Source: cppcc docs/b3ubot_end_to_end.md @ ff32fc8 — curated 2026-08-08; edits land in the source first.

b3ubot — End-to-End Implementation Plan

This is the **master implementation plan** for b3ubot: the complete development process from an empty repository to a production-ready Adaptive Programming Engine, organized according to the structure of `design.md` (the authoritative living design document — read it first; nothing here redefines design, only sequences its implementation).

Division of labor between the planning documents: - `design.md` — what the engine IS (architecture, contracts, locks, decisions). - `end_to_end.md` (this file) — the ROAD: phases → milestones → units of work, with the live §11 ledger. - `steps/step.N.txt` — the active slice: one (or a few) UoWs promoted to execution, with gates, LOCKs, and retro pairs.

0. Plan structure and conventions

- **Phase (Px)** — a coherent capability tranche with an entry/exit condition.
- **Milestone (Px.My)** — a demonstrable state inside a phase.
- **Unit of Work (Ux.y)** — ~1 day of focused work with named verification gates. UoWs are promoted to `steps/step.N.txt` for execution (skeleton → user ratification → execute → retro), exactly the b3u rhythm.

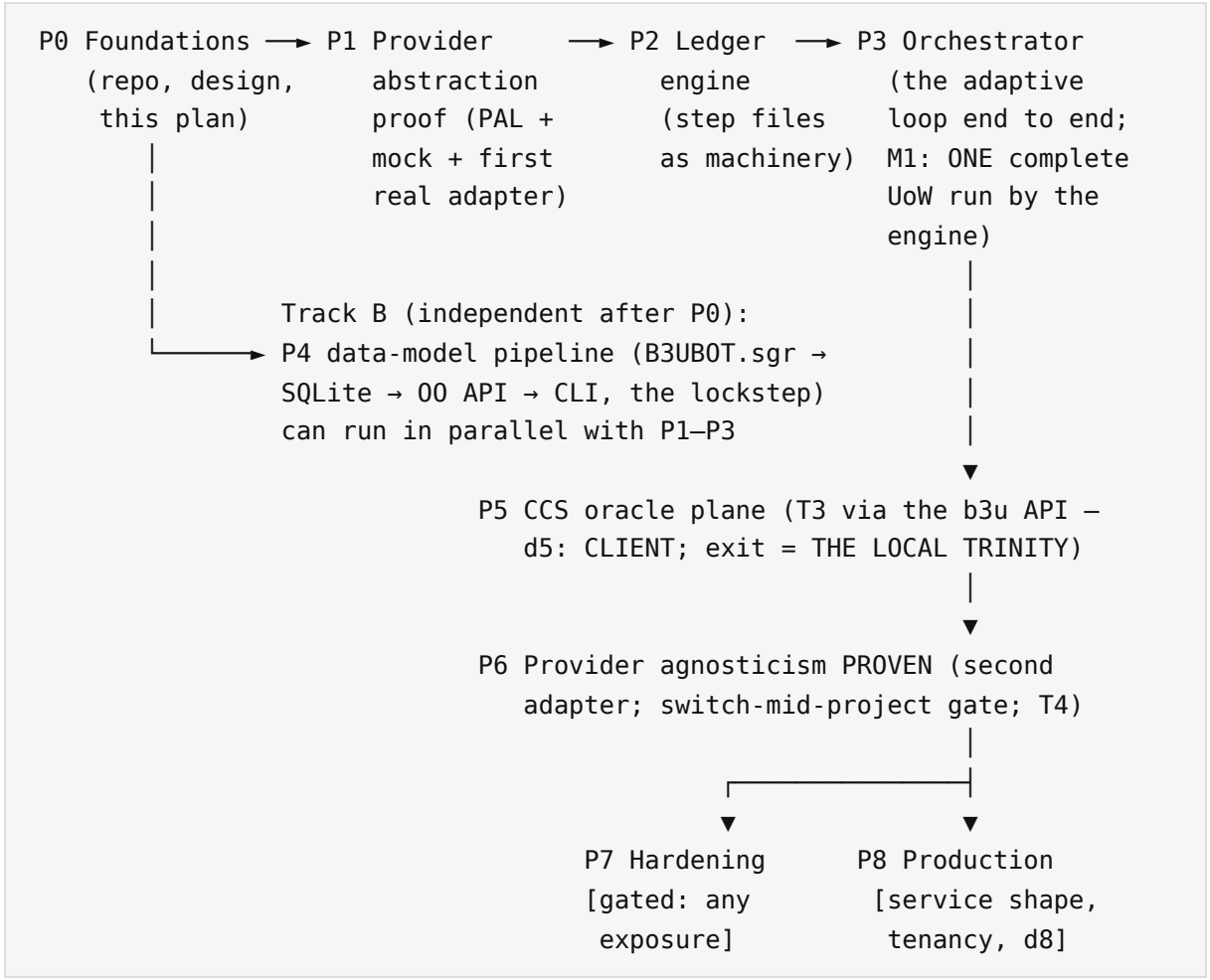
On completion, the §11 ledger row gets the step number, commit, and DONE status **in the same commit as the work** (§12 maintenance rules).

