Evaluation

The spreadsheet grid is represented as a `Vec<Vec<cell>>`, where each `cell` stores:

- An optional formula string
- An evaluated integer value
- An error flag (1 if invalid, 0 otherwise)

When a formula is entered, it is parsed and stored in the cell. During evaluation, a dependency graph (`Vec<DAGNode>`) is built. A topological sort ensures cells are evaluated in correct order, respecting dependencies and propagating errors.

Supported formulas include:

- Arithmetic: `A1 + B2`, `3 * C3`
- Aggregates: `SUM(A1:A5)`, `AVG(B1:B3)`
- Functions: `SLEEP(3)`, `SLEEP(B2)`

8.4 Interaction Flow

1. **Startup:** The program initializes the grid and starts in Normal mode.
2. **Navigation:** Users move the cursor using `h, j, k, l`.
3. **Editing:** Pressing `i` enters Insert mode, where formulas or values can be typed.
4. **Command execution:** Pressing `:` enters Command mode. Commands like `:wq`, `:e filename`, or `:A1` are parsed and executed.
5. **Visual selection:** Pressing `v` activates Visual mode to select ranges for copy/paste.
6. **Evaluation:** After edit or file load, the sheet is re-evaluated using a DAG-based topological traversal.

8.5 Formula Evaluation Example

Given:

```
A1 = 5
A2 = A1 + 3
A3 = SUM(A1:A2)
```

The evaluation order will be:

- `A1`: directly assigned value 5
- `A2`: depends on `A1`, evaluates to 8
- `A3`: range sum of A1 and A2 = 13

8.6 Clipboard Functionality

Copying is done using `y`, pasting with `p`. Clipboard supports:

- Single cells
- Rows and columns
- Arbitrary rectangular ranges

These are stored using the `ClipboardContent` enum.

8.7 Error Handling

`ERR` is displayed in cells with:

- Invalid formulas
- Circular dependencies
- References to error cells

8.8 Search and Replace

Searches can be performed using `/pattern` and `?pattern`. Matches are navigated using `n/N`. Replace is done using `:s/old/new/g`.

8.9 Extensibility

The modular design makes it easy to:

- Add new commands in `commands.rs`
- Extend formula parsing in `main.rs`
- Introduce new modes via `editor.rs`
- Improve visualization using `egui_ui.rs`