

fuel-code: Core Abstractions (Post-Phase 4 Audit)

Last updated: 2026-02-27. Based on a full codebase audit after four phases of implementation.

The Minimal Set

Six constructs explain every aspect of the codebase as it stands today. Blueprint becomes the 7th when Phase 5 lands.

Construct	One-line definition	Why it's in the set
Event	An immutable, timestamped, typed fact about something that happened	The atomic unit. Everything enters the system as an event. The pipeline is the backbone.
Session	A CC invocation from start to end, with a lifecycle state machine	The primary view. Users think in sessions. Stats, transcripts, summaries hang off sessions.
Workspace	A project identity derived deterministically from a git remote URL	The organizing principle. Sessions and git activity are grouped by workspace.
Device	A machine that emits events	Attribution. "Where did this happen?" Every event, session, and git action is attributed to a device.
Transcript	The parsed, structured representation of a session's CC conversation	The richest data. Two tables, async pipeline, recovery, summarization. Where most complexity lives.
GitActivity	A correlated, queryable git operation (commit/push/checkout/merge)	The bridge between CC sessions and git work. Enables the timeline view.

The core relationship: Event is the input. Session + Transcript + GitActivity are the processed outputs, organized by Workspace and attributed to Device. Everything flows through the event pipeline; everything materializes into one of these constructs.

Abstraction Status

1. Workspace — Clean, Tight

The most disciplined abstraction. Type definition, canonical ID derivation, resolver/upsert, junction table, route, CLI command — all present and aligned. The `normalizeGitRemote()` / `deriveWorkspaceCanonicalId()` chain in `shared/canonical.ts` is the most principled code in the repo: pure functions, deterministic, no side effects, fully tested.

Key files: `shared/src/canonical.ts` , `shared/src/types/workspace.ts` , `core/src/workspace-resolver.ts` , `core/src/workspace-device-link.ts` , `server/src/routes/workspaces.ts` , `cli/src/commands/workspaces.ts`

No issues found.

2. Session — Richest Abstraction, Most Drift

Session is the center of gravity. It touches more code than any other abstraction: lifecycle state machine, transcript pipeline, summary generation, recovery, backfill. But the TypeScript type has significantly drifted from the database schema.

Key files: `shared/src/types/session.ts` , `core/src/session-lifecycle.ts` , `core/src/session-pipeline.ts` , `core/src/session-recovery.ts` , `core/src/handlers/session-start.ts` , `core/src/handlers/session-end.ts` , `server/src/routes/sessions.ts` , `cli/src/commands/sessions.ts` , `cli/src/commands/session-detail.ts`

Drift findings:

- **Phantom field `cc_session_id`** : The `Session` interface defines `cc_session_id: string` (`session.ts:39-40`), but there is no such column in the database. Claude Code's session ID is stored directly as the `id` column. The JSDoc on `id` says "ULID primary key" — this is wrong; it's the CC session ID (a UUID, not a ULID). Any code that reads `session.cc_session_id` from a DB row gets `undefined` .
- **17 fields in type vs 29+ columns in DB**: The `Session` interface omits: `initial_prompt` , `source` , `summary` , `tags` , `transcript_s3_key` , `parse_error` , `remote_env_id` , `end_reason` , all 11 stat columns (`total_messages` , `user_messages` , `assistant_messages` , `tool_use_count` , `thinking_blocks` , `subagent_count` , `tokens_in` , `tokens_out` , `cache_read_tokens` , `cache_write_tokens` , `cost_estimate_usd`), `created_at` , `updated_at` . The type has `cwd` , `git_remote` , and `cc_session_id` which are NOT columns.
- **capturing is dead code**: Defined in the lifecycle state machine (`TRANSITIONS` allows `detected` \rightarrow `capturing`) but nothing in the codebase ever transitions to it. Sessions go `detected` \rightarrow `ended` directly. The spec describes `capturing` as "events streaming in, session is live" but this intermediate state is never signaled.
- **archived transition never triggered**: The `TRANSITIONS` map allows `summarized` \rightarrow `archived` but there is no archival process, no TTL-based pruning, no archive CLI command. The entire archival concept described in `CORE.md` is unrealized.
- **transcript_path vs transcript_s3_key** : The `Session` type uses `transcript_path` . The database column is `transcript_s3_key` . These are semantically different (local path vs S3 key). The type doesn't represent the actual column.