Status vocabulary (used in the §11 ledger): `todo` / `active` (has a step.N) / `done` / `dropped` (with reason).

Standing locks (design §2.10): B-1 cppcc never leaves the server; B-2 no public exposure before hardening; B-3 no disclosure / local-only; B-4 provider keys env-only; B-5 egress policy on workspace content; B-6 deterministic oracles gate, AI review advises.

---

1. The road at a glance



Two tracks after P0: **Track A** (the engine: P1 → P2 → P3) and **Track B** (the data-model pipeline: P4). They join where the orchestrator persists state through the model (P3 ↔ P4). Everything outward-facing waits on P7 — the b3u P5 lesson applied from birth.

The workflow DAG substrate (design §2.13; `dag/`, a fork of bet3/bot/dag — d11 RESOLVED at founding) underlies P3: workflows are collections of on-the-fly DAG instances, so the orchestrator UoWs build on the substrate's toolchain from U3.9 (which, despite its number, is P3's FIRST unit — see §5).

The deployment target for P1–P6 is THE LOCAL TRINITY (design §2.9, user directive 2026-07-13): b3ubot + the local-mode b3u instance + the Claude Code client on one Linux machine, with the Claude Code client as the ONLY internet-communicating component. d1 and d5 are RESOLVED accordingly — first adapter = `claude-code` (the local client); CCS plane = B3uDevClient against local b3u. The one cross-repo dependency is b3u's §8b BI side-track (the UB.1 verdict-only validate operation), consumed by U5.2.

Rough scale: ~31 UoWs at ~1 day each plus two milestone-level phases, excluding decision latency (d8) and any counsel lead time.

---

2. P0 — Foundations

P0.M1 — Project founding

UoW	Title	Status
U0.1	Repo founding: README, .gitignore, docs/, steps/, journals/	DONE
U0.2	design.md — authoritative living design document	DONE
U0.3	end_to_end.md — this master plan	DONE
U0.4	Dev environment + smoke check (uv venv; cppcc sibling probe; mock-provider hello)	todo

Exit: a fresh checkout builds its venv and passes a smoke script that proves the two external seams exist (a built cppcc sibling for later; a scripted mock-provider exchange).

---

3. P1 — Provider abstraction proof (Track A)

Entry: P0. Exit: ONE round trip — a developer intent goes to a real provider through the PAL, comes back as a proposed diff, is applied through the gated tool surface, and the result is shown — with the SAME arc running keyless on the mock provider.

P1.M1 — The PAL and one real provider

U1.1 — Envelope + adapter contract + mock provider - *Objective:* design §2.4 as code: the request/response envelope, the adapter interface (`capabilities()/run()/abort()`), and the mock adapter (fixture-scripted responses; zero network, zero keys). - *Done when:* pytest exercises the whole contract against the mock; no provider name appears outside `adapters/`.

