WHITEPAPER · V2.0 · 2026

Mirror Box Orchestrator

Auditable, multi-vendor AI software engineering for any codebase

A new architecture for AI-assisted software development — one that treats vendor monoculture as a bug, human oversight as the design, and auditability as a first-class requirement.

Serious Engineering March 2026 Architecture Whitepaper

CONTENTS

01 — THE PROBLEM

AI coding tools have a structural blind spot

The current generation of AI coding agents — from autocomplete copilots to autonomous coding assistants — share a common architecture: a single model family handles the entire workflow. The same vendor plans the change, writes the code, and reviews its own work.

This creates a correlated failure mode that the industry has largely ignored. When a model family has a blind spot — and every model family does — that blind spot propagates unchecked through every stage of the pipeline. There is no external check, no independent second opinion, no structural mechanism to catch errors that the primary model cannot see in itself.

> "*A plan produced by one model family, reviewed by the same model family, catches only the errors that model family is capable of catching. Correlated blind spots are the primary failure mode of single-vendor AI pipelines.*"

Beyond the quality problem, there is a transparency problem. Most AI coding tools operate as black boxes. They produce output; you accept or reject it. The reasoning behind a decision — why this approach, why this file, why this change — is opaque. When something goes wrong, there is no audit trail to learn from.

And there is a cost problem. Tools that lack structural understanding of a codebase apply the same heavyweight process to trivial changes as they do to complex ones. Changing a config value should not cost the same as refactoring a core module. Without codebase intelligence, every task looks the same.

These three problems — correlated errors, opacity, and indiscriminate cost — are not accidents. They are consequences of architectural choices. The Mirror Box Orchestrator makes different ones.

---

02 — WHAT IT IS

A transparent, auditable AI software engineering system

The Mirror Box Orchestrator (MBO) is a coordination layer for AI-assisted software development. It orchestrates multiple AI models — from different vendors, running locally or in the cloud — to plan, independently review, and execute code changes on any repository.

It is not an autocomplete tool. It is not an autonomous agent. It is a structured pipeline with human oversight at every decision point that matters, full auditability of every action taken, and deliberate use of AI vendor diversity to improve output quality.

3+	$0.006	0	100%
VENDOR FAMILIES PER TASK	TARGET AVG. COST PER TASK	COMMITS WITHOUT HUMAN APPROVAL	PIPELINE DECISIONS LOGGED

MBO works on any codebase. It does not require a specific language, framework, or architecture. It onboards itself to a project through a structured interview and codebase scan, building a structural map that informs every subsequent decision. Legacy PHP site from 2009. Greenfield TypeScript monorepo. Mobile app. Microservices. It is codebase-agnostic by design.

It uses the AI sessions you already have. If you are authenticated with Claude, Gemini, or OpenAI's CLI tools, the orchestrator uses those sessions directly — no new accounts, no new API keys, no new authentication layer. Local models are supported for tasks where speed and cost matter more than frontier capability.

---

03 — CORE PRINCIPLES

Eight commitments that govern every decision

Every architectural choice in MBO flows from a small set of non-negotiable principles. When two approaches conflict, the one that better satisfies these principles wins.

P-01

Human approval gates every commit

No model has uncontrolled write access to repository history. Nothing commits without a human typing "go." This gate cannot be bypassed by any error condition, timeout, or model output.

P-02

Architecture is explicit, never inferred

The system maintains a living structural map of every codebase it works on. Models receive structured context derived from that map — not raw file dumps. What the system knows is always inspectable.

P-03

Every decision is reproducible

The event log is append-only. Every pipeline run can be fully reconstructed from its event history. Nothing happens off the record.

P-04

Non-LLM verification runs first

Compilation, static analysis, linting, and test execution run automatically. Human judgment is reserved for decisions that cannot be automated.

P-05

Failure defaults to safe rollback

Every execution creates a reversible branch. Failed builds revert automatically. The system never leaves a repository in an inconsistent state.

P-06

Independent derivation before comparison

No agent sees another agent's work before producing its own. Consensus is earned through independent agreement, not inherited through shared context.

P-07

Conversation-first interface

The input bar accepts natural language. Any text — task names, identifiers, natural requests — is interpreted with intent-matching semantics. The system never returns an "unrecognized command" error.

P-08

The application never exits on error

Under no condition does the system terminate and drop the user to a shell. All errors are caught, surfaced in plain English, and the interface remains live. The user is always in control.