3. Event — Well-Structured Core, Partially Populated

The Event pipeline (emit \rightarrow HTTP/queue \rightarrow Redis Stream \rightarrow consumer \rightarrow handler \rightarrow broadcast) is the most thoroughly implemented piece of infrastructure. The type, schemas, transport, consumer, and WebSocket broadcasting all work end-to-end.

Key files: `shared/src/types/event.ts` , `shared/src/schemas/` (payload schemas), `core/src/event-processor.ts` , `core/src/handlers/index.ts` , `server/src/routes/events.ts` , `server/src/pipeline/consumer.ts` , `server/src/redis/stream.ts` , `server/src/ws/broadcaster.ts` , `cli/src/commands/emit.ts` , `cli/src/lib/queue.ts` , `cli/src/lib/drain.ts`

Drift findings:

- **14 types declared, 6 have handlers:** `session.start` , `session.end` , `git.commit` , `git.push` , `git.checkout` , `git.merge` have handlers. `session.compact` has a Zod schema but no hook to emit it and no handler. The 4 `remote.*` and 3 `system.*` types have neither schemas nor handlers — they are forward declarations.
- **No PreCompact hook:** CORE.md specifies it. The hooks package only has `SessionStart.sh` and `SessionEnd.sh` .
- **_device_name / _device_type transport hints:** The emit command injects these into `event.data` , and the event processor reads them out, uses them for device resolution, then delete s them from the data object before persistence. This is undocumented, untyped mutation of the event payload — a pragmatic hack that works but violates the "events are immutable" invariant at the application layer.
- **Handlers don't use validated payload types:** Zod schemas define `SessionStartPayload` , `GitCommitPayload` , etc., but handlers cast from the untyped `event.data` with raw as assertions (`event.data.cc_session_id as string`). The validation happens at the ingest boundary but the typed result is thrown away.

4. Device — Minor Type Mismatch

Functionally complete: init generates the ID, config stores it, events carry it, the resolver upserts it.

Key files: `shared/src/types/device.ts` , `core/src/device-resolver.ts` , `server/src/routes/devices.ts` , `cli/src/commands/init.ts` , `cli/src/commands/status.ts`

Drift findings:

- **Type/DB column name mismatch:** The `Device` interface uses `platform` and `os_version` . The database columns are `hostname` , `os` , `arch` . The type omits `hostname` entirely. The `resolveOrCreateDevice()` function writes to the DB columns but the TypeScript type uses different field names.

5. Blueprint — Entirely Absent

Zero lines of implementation. No type, no schema, no detector, no routes, no CLI commands, no database tables. The `remote_envs` table doesn't exist. The `sessions.remote_env_id` column exists in the migration but references nothing and is never populated. The `infra/` directory doesn't exist.

This is expected (Phase 5 work), but one of the "Five Abstractions" from the spec has no material presence in the code.

6. GitActivity — Emerged as First-Class

Not one of the original five, but now a full abstraction with its own type, database table (5 indexes), 4 handlers, a correlation engine, and query surface area across timeline, session detail, and workspace detail routes.

This is a denormalized materialized view of git-type events. The same data exists in the `events` table, but the dedicated table with typed columns enables efficient queries that would be expensive as JSONB lookups.