U1.2 — First real adapter (d1 RESOLVED: claude-code) - *Objective:* the `claude-code` adapter driving the LOCAL Claude Code client (design §2.4/§2.9 — the trinity's sole internet-touching component); B-4 key/credential hygiene from the first line (env-only, never logged,

clientless = honest skip). - *Done when:* the same contract suite passes against the live adapter (auto-skip when no Claude Code client is available — the b3u keyed/keyless discipline).

U1.3 — One-round-trip CLI - *Objective:* `b3ubot ask` (working title): intent → context assembly → PAL → proposed diff → gated apply → result. The tool surface's minimal core (file read/write, workspace-scoped) lands here. - *Done when:* the arc completes on a toy workspace via mock AND via the real adapter; every workspace byte that left is in the ProviderRun trail (B-5's first, file-based form).

U1.4 — P1 gates + retro - *Objective:* the live-gates script (the p1_gates.sh pattern): boot, mock arc, keyed arc, hygiene legs (no key in logs; egress trail complete). - *Done when:* gates green keyless AND keyed; retro pair lands.

U1.5 — Human-relay adapter (manual, decouples workflow from any AI tool) - *Objective:* `adapters/human/` (design §5 d13): a third first-class PAL adapter where the developer manually relays context to/from an AI coding tool running in a separate session, validating the whole Adaptive Programming workflow without any AI-tool-specific automated integration. Injectable `output_fn`/`input_fn` give it the same offline, zero-network, zero-keys testability as the mock adapter. - *Done when:* the shared contract suite exercises it with zero real interactivity; a dedicated unit test suite covers its own logic; `make smoke`/`make test` stay green.

U1.6 — Wire `--provider human` into `ask` (reconcile `propose_ask()`) — DONE (step.8) - *Objective:* `propose_ask()` hard-requires a `TEXT`-kind response and today raises for the human adapter (step.7 §1 finding; Q-7-C). Reconcile the CLI's acceptance rule with the human adapter's single-DONE response shape (step.8.txt Q-8-A/Q-8-B) and add `human` to `_make_adapter()`/ `--provider`. - *Done when:* `ask --provider human` completes end-to-end against an injected paste-back without the generic TEXT-response error; no regression on mock/claude-code; the reconciliation approach is recorded, not silent (step.8.txt G4). - *Delivered:* Q-8-A resolved Option A (`propose_ask()` accepts a distinct TEXT response OR a content-bearing terminal DONE, mirroring test_pal_contract.py's own step.7 extension); a new `adapter=` injection seam on `propose_ask()` lets tests exercise `provider="human"` with fake I/O. Q-8-B resolved (ii): `main()` refuses loudly on a non-TTY stdin before `input()` can block. 55 passed / 11 skipped (additive vs step.7's 52p/11s baseline), zero regressions.

---

4. P2 — The ledger engine (Track A)

Entry: P1. Exit: b3ubot parses and emits its own ledger byte-stably, with the b3u step corpus as the acceptance fixture.

U2.1 — Step-file parser — DONE (step.10) - *Objective:* the canonical sections as a schema; the 50 b3u step files (read-only fixtures) must parse. *Done when:* corpus parses; unknown-section and malformed inputs refuse loudly. (Corrected from "34": that figure was accurate when P2 was originally scoped; the b3u corpus grew to 50 — step.1 through step.50, contiguous — by the time this step executed. Verified live via `ls b3u/steps/step.*.txt` excluding `.diff/.cdiff/.gdiff` siblings; step.10.txt §1 has the full survey.) - *Delivered:* `app/ledger/parser.py` (`parse_step_file`/`parse_step_text` → `StepFile`/`Section` dataclasses) + `app/ledger/errors.py` (`UnknownSectionError`/`MalformedStepFileError`); schema derived from a full corpus survey, written up as `docs/step_file_schema.md`. All 50 b3u corpus files parse clean (the graded fixture set) plus this repo's own step files (bonus check, self-inclusive — step.10.txt itself parses too). 70 new tests (`tests/test_ledger_parser.py` + `tests/fixtures/ledger/`), zero regressions (55p/11s → 125p/11s). Section (0) titled exactly "Status" is the one universal invariant found (50/50); the closing-section vocabulary (Verification gates/LOCKs/Open questions/Acceptance/Hash backfill) is near-universal but NOT rigidly ordered or titled — one corpus file (`b3u/steps/step.8.txt`) uses "Oracle (gates G1-G3)" instead of "Verification gates" and orders Hash backfill before Acceptance — so the parser enforces the marker shape and section (0), not an exhaustive title/order template.

U2.2 — Emitter + THE round-trip oracle — DONE (step.11) - *Objective:* emit(parse(file)) == file, byte-stable, over the corpus and over generated skeletons. *Done when:* the oracle is a regression suite (this is the ledger's .fgr/.ubr moment). - *Delivered:* `app/ledger/emitter.py` (`emit_step_file`), the structural inverse of U2.1's parser, plus a new `app/config/` package (`config.txt` + `loader.py`) that resolves the internal-banner block as a per-deployment runtime value (`${HOME}/BANNER.md`, expanded; absent ⇒ no banner block at all) — an operator design decision made mid-step: the banner is NOT step-file data, so `StepFile` gained no banner field and the emitter never reproduces one from a parsed source's own bytes. Two round-trip guarantees, named explicitly (no silent redefinition of "byte-stable"): **byte-identical** over every real file in the corpus (50 b3u + this repo's own 11, self-inclusive of step.11.txt — 61/61 PASS live), and **structural** (`parse(emit(skeleton))` reproduces the same header fields/section shape/body text) over 4 hand-built "generated skeleton" `StepFile`s that never went through a real file, which have no original bytes to match. Byte-identity required additively extending U2.1's model (`Section.raw` / `StepFile.title_raw` / `StepFile.header_raw`, all `None`-default, zero existing-field/test impact): a fresh corpus resurvey found 52/60 files wrap their title onto continuation lines that `title_line` silently dropped, and 2/534 section-marker transitions have a real (not assumed)

0-vs-1 leading-blank-line difference that `Section.body`'s stripping couldn't distinguish. 79 new tests (`tests/test_ledger_roundtrip.py` + `test_config_loader.py`) plus 1 more from U2.1's own self-inclusive corpus parametrization growing by a file (step.11.txt itself), zero regressions (125p/11s → 205p/11s; U2.1's own test bodies unedited and fully green). Full writeup: docs/step_file_schema.md "What U2.2 needed on top of U2.1's model", steps/step.11.txt.

U2.3 — UoW ledger operations — DONE (step.12) - *Objective:* the §11-table model (corrected from the original "§9" — stale, see below): rows, status transitions (todo → active → done/dropped), same-commit update discipline encoded as an API. *Done when:* ledger ops round-trip and validate transitions. - *Delivered:* `app/ledger/table_parser.py` (`parse_ledger_table_text`/ `_from_doc`/`_file` → `LedgerRow`/`LedgerTable`), `table_emitter.py` (`emit_ledger_table_text`, `replace_ledger_table_in_doc`), `table_transitions.py` (`transition`, `record_commit`, `add_row`, `validate_row_invariants`), `table_errors.py` — a SEPARATE module family from U2.1/U2.2's `parser.py`/`emitter.py`/`errors.py` (a different format: the §11 TABLE, not a `steps/step.N.txt` file). **Ground-truth correction found while drafting:** this bullet and three other places in this file (§0, lines ~16/29/30/32) said "§9 ledger"/"§10 maintenance rules" — stale (section numbers shifted as the plan grew, the same root cause as step.10's "34→50"); the table is actually at §11, maintenance rules at §12 — corrected here, and `table_parser.py` locates the heading by regex rather than a hardcoded number so it cannot go stale the same way again. **A real formatting bug found by the parser's own strict cell-width check**: two continuation rows (U3.9/U3.9a's "see §5..." lines) were one character narrower than every other row in the live table — fixed by hand (2 single-space insertions), not by loosening the parser. **Resolved transition rule set** (no real `dropped` row exists in either this repo's or the b3u sibling's own ledger to crib from — checked live): todo→active (requires `step=`), todo→dropped and active→dropped (both require `reason=`), active→done (requires `commit=`); `done`/`dropped` both terminal (ties directly to §12's monotonic-numbering rule — reviving a closed UoW is always a NEW `add_row()`, never a transition out of a closed one). **Where a `dropped` reason is recorded**: the table has no Reason column; `transition(..., "dropped", reason=...)` appends `" (dropped: <reason>)"` to the Title cell, following the same parenthetical-annotation convention the live U1.5/U1.6 rows already use. Two round-trip guarantees (same shape as U2.2's precedent): **byte-identical** over the real live table with zero mutations (verified live, including `replace_ledger_table_in_doc` as a byte-identical no-op against the real `end_to_end.md`), **structural** for any row touched by `add_row`/`transition`/`record_commit` (word- wrapped synthesis, no hyphenation — the historical corpus's hand- hyphenated wraps like "decou-"/"ples" are not reproduced for fresh content). The split-row pattern (U3.9/U3.9a's real precedent) is `add_row()` with a lettered id — no separate

