

# A Trust-Centered Rule-Based macOS Filesystem Organizer

Kalash Kumari Thakur, Soniya Malviya, Aryan Soni

**Abstract**—This paper presents *Clippy*, a macOS filesystem organization tool that prioritizes correctness and user trust over aggressive automation. The system organizes files using declarative rules, generates explicit action plans, and executes only user-approved operations. All filesystem mutations are explainable, reversible, and designed to avoid surprising behavior. The architecture follows a strict separation of concerns between domain models, planning logic, and user interface components, enforced through a formal system contract and safety-oriented failure philosophy.

**Index Terms**—Filesystem Organization, Rule-Based Systems, Trust-Centered Design, macOS, Software Architecture, Reversible Systems

## I. INTRODUCTION

Personal computers accumulate heterogeneous files across downloads, documents, and project directories, making manual organization time-consuming and error-prone. Conventional file organizers often rely on aggressive automation that silently move or delete files, eroding user trust when actions are not transparent or reversible.

Clippy addresses this limitation by adopting a trust-centered design philosophy: every filesystem change must be explainable, reversible, and unsurprising. Instead of executing rules automatically, Clippy generates an explicit action plan that users review and approve. Filesystem writes occur only after user consent, and all actions are logged to support safe undo operations.

The main contributions of this work are:

- 1) A modular architecture separating domain models, engines, and UI.
- 2) A formal system contract encoding safety and trust invariants.
- 3) A macOS implementation leveraging Swift, SwiftUI, and FSEvents.

## II. RELATED WORK

Traditional file managers provide manual operations such as move, copy, and delete, but do not support high-level declarative rules or batch planning mechanisms. Third-party cleanup utilities often execute operations directly in response to filesystem events, reducing transparency and weakening undo guarantees.

Clippy differs by treating automation as advisory rather than autonomous. Filesystem events are observed to suggest staleness but never trigger execution automatically. This aligns with safety-critical software design principles where explicit confirmation and robust logging are prioritized.

## III. SYSTEM DESIGN

### A. Core Philosophy

Clippy follows four guiding principles:

- **Explainable:** Every action and skip includes a human-readable reason.
- **Reversible:** All changes are safely undoable via execution logs.
- **Unsurprising:** No hidden or automatic execution.
- **Conservative:** In ambiguous cases, operations are skipped with explanation.

### B. Architecture Overview

The system consists of three layers:

1) **ClippyCore (Domain Library)** Defines immutable domain models such as `FileDescriptor`, `Rule`, `Condition`, `Outcome`, and `ActionPlan`. These models are pure and independent of the filesystem.

2) **ClippyEngine (Logic Library)** Implements functional components including:

- **FileScanner** (read-only scanning)
- **Planner** (rule evaluation and plan generation)
- **ExecutionEngine** (applies approved plans)
- **UndoEngine** (reverses safe operations)
- **FileSystemObserver** (FSEvents-based monitoring)

3) **Clippy Application Layer** Built with SwiftUI, integrates navigation views, organize view, rules view, history manager, and search functionality.

## IV. MODULES AND RESPONSIBILITIES

### A. ClippyCore

ClippyCore models filesystem entities as immutable values. The `FileDescriptor` captures metadata including path, timestamps, and file type. Declarative rules combine conditions (extensions, sizes, dates) with outcomes (move, rename, trash, skip).

Pure modeling ensures deterministic planning without direct filesystem interaction.

### B. ClippyEngine

**FileScanner** enumerates directories and produces file descriptors.

**Planner** evaluates rules and produces an `ActionPlan`. Conflicting rules result in conservative skipping.

**ExecutionEngine** applies approved plans while:

- Preventing overwrites

- Creating parent directories when necessary
  - Logging failures without aborting execution
- UndoEngine** reverses successful operations safely and skips unsafe restorations.

## V. FORMAL SYSTEM CONTRACT

The formal contract enforces invariants:

- 1) No Unplanned Mutation
- 2) Observation Is Non-Causal
- 3) Reversibility Over Destruction
- 4) Conservative Undo
- 5) Mandatory Explanation
- 6) Stale Is Acceptable, Wrong Is Not

Subsystem boundaries ensure scanners read but do not write, planners decide but do not mutate, executors write but do not independently decide, and observers only monitor.

## VI. WORKFLOW

The system operates as follows:

- 1) User selects folder.
- 2) FileScanner generates file descriptors.
- 3) User enables rules.
- 4) Planner generates ActionPlan.
- 5) User reviews plan.
- 6) ExecutionEngine applies approved operations.
- 7) UndoEngine allows safe rollback.

## VII. IMPLEMENTATION

Clippy targets macOS 13.0+, implemented in Swift 5.9+ using SwiftUI and Swift Package Manager. The project separates source directories into core, engine, and UI layers. Passive filesystem observation is implemented using FSEvents.

The project repository is available at:

<https://github.com/Aryan2vb/Clippy>

## VIII. CONCLUSION AND FUTURE WORK

Clippy demonstrates that filesystem organization tools can prioritize trust, safety, and explainability without sacrificing usability. The strict separation of concerns and formal contract prevent unintended side effects and irreversible data loss.

Future work includes:

- Expanded rule templates
- Advanced visualization of action plans
- Analytics on execution history
- Additional condition and outcome types

## ACKNOWLEDGMENT

This project was developed by Kalash Kumari Thakur, Soniya Malviya, and Aryan Soni as an educational macOS application exploring trustworthy automation for filesystem organization.

## REFERENCES

- [1] Apple Inc., “The Swift Programming Language,” 2023.
- [2] Apple Inc., “SwiftUI Documentation,” 2023.
- [3] Apple Inc., “File System Events Programming Guide,” 2023.