Key files: `shared/src/types/git-activity.ts` , `core/src/git-correlator.ts` , `core/src/handlers/git-commit.ts` , `core/src/handlers/git-push.ts` , `core/src/handlers/git-`

```
checkout.ts , core/src/handlers/git-merge.ts ,
server/src/db/migrations/003_create_git_activity.sql
```

7. Transcript — Emerged as First-Class

Also not one of the original five, but now a substantial subsystem with its own types (`RawTranscriptLine` , `TranscriptMessage` , `ParsedContentBlock` , `ParseResult` , `TranscriptStats`), two database tables (`transcript_messages` , `content_blocks`), an async pipeline, recovery mechanisms, and summary generation.

Key files: `shared/src/types/transcript.ts` , `core/src/transcript-parser.ts` , `core/src/summary-generator.ts` , `core/src/summary-config.ts` , `core/src/session-pipeline.ts` , `core/src/session-recovery.ts` , `server/src/db/migrations/002_transcript_tables.sql` , `server/src/routes/sessions.ts` (transcript endpoints)

Cross-Cutting Discipline Issues

Types Don't Match the Database

The most systemic issue. Both `Session` and `Device` interfaces claim to "map to the Postgres table" but don't.

- Session type has 17 fields for a 29-column table
- Device type uses different column names (`platform` / `os_version` vs `hostname` / `os` / `arch`)
- TypeScript gives false confidence about the shape of data from DB queries
- Fields like `cc_session_id` exist in the type but not the DB, so `row.cc_session_id` is always `undefined`

Validated Schemas Discarded at Handler Boundary

The Zod payload registry validates event payloads at ingest time. But handlers don't use the validated/typed result. Instead they do: `const ccSessionId = event.data.cc_session_id as string` . The schemas define types like `SessionStartPayload` that could give handlers full type safety, but they're unused. Validation and consumption are disconnected.

Duplicated Pagination Logic

Three route files (`sessions.ts` , `timeline.ts` , `workspaces.ts`) independently implement identical base64 cursor encode/decode pagination helpers. Same `PaginationCursor` type, same `decodeCursor` / `encodeCursor` functions, copy-pasted three times.

tx: any Type Escapes

Three files cast postgres.js transactions to `any` to work around a TS 5.9 typing issue (`git-commit.ts:56` , `session-lifecycle.ts:329` , `session-pipeline.ts:185`). This disables type checking within transaction blocks — the most critical code paths.

result_s3_key — Column Exists, Never Written

The `content_blocks` table has a `result_s3_key` column for offloading large tool results to S3. The `buildArtifactKey()` utility exists in `s3-keys.ts` . Neither is ever used. Large tool results stay inline in `result_text` regardless of size.

hooks_installed — Ghost Column

The `workspace_devices` table has a `hooks_installed` column from the initial migration. It's never read or written anywhere. The functional column is `git_hooks_installed`.

Additional Subsystems (Not Core Abstractions)

These are significant pieces of infrastructure that don't rise to "core abstraction" status but are worth noting:

- **Session Backfill:** Scans `~/ .claude/projects/` for historical sessions, emits synthetic events, uploads transcripts. Has its own state file, concurrency management, and resume support. (`core/src/session-backfill.ts` , `core/src/backfill-state.ts` , `cli/src/commands/backfill.ts`)
- **Local Event Queue:** Disk-based fallback when the backend is unreachable. Atomic writes, ULID-ordered draining, dead-letter after 100 attempts. (`cli/src/lib/queue.ts` , `cli/src/lib/drain.ts` , `cli/src/lib/drain-background.ts`)
- **WebSocket Protocol:** Subscription-based real-time updates with keepalive. Types in `shared/src/types/ws.ts` , server in `server/src/ws/` , client in `cli/src/lib/ws-client.ts` .
- **Git Hooks Prompt System:** Auto-prompts users to install git hooks on first CC session in a new workspace. Uses `pending_git_hooks_prompt` / `git_hooks_prompted` columns on `workspace_devices` , exposed via `/api/prompts/pending` and `/api/prompts/dismiss` , consumed by CLI pre-action hook.
- **Structured Error Hierarchy:** `FuelCodeError` base with `ConfigError` , `NetworkError` , `ValidationError` , `StorageError` subclasses. (`shared/src/errors.ts`)