`split_row()` (named reasoning in `table_transitions.py`'s `add_row()` docstring). Honest scope on "same-commit update discipline": `transition(row, "done", ...)` cannot succeed without a `commit=` argument (mechanically enforced); git atomicity itself is not something a library call can enforce (named explicitly, not overclaimed — docs/ledger_table_schema.md "What this API can and cannot enforce"). 40 new tests (`tests/test_ledger_table_roundtrip.py` + `test_ledger_table_transitions.py`), zero regressions (205p/11s → 247p/11s — corrected from "245p/11s", a hand-authoring typo found live while drafting step.13; 205+42=247 is what the suite actually reports and what step.12.diff.txt's own Numbers table already said). Full writeup: `docs/ledger_table_schema.md`, `steps/step.12.txt`.

U2.4 — Retro-pair machinery + ledger CLI — DONE (step.13) - *Objective:* narrative-retro templating, `.gdiff.txt` generation via `scripts/step_gdiff` (corrected from "cdiff generation (the claude_diff pattern)" — `claude_diff/.cdiff.txt` was retired repo-wide for an off-by-one range + self-referential growth bug; every step in this repo, 1 through 12, has used `step_gdiff/.gdiff.txt` exclusively, never `claude_diff/.cdiff.txt`), `b3ubot ledger` surface. *Done when:* a synthetic UoW walks draft → close leaving the full artifact set. - *Delivered:* `app/ledger/retro.py` (`draft_step_skeleton`, `generate_diff_skeleton`, `run_step_gdiff`) — a survey of steps/step.7.diff.txt–step.12.diff.txt (all six read in full) found an invariant STRUCTURAL template (title/banner/"Executed:" line/four section headings/closing line) but genuinely narrative CONTENT inside it (plan-vs-actual, gate evidence, findings, numbers); the generators pre-fill only the mechanical parts and mark every narrative slot with `FILL_IN`, deliberately not fabricating reflective prose. `run_step_gdiff` shells out to the real `scripts/step_gdiff` rather than re-deriving its git-diffing logic. New `b3ubot ledger` CLI subcommand (`app/cli.py`): `show` / `transition` / `add-row` / `record-commit` / `close` / `draft-step` / `draft-retro` / `gdiff`, all thin read-doc/call-library/write-doc wrappers over U2.1–U2.3's existing APIs — no new ledger behavior lives in the CLI layer. The synthetic-UoW done-when (`tests/test_synthetic_uow_arc.py`) walks draft → ratify(trivial) → execute(trivial) → verify(trivial) → close entirely inside a throwaway `tmp_path` git repo and asserts the FULL real artifact set: a step.N.txt-shaped file, a step.N.diff.txt-shaped file (`FILL_IN` placeholders present), a REAL `step.N.gdiff.txt` (the actual `scripts/step_gdiff`, confirmed feasible against a synthetic repo and run for real, not mocked — investigated live, see steps/step.13.txt), and a ledger row transitioned to `done`. 28 new tests (`tests/test_ledger_retro.py`, `test_ledger_cli.py`, `test_synthetic_uow_arc.py`), 247p/11s → 277p/11s. One real live-data collision found and fixed, not silently worked around: dogfooding U2.4's own row through `done` (below) broke a pre-existing step.12 test that had picked U2.4 as its "a real todo row" example —

fixed by pointing it at U3.1 (the next real todo row) instead of loosening the test. Full writeup: `docs/ledger_cli_and_retro.md`, `steps/step.13.txt`.