04 — THE PIPELINE

From natural language request to approved commit

Every task follows the same structured pipeline. The complexity of a given task determines how many stages are engaged — a trivial change may skip directly to execution, while an architectural change invokes the full sequence.

01	Intent Classification	The system interprets the user's natural language request, classifies its complexity and risk, and determines the appropriate routing tier. A fast, inexpensive local model handles this — it does not require frontier capability.	Auton
02	Codebase Impact Analysis	The structural map of the codebase is queried: which files are affected, what depends on them, what tests cover them, and how wide the blast radius of this change is. This informs all downstream routing decisions.	Autom
03	Independent Planning	A planning agent produces a structured execution plan: files to change, invariants to preserve, rollback strategy, tests required. For complex tasks, a second planning agent from a different vendor produces its own plan independently — without access to the first.	Multi-
04	Adversarial Review	A reviewer agent — always from a different model family than the planner — evaluates both plans and challenges them. Disagreements between the two derivations are surfaced explicitly. If the agents converge independently, confidence is high. If they diverge, a tiebreaker resolves the dispute with full reasoning logged.	Cross
05	Patch Generation & Sandbox	A code-specialized model generates the actual file changes. Changes are applied in an isolated workspace — not the live repository. Automated verification runs: compilation, linting, static analysis, tests. The human can inspect and test-drive the result before any approval decision.	Iso
06	Human Approval	The diff, verification results, and model reasoning are presented for review. The human types "go" to proceed or rejects to restart. This gate is mandatory for every change that touches the repository. It cannot be automated away.	Human

07	Commit & Graph Update	On approval, changes are committed via controlled Git operations. The structural map of the codebase is updated to reflect the new state. The event log records the complete history of the task — every model input, output, decision, and approval.	Auto

06 — COST INTELLIGENCE

A structural understanding of your codebase that makes routing decisions intelligent

Most AI coding tools treat every task with the same level of process overhead. The underlying model has no structural understanding of the codebase, so it cannot know whether a change is trivial or architectural. The safest default is to apply maximum scrutiny to everything — which makes simple changes unnecessarily expensive.

MBO takes a different approach. On first use with any project, it builds an **Intelligence Graph** — a structural map of every file, function, import, call relationship, and test in the codebase. This graph is maintained incrementally: after every task, only the files that changed are re-analyzed.

Before any task begins, the graph is queried: if this changes, what else breaks? The answer determines routing:

T-0	Direct execution	Zero dependents, safelisted file, graph confirms no blast radius. No pipeline overhead. Change is made, logged, and reversible. Examples: a CSS color value, a config string, a copy change in a leaf component.	~$0

T-1	Lightweight review	Small change, minimal dependents, low risk. Single planning pass with structural verification. Human approval required.	~$0.003
T-2	Full DID pipeline	Multi-file, moderate complexity, meaningful dependency chain. Dual independent derivation with cross-vendor review, sandbox, human approval.	~$0.006
T-3	Maximum scrutiny	Architectural change, wide blast radius, or task touches files flagged as danger zones during onboarding. Full pipeline with tiebreaker, extended sandbox, and validation report at approval.	~$0.

The graph makes this routing decision intelligent rather than conservative. A CSS change that genuinely affects nothing else does not need a tiebreaker. An architectural refactor that touches the core data model does. The system knows the difference because it has read the codebase.

07 — HUMAN CONTROL

Autonomy is not the goal. Trustworthy assistance is.

The most valuable property of MBO is not what it does autonomously. It is what it explicitly does not do autonomously.

No AI model in the MBO pipeline has uncontrolled write access to your repository. Models may draft plans, generate code, and run validation in isolated workspaces. They cannot commit. They cannot

merge. They cannot mutate repository history. Every change that touches your codebase requires a human to type "go" first.

This is not a limitation of the current implementation to be relaxed in future versions. It is the design. It reflects a deliberate view about where AI judgment is reliable and where human judgment is irreplaceable.

> ON THE APPROVAL GATE
>
> The approval gate surfaces the diff, verification results, model reasoning, and a plain-English summary of what is about to happen. The human can read everything, test-drive the change in the sandbox, ask the system to reconsider its classification, or reject and restart. There is no countdown timer. There is no "auto-approve after N seconds." The gate waits.

Beyond the approval gate, MBO is designed around the principle that **the user should always be in control of the interface itself**. The terminal application never crashes to a shell. Errors are surfaced in plain English — not stack traces. Session state survives failures. The system accepts input even in degraded states.