---

5. P3 — The orchestrator (Track A)

Entry: P2 + the DAG substrate (U3.9 — numbered by arrival order, first by dependency). Exit — M1, the engine's first light: b3ubot runs ONE complete UoW on a toy workspace end to end (intent → skeleton → human ratify → execute → verify → retro) with a real provider, leaving a ledger AND a frozen execution DAG indistinguishable in form from hand-run b3u artifacts.

P3 exit state (step.21, U3.8): M1 ACHIEVED, with one named caveat. *The arc runs end to end in ONE invocation (`b3ubot run`) — proven hermetically (pytest, mock provider) AND live (the CLI smoke whose frozen execution DAG is committed verbatim as the `m1_exec` regression corpus); the frozen DAGs pass the substrate's round-trip oracle per-run and in the permanent 4-language corpus matrix; ledger + step-file + retro-pair form asserted mechanically against the hand-run machinery. The "with a real provider" clause is held by a STANDING keyed oracle leg (`scripts/m1_oracle.sh`) that auto-runs the identical arc live when the environment provides a usable `$ANTHROPIC_API_KEY` — in this environment the key file is an unfilled template, so the live confirmation is recorded honestly as pending-on-environment (the step.3/9 keyless discipline), never fake-passed.*

UoW	Title (gates named at promotion)
U3.9	**DAG substrate adoption** (design §2.13): rebuild the dag/ toolchain from the cloned fork (cppcc frontend, cctdag, 4-language OO APIs); the upstream regression suite green IN-REPO (byte-identity matrix + 2-pass round-trip). *Precedes every other P3 UoW.* **DONE step.5, pulled forward ahead of P2 at the operator's explicit direction 2026-07-18 — self-contained within dag/, no P1/P2 dependency.**
U3.9a	First b3ubot-kind instance (`execution`) authored + round-tripped against U3.9's now-proven substrate. Split out of the original U3.9 text (end_to_end.md §12's re-planning rule) — real NEW authoring work, not verification of what already exists.
U3.1	State machine (design §4) + engine-state persistence; the EXECUTING entry compiles the ratified skeleton into an on-the-fly execution DAG (nodes=tasks, edges=deps, kv=state — §2.13). **DONE step.14** — `app/orchestrator/` (new package): `state_machine.py` (8 states, 9 legal edges, `validate_transition`/`legal_targets`, refuses loudly on an illegal move — also resolves the ledger-`active`-vs-design-§4 relationship in its own docstring, mirrored in design.md §4); `persistence.py` (file-based,

UoW	Title (gates named at promotion)
	`~/.b3ubot/orchestrator/<uow>.json`, mirrors `app/egress.py`'s `data_home()` exactly — `start_state`/`load_state`/`advance_state`, resumable across a process restart); `compiler.py` (`compile_execution_dag`: RATIFIED→EXECUTING, a fixed V0 4–5-node template matching `dag/instances/execution_example.dag`'s own shape, `oracle_gate` added only when a step's own gates section names a T3/CCS check — none do yet). 50 new orchestrator tests (`tests/test_orchestrator_{state_machine,persistence,compiler}.py`) + 2 from the step-file parser's/emitter's own self-inclusive corpus parametrization picking up this step's own step.14.txt (the same growth pattern steps 10–13 documented for their own predecessors), zero regressions (277p/11s → 329p/11s). Full writeup: `docs/orchestrator_state_machine.md`, `steps/step.14.txt`.
U3.2	Tool surface + policy gates (allowlist shell, git ops, approval for destructive — the no-`--force` posture). **DONE step.15** — `app/tools/` (U1.3's single module grown into a package, every existing call site working unchanged through `__init__` re-exports): `workspace.py` (+`list_dir`, +`delete_file` refuse-stub, all ops C-4-B-scoped); `shell.py` (`run_shell`: NAMED allowlist — default `bash make pytest python python3`, `shell_allowlist` key in `app/config/config.txt` overrides/narrows, refusal names the fix; mandatory timeouts, output capture, non-zero-exit-as-data; `run_gate`: §2.7 oracle invocation V0 — runs a named `scripts/` gate script under the same machinery, real T0/T1 tiers stay U3.6); `gitops.py` (status/diff/add/commit on a verified repo root — git's walk-up to an enclosing repo refused; push/history-rewrite/branch-delete exist as `ApprovalRequiredError` refuse-stubs naming U3.4); `trail.py` (LOCK C-15-B: EVERY call — refusals included — appended to `~/.b3ubot/tool_calls.jsonl`, the file-based ProviderRun-trail precursor, a SIBLING of `app/egress.py` sharing its `data_home()`). DAG/scheduler wiring deliberately NOT here (U3.5). 51 new tests (`tests/test_tools{,_shell,_git,_trail}.py` + hermetic-`$B3UBOT_HOME` `tests/conftest.py`), zero regressions (329p/11s → 380p/11s, 382 with this step's own corpus pickup); `tests/test_ledger_table_roundtrip.py`'s thrice-broken todo-row example converted to a status-query pick (step.14 retro's own named remedy). Full writeup: `docs/tool_surface.md`, `steps/step.15.txt`.
U3.3	SKELETON flow: PAL-drafted step files (ground-truth probe prompts; LEANs presented for ratification). **DONE step.16** — `app/orchestrator/drafting.py` (`draft_skeleton()`: design §4's `INTENT --draft(PAL)--> SKELETON` arrow, and design §2.3's first ledger op `draft(intent) -> skeleton`, as code — the PAL-driven sibling of U2.4's manual `draft_step_skeleton()` template writer, which stays untouched). Bounded V0 workspace probe through U3.2's trail-recorded tool surface (`list_dir` of the root + README.md + intent-named files, capped at 8 files / 20k chars each — deliberately never the whole tree; §2.4's "Providers see what the policy layer allows — no more"), engine-owned context

UoW	Title (gates named at promotion)
	assembly, provider draft accepted via step.8's widened TEXT-or-content-bearing-DONE rule (so the human-relay adapter is a first-class drafting provider — proven with injected I/O), validation = `app/ledger/parser.py` refuse-loudly + the engine-assigned `Step:`/`UoW:` header check (the ENGINE owns step-number selection: next free plain `step.N.txt` in the target workspace's `steps/`, retro `.diff`/`.gdiff` siblings never counted) + a §(1) to stamp. ONE fixed retry with the validation error AND the failed draft fed back, then a loud `SkeletonDraftError` — deliberately NOT a budget system (d9 stays U3.6's open decision). The engine STAMPS its own probe record into the drafted skeleton's §(1) — what was actually listed/read, with sizes — never trusted from provider prose; every attempt is egress-recorded unconditionally (the C-4-C posture) BEFORE validation; engine state advances INTENT→SKELETON with the step recorded, and an illegal move refuses BEFORE any provider call. New top-level `b3ubot draft` CLI — placed as `ask`'s sibling (a PAL round trip with engine-state effects), not a `ledger` subcommand; same Q-8-B non-TTY refusal for `--provider human`. No provider name appears in the orchestrator (§2.11 — `app/cli.py`'s `_make_adapter()` keeps the mapping and passes the built instance in). 18 new tests (`tests/test_orchestrator_drafting.py`, incl. the toy-workspace-only guarantee: this repo's own `steps/` asserted untouched) + 2 self-inclusive corpus pickups, zero regressions (382p/11s → 402p/11s). Ratify gate/amendment loop NOT here — the drafted skeleton sits awaiting ratification (U3.4). Full writeup: `steps/step.16.txt`.
U3.4	Ratify gate + amendment loop (human-in-the-loop, the load-bearing gate). **DONE step.17** — `app/orchestrator/ratify.py` (`ratify_skeleton()`: design §4's `SKELETON --human ratify--> RATIFIED` arrow, the first of the two load-bearing human gates). State legality proven BEFORE anything is read or presented (step.16 retro's carried lesson — an illegal ratify refuses with zero reads); skeleton loaded via the engine record's own `step` field through the trail-recorded tool surface, parsed by `app/ledger/parser.py`; summary + LEANs presented through injectable I/O end to end ("LEANs are presented for override exactly as the b3u steps present them"): `extract_leans()` finds the open-questions section by title PREFIX (the corpus suffixes titles) and splits on `Q-N-X` marker lines only — no prose micro-parser; no markers ⇒ one unsplit block, no section ⇒ an honest "no LEANs to present". Three decisions: **ratify** (per-LEAN confirm-or-override; all confirmed ⇒ the engine stamps its RATIFICATION RECORD into §(0) — the step.16 engine-stamp pattern, post-stamp re-parse guard — and advances SKELETON→RATIFIED; ANY override BECOMES an amendment: a skeleton is ratified as its text stands), **amend** (decline with notes → appended to `steps/<step>.amendments.txt` BESIDE the skeleton, timestamped, trail-recorded — NO state change: the pre-ratification amendments cycle is file-level, worked out against the machine's actual 9 edges, which SUFFICED; the diagram's RATIFIED→SKELETON edge is the post-ratification variant, undriven until U3.5 makes RATIFIED dwell real), **discard**

UoW	Title (gates named at promotion)
	(double-confirmed ⇒ SKELETON→DISCARDED terminal; the file stays on disk). Automatic PAL re-draft on amendments deliberately NOT built (the dispatch's LEAN, taken — and `draft_skeleton()` correctly refuses SKELETON→SKELETON, so it is a genuinely new flow, recorded as Q-17-A). Plus the **destructive-op approval mechanism** (`app/tools/approval.py`: injectable `ApprovalCallback` + `interactive_approval()`): `delete_file` and `git_delete_branch` gained REAL approved paths (approve=None still refuses exactly as U3.2's stubs did; declined refuses with the human-DECLINED message; approved executes, trail args carry `"approved": true` — three distinct audit facts); `git_push`/`git_rewrite_history` STILL refuse, each for its own narrower named reason (remote/refspec policy undesigned / no concrete verb + no §2.6 provenance record) and take no callback at all — no fake flows. New `b3ubot ratify` CLI (`draft`'s sibling; Q-8-B non-TTY refusal unless `--yes`, the one honest bypass: ratify-as-is, every LEAN confirmed as leaned). 25 new ratify tests (`tests/test_orchestrator_ratify.py`) + 14 approval tests grown into `tests/test_tools{,_git}.py`, zero regressions beyond the honestly updated U3.2 stub-message pins (402p/11s → 443p/11s incl. this step's own corpus pickup). Full writeup: `steps/step.17.txt`.
U3.5	EXECUTING loop: scheduling from the execution DAG's ready frontier; tool acting + evidence recorded into the step file AND node kv state. **DONE step.18** — `app/orchestrator/executing.py` (`execute_uow()`: design §4's "Everything between the gates is machine" + §2.13's "the DAG IS the live progress surface — what may run next = the ready frontier", as code). State legality FIRST (an EXECUTING record RESUMES; only RATIFIED may enter); compile via U3.1's `compile_execution_dag`, instance WRITTEN to `steps/<step>.execution.dag` BESIDE the skeleton (the U3.4 amendments-file precedent; §2.12 "file trees for … workflow DAG instances"; NOT a workspace `dag/` dir — that name is the substrate's), full rewrite via `to_dag_text` through the trail-recorded tool surface on EVERY node status change (running first, so a crash leaves an honest marker; then done/failed/declined + fact kvs — timestamps, `provider`/`run_id`, `approved`, `applied`, `exit_code`, the execution_example.dag attr shape). Round-trip verified through the REAL `dag/cppcc/bin/dagModel` at write + completion (droppings cleaned; parser-absent degrades to a NAMED skip for writes but REFUSES resume — reading back IS parsing). Per-node-kind V0 semantics mapped onto what exists: context_assembly = U3.3's bounded probe (intent = title + §(2) deliverables); provider_run = ONE PAL call through the injected adapter (deterministic first entry "EXECUTE :