This matters because an AI coding tool that loses your session, corrupts your context, or drops you to a shell at an inopportune moment is not just annoying — it is a trust failure. Every interaction with MBO is designed to reinforce that the human is in charge.

---

08 — COMPARISON

How MBO differs from existing tools

PROPERTY	TYPICAL AI CODING AGENT	MIRROR BOX ORCHESTRATOR
Model vendor strategy	Single vendor throughout	✓ Cross-vendor by design
Independent review	Model reviews its own output	✓ Adversarial cross-vendor review
Commit control	Varies — often autonomous	✓ Human "go" required, always
Audit trail	Limited or none	✓ Append-only, cryptographically chained
Codebase understanding	Raw file context	✓ Structural graph with impact analysis
Cost routing	Uniform — same cost for any task	✓ Complexity-tiered, graph-informed
Auth requirements	New account / API key	✓ Uses existing CLI sessions
Local model support	Rare	✓ First-class, for classification & patching
Prompt injection defense	Typically none	✓ Structural data/ instruction firewall
Failure behavior	Crash / silent / undefined	✓ Graceful degradation, always recoverable

09 — DEPLOYMENT

Three form factors, one codebase

MBO ships in three interfaces built on the same underlying pipeline. The choice of interface does not change what happens — it changes where you interact with it.

CLI

Terminal interface

The primary development target. macOS and Linux native. Windows via WSL2 or Docker. This is where the orchestrator first builds itself.

VS Code

Editor extension

First-class interface, not an afterthought. The same pipeline rendered inside VS Code panels. Meets developers where they already work.

Web

Hosted application

No install required. Connect a repository, describe what you want, watch it work. The heavy lifting is in the models — the server handles only coordination logic.

AUTH MODEL

MBO detects what you already have. If you're logged into Claude CLI, Gemini CLI, or OpenAI CLI, those sessions are used directly. Local models via Ollama or LM Studio are detected automatically and used for cost-sensitive roles. One optional OpenRouter key in one config

file covers any model not available via CLI session. There is no per-project credential management.

CURRENT STATUS

Initial shipping target is macOS CLI. Docker, Windows, Linux, VS Code extension, and hosted web app are all specified and designed into the architecture from the first version. Platform expansion follows Milestone 1.0.

---

10 — WHERE WE ARE

Shipping software, not a pitch deck

MBO is not a prototype. It is not a concept. The core system is running. Here is what is already built and what remains before public beta.

WHAT IS SHIPPED

The two hardest parts of this system are done. The **Intelligence Graph** — the structural map of files, functions, imports, calls, and tests that drives all routing decisions — is built and live. The MCP server that exposes it to every model in the pipeline is running under process supervision, with incremental updates after every task. The graph is not a future milestone; it is the foundation everything else runs on.

The **full terminal interface** is running: a four-tab TUI with live pipeline streaming, a per-stage token display (the Tokenmiser), a stats panel, task overlay, governance document viewer, inline task creation, and slash command autocomplete. The operator loop — the conversational anchor of the system — is live and maintains session

state across tasks. DID Gate 1 (independent plan derivation and convergence detection) is implemented and working. The human approval gate is enforced. Every session writes a full audit log.

The system routes through Claude and Gemini CLI sessions today. Local model support via Ollama is detected and active. The multi-vendor pipeline is not theoretical — it is the current operating mode.

WHAT REMAINS BEFORE PUBLIC BETA

Current version is **v0.3.56**. Public beta targets **v0.4**. The gap is implementation work, not architectural work. What is left: DID Gate 2 (blind code consensus between vendors), the real dry-run verification pass against the project's own test suite, the final config and role-assignment hardening, a code quality sweep, and documentation sync. The structure is in place. The remaining work is filling it in correctly.

> ON THE BETA GATE
>
> v0.4 ships when and only when a private demo convinces us it is ready for external eyes. There is no date-driven release. The acceptance bar is: would we use this on a real project, without hesitation, today?

HOW IT BUILDS ITSELF

MBO is developed using MBO. The orchestrator plans its own tasks, reviews its own patches through the cross-vendor pipeline, and requires human approval before any change to itself is committed. This is not a parlor trick — it is the most direct possible test of whether the system works. Every bug found in self-development is a bug that would have affected a user's codebase. Every fix is validated by the same pipeline a user would rely on.

> "*The goal is not to automate the developer out of the loop. The goal is to make the developer's judgment more powerful — by giving it better information, better process, and better tools.*"

The architecture is proven. The foundation is shipped. Beta